# Entwurf und Implementierung eines WTA- Netzes für Graphmatching unter Einbeziehung lokaler Ähnlichkeiten

Diplomarbeit

René Ejury

Studiengang Technische Informatik

TU Berlin

Betreuerin: Kristina Schädler

15.2.1998

# Zusammenfassung

Beim Vergleich von Graphen werden üblicherweise Identitäten einander zugeordnet und daraus Strukturen ermittelt, welche in beiden Graphen auftreten. Da im Alltag jedoch oft Ähnlichkeiten eine wichtigere Rolle als Identitäten spielen, wurde versucht, Graphvergleiche durch die Berücksichtigung lokaler Ähnlichkeiten zu verbessern.

Diese Arbeit beschreibt nun die Entwicklung eines Rückkopplungsnetzes für Graphvergleiche, wobei lokale Ähnlichkeiten im Netz abgebildet werden. Da diese Ähnlichkeiten lokal, das bedeutet auf der Ebene der Knoten- bzw. Kantenmarkierungen von Graphen bestimmt wurden, werden für die verschiedenen vorgestellten Merkmalsarten Ähnlichkeitsquantifizierungen und Generalisierungen entwickelt.

Die ermittelten Ähnlichkeiten müssen natürlich auf das Vergleichsnetz abgebildet werden- dafür werden mehrere Möglichkeiten entwickelt und die Änderungen des Netzes durch diese Möglichkeiten werden ausführlich beschrieben.

Die Implementierung des Netzes wird ausführlich beschrieben- sie erfolgte objektorientiert in C++ und bietet somit eine gute Grundlage für spätere Erweiterungen.

Die vorgesehenen Programmnutzungen - mit einer Programmierschnittstelle in C++ oder einer Bedienoberfläche werden ebenfalls vorgestellt. Abschließend wird die Qualität der verschiedenen Verfahren, lokale Ähnlichkeiten im Netz abzubilden, anhand von Testdatensätzen mit organischen Verbindungen beurteilt.

# Inhalt:

## TEIL 1:



## TEIL 2:

# Vorbemerkungen

Vor der eigentlichen Arbeit möchte ich mich an dieser Stelle bei allen bedanken, die durch ihre Hilfe und Unterstützung- alle auf ihre eigene Weise- dazu beigetragen haben, daß diese Arbeit heute vor mir liegt.

Als erstes will ich hier meinen Eltern, besonders meiner Mutter, danken. Ohne deren Unterstützung wäre ein Studium für mich nur sehr viel schwerer durchzuführen gewesen.

Besonderer Dank gilt natürlich der Assistentin, welche meine Diplomarbeit betreute. Kristina Schädler war immer offen für meine Probleme- die Besprechung von Fragen direkt zum Zeitpunkt der Entstehung half mir oft über die Sackgassen in meinen Gedanken hinweg.

Weiterhin möchte ich Gerhard Weber danken, der sich die Mühe machte, diese Arbeit als 'unabhängiger Kritiker' zu lesen und dem ich viele Hinweise auf sprachliche Hindernisse dieser Arbeit verdanke.

Und Dank gilt natürlich auch all jenen, die mich in der oft sehr anstrengenden Zeit diese Arbeit durch ihre Gegenwart und die Erledigung vieler Alltagsdinge unterstützten.

# ... anstatt einer Widmung ...

Diese Arbeit wurde von August 1997 bis Februar 1998 in der Bundesrepublik Deutschland geschrieben.

Aufgrund verschiedener Ereignisse in dieser Zeit sehe ich mich hier genötigt, alle Informatiker/ -innen oder sonstige Leser/-innen dieser Arbeit daran zu erinnern, daß es bei der Beschäftigung mit abstrakten Problemen leider oft zu leicht fällt, die gesellschaftliche Realität zu verdrängen. Und diese ist in der BRD (und nicht nur hier) leider auch heute noch (oder auch heute wieder) von rassistischer Gewalt und Ausländerfeindlichkeit geprägt.

Diese Arbeit soll also auch eine Mahnung sein- eine Mahnung, diese rassistische Gewalt nicht zu ignorieren sondern dafür zu kämpfen, daß alle Menschen auf dieser Welt das gleiche Recht auf Leben haben.

Auf der folgenden Seite sind einige Nachrichten zusammengestellt, welche während der Zeit der Erstellung dieser Arbeit in der Tagespresse zu finden waren* .

* alle Ausschnitte sind (gekürzte) Zitate aus der Wochenzeitschrift 'Jungle World' von Ausgust 1997 bis Februar 1998

# Beginn der Diplomarbeit: August 1997

Mit einer Bierflasche verletzte in Leipzig eine Gruppe junger Deutscher einen Mazedonier. Zuvor hatten die sechs Deutschen ihn mit ausländerfeindlichen Parolen beschimpft. In Floß bei Weiden in der Oberpfalz haben Unbekannte auf einem jüdischen Friedhof 44 Grabsteine umgeworfen oder beschädigt. Ein Kapitänleutnant der Reserve der Bundesmarine wurde zum Matrosen der Reserve degradiert, weil er auf einer Atlantiküberfahrt gesagt hatte: "Alles, was nicht arisch ist und in Deutschland lebt, gehört erschossen oder in die Gaskammer." Sieben indische Asylbewerber wurden am 4. September im sächsischen Pirna von jugendlichen Rechtsradikalen attackiert. Die 13 bis 18 Jahre alten Angreifer zeigten den Hitlergruß, brüllten Parolen und prügelten mit Ledergürteln auf die Flüchtlinge ein, von denen einer verletzt wurde. Im Ostberliner Bezirk Pankow wurde in der Nacht des 8. September ein Brandanschlag auf ein Ausländerwohnheim verübt. Die Brandsätze prallten jedoch ab und verbrannten auf dem Rasen. Drei Polizeibeamte aus Frankfurt/Main haben vor dem dortigen Landgericht die Mißhandlung von Ausländern gestanden. Sie hatten die Festgenommenen mit Fäusten und Schlagstöcken traktiert; einer der Angeklagten, die in erster Instanz zu Haftstrafen zwischen 27 und 36 Monaten verurteilt wurden, hat einem Gefangenen die Dienstpistole in den Mund gehalten, "um ihm einen Schreck einzujagen". Die Waffe sei aber "nicht durchgeladen" gewesen. Bereits zum siebten Mal innerhalb weniger Monate ist es im Lübecker Stadtteil St. Jürgen zu Anschlägen Rechtsradikaler auf kirchliche Einrichtungen gekommen. In der Nacht zum 17. September hinterließen die Täter an zwei Stellen Hakenkreuz-Schmierereien und den Namen des evangelischen Pastors Günter Harig. Harig gewährt einer Flüchtlingsfamilie Kirchenasyl. In der Nacht zum 20. September schlugen zwei junge deutsche Männer mit Flaschen auf zwei Asylbewerber ein. Die Asylbewerber wurden bei dem Überfall in der Fußgängerzone des vogtländischen Reichenbach schwer verletzt. Im sächsischen Radeburg sammelten sich am 20. September 15 Rechtsradikalen zum Überfall auf einen türkischen Wirt. Sie riefen rechtsradikale Parolen und warfen Flaschen. Drei junge Männer schlugen am 21. September in sächsischen Hoyerswerda brutal auf zwei Obdachlose ein. Unbekannte sprühten am 28. September Hakenkreuze und SS-Runen an eine Kirche im thüringischen Gotha. Großflächig mit Hakenkreuzen und Naziparolen wurde in den Nacht zum 1. Oktober eine Kirche und ein Friedhof im Brandenburgischen Jüterbog beschmiert. Der geständige 17jährige Täter gab Frust und Ausländerfeindlichkeit als Motiv an. Lebensgefährliche Verletzungen fügten drei Rechtsextreme am 26. September einem Weimarer Lebensmittelhändler zu. Die drei 16, 17 und 23 Jahre alten Deutschen waren mit dem Bierpreis, den ihnen der aus Vietnam stammende Händler nannte, nicht einverstanden. Der 16jährige stach in der folgenden Auseinandersetzung mit einem Messer zu und verletzte den Vietnamesen an Leber und Niere. Mit Baseball-Schlägern haben in Fahrland sechs bis zehn Skinheads einen Ungarn zusammengeschlagen. Sie forderten zunächst Geld von dem 32jährigen und verletzten ihn dann schwer. Auf die Zentrale Aufnahmestelle für Asylbewerber im sächsischen Chemnitz wurde in der Nacht zum 8. Oktober ein Brandanschlag verübt. Die von rund 400 Flüchtlingen bewohnte Unterkunft mußte geräumt werden. Im brandenburgischen Prenzlau überfielen am 8. Oktober zwei Jugendliche den Besitzer eines vietnamesischen Imbisses. Die beiden Rechtsradikalen beschimpften den Vietnamesen zunächst mit Parolen wie "Fidschis raus"; als er sich zur Wehr setzte, griffen sie ihn mit einer Flasche an und fügten ihm eine Wunde an der Hand zu. Der Vietnamese wehrte sich mit seinem Kochmesser und verletzte einen der beiden Angreifer. Mit Holzlatten attackierten Jugendliche am 14. Oktober einen türkischen Imbiß in Burg bei Magdeburg. Die vermummten Täter demolierten die Einrichtung und brüllten ausländerfeindliche Parolen. Sechs Tatverdächtige wurden festgenommen. Zwei mit Benzin gefüllte brennende Brandflaschen wurden in der Nacht zum 19. Oktober auf ein Asylbewerberheim in Trassenheide geworfen. Die Bewohner konnten den entstanden Brand löschen, verletzt wurde daher niemand. Als Tatverdächtige wurden wenig später in dem Badeort Zinnowitz zwei 18jährige festgenommen. Mit einem Kampfhund machte im brandenburgischen Eberswalde ein Gruppe junger Deutscher am 22. Oktober Jagd auf einen Angolaner. Zunächst beschimpften sie ihn mit rassistischen Sprüchen und schlugen auf ihn ein. Anschließend hetzten sie zwei Kampfhunde auf den 35jährigen. Er erlitt Prellungen und Bißwunden. In ein Taxi konnte sich in Halle am 22. Oktober ein marokkanischer Student retten. Er wurde von einer Gruppe mit Schlagstöcken bewaffneter Jugendlicher attackiert. Wie die Jüdische Gemeinde mitteilte, stürzten Unbekannte in dem Teil des Friedhofs Berlin-Weißensee, in dem während der Shoa bestattet wurde, drei Grabsteine um. Andreas Nachama, der Vorsitzende der Jüdischen Gemeinde zu Berlin, stellte fest, daß "die Bereitschaft zu Taten, die durch Judenhaß motiviert sind", bedrohlich steige. Ein polnischer Student der Europa-Universität Viadrina in Frankfurt/Oder ist am 31. Oktober aus einer Gruppe von vier deutschen Jungmännern heraus geschlagen worden. Er erlitt eine Platzwunde am Kopf und mußte stationär behandelt werden. Die kurzgeschorenen rechten Jugendlichen hatten zuvor den Studenten und einen Kommilitonen als "Zecken" beschimpft. Ein chinesischer Asylbewerber ist in der Nacht zum 7. November in Frankfurt/Oder angegriffen worden. Drei Männer schlugen ihn an einer Bushaltestelle ins Gesicht. Ein 53jähriger Grieche ist am 10. November in Hennigsdorf nördlich von Berlin niedergestochen und dabei von Passanten mit ausländerfeindlichen Parolen beschimpft worden. Ein 27jähriger Hennigsdorfer habe den Mann vor einem Restaurant mit dem Messer angegriffen, teilte die Oranienburger Polizei mit. Im brandenburgischen Bernau haben drei Rechtsradikale einen Russen mißhandelt. Zwei minderjährige Jugendlichen bedrohten ihn mit Gaspistolen und jagten ihn schließlich zusammen mit einem 26jährigen durch die Stadt. Der 26jährige prügelte solange auf einen der Ausländer ein, bis er zu Boden ging, anschließend traten die 17- und 15jährigen auf ihn ein. Als der Russe zu fliehen versuchte, schoß der 17jährige zweimal mit der Gaspistole auf ihn. Wie die Staatsanwaltschaft Frankfurt/Oder mitteilte, ereignete sich der Überfall bereits am 8. November. Gegen die beiden älteren Täter wurde Haftbefehl erlassen, der jüngere wurde in ein Jugendheim eingewiesen. Mit dem Hitlergruß und ausländerfeindlichen Parolen haben Skinheads am 26. November eine ukrainische Reisegruppe in der KZGedenkstätte Sachsenhausen beleidigt und beschimpft. Zuvor hatten die Skins vor dem Eingang zur Gedenkstätte ein Fernsehteam aus England angepöbelt. Fünf rechte Jungmänner beschimpften und schlugen am 22. November im Zug von Belzig nach Berlin-Wannsee den 28jährigen Ghanaer Martin Agyare. Nach Ermittlungen der Potsdamer Polizei riefen die BSC-Fans "Nigger raus" und "Deutschland den Deutschen". Als sie begannen, auf Agyare einzuschlagen, schoß dieser mit einer Schreckschußpistole auf einen 17jährigen Angreifer. Agyare war bereits 1994 überfallen und aus der S-Bahn geworfen worden. Er überlebte schwer verletzt, sein linker Unterschenkel mußte amputiert werden. Die Täter, nach Aussagen des Opfers sechs Skinheads, wurden nie gefunden. Zehn junge Deutsche schlugen am 21. November in Halle auf zwei indische Staatsangehörige ein und beschimpften sie dabei mit rassistischen Parolen. Die beiden Inder erlitten Kopfverletzungen. In Berlin schlug und trat am 25. November ein 26jähriger Deutscher auf zwei peruanische Fahrgäste in einem Nachtbus ein. In Lebensgefahr schwebte ein 30jähriger Türke zeitweise, nachdem er am 11. Dezember in Eberswalde bei Berlin zusammen mit einem deutschen Begleiter von rechten Skinheads angegriffen worden war. Auf ein Aussiedlerheim in Schönerlinde, in dem neben 27 Aussiedlern aus Ost- und Südosteuropa auch 21 jüdische Zuwanderer aus Rußland untergebracht sind, wurde am 13. Dezember ein Brandanschlag verübt. Unbekannte Täter schleuderten nachts gegen 2.30 Uhr eine Brandflasche durch das Fenster eines Unterkunftsraums. Ein Teppich fing Feuer. Wie die Polizei in Eberswalde mitteilte, konnten drei Bewohner, die durch das Scherbenklirren aufgewacht waren, den Brand selbst löschen. Rechtsextremistische Täter vermutet die Polizei hinter einem Brandanschlag auf einen moslemischen Gebetsraum und ein türkisches Bistro in Wasserburg am Inn. Im Keller des unbewohnten Gebäudes, in dem sich beide Einrichtungen befinden, fand die Polizei Spuren von Brandbeschleuniger. Gleich zweimal wurde in der Nacht zum 13. Dezember ein Asylbewerberheim in Ladebow bei Greifswald überfallen. Acht bis zehn mit einer Axt bewaffnete Jugendliche drangen in das Haus ein und skandierten "Sieg Heil". Sie flohen, als Alarm ausgelöst wurde, kehrten aber nochmals zurück, als die Polizei wieder fort war. Mit einem Schädelbasisbruch wurde am 22. Dezember ein Chinese in ein Berliner Krankenhaus eingeliefert. Zehn jugendliche Skinheads hatten ihn in der S-Bahn mit einer Stahlrute angegriffen und schwer verletzt. Kurz darauf mißhandelten die Skins eine 15jährige Punkerin. Am selben Tag gelang in Magdeburg einem Iraker, der von zwei Rechten in der Straßenbahn angegriffen worden war, die Flucht. Bei einem Brandanschlag auf ein überwiegend von Türken bewohntes Haus in Ludwigsburg wurden in der Nacht zum 25. Dezember fünf Menschen verletzt. Unbekannte hatten die Scheibe der Eingangstür eingeschlagen und im Treppenhaus Benzin angezündet. "Nigger und Kanaken raus", grölten 25 Neonazis, als sie am 28. Dezember eine Gaststätte im pfälzischen Kirchheimbolanden überfielen. So kam es vor der Gaststätte zu einer Massenschlägerei, bei der ein US-Amerikaner und ein Nazi schwer verletzt wurden. 21 Skinheads wurden festgenommen. Ebenfalls am 28. Dezember bedrohten Jugendliche in Oranienburg einen indischen Gastwirt. Der 37jährige sei von der rund 30köpfigen Gruppe beschimpft worden, so ein Polizeisprecher. Nach ihrer Festnahme zeigten die Männer und Frauen den Hitlergruß und riefen "Sieg Heil". Einen Gedenkstein der Jüdischen Gemeinde beschädigten Unbekannte in Berlin am Morgen des 30. Dezember. In Mahlow bei Potsdam griffen zehn Jugendliche vier Türken an. Sie beschimpften die Türken mit ausländerfeindlichen Parolen und schlugen sie mit Flaschen und Stöcken. Eine 29jährige Frau und ein 21jähriger Mann erlitten leichte Verletzungen. In Eberswalde wehrte sich ein türkischer Koch mit einem Messer gegen zehn rechte Jugendliche, die ihn in seinem Imbißstand angriffen. Als ihm ein 24jähriger eine Schreckschußpistole an den Kopf hielt, schnitt ihm der Koch unter anderem einen Finger ab. Während die rechten Schläger auf freiem Fuß blieben, wurde der Koch wegen Körperverletzung festgenommen. In Brand-Erbisdorf wurden zwei Rußlanddeutsche von Neonazis angegriffen und verletzt. Im niedersächsischen Stadthagen stachen zwei Skinheads einen 21jährigen Türken nieder. Das Opfer wurde mit drei Messerstichen ins Krankenhaus eingeliefert. In einem Behindertenheim in Sangerhausen überfielen sieben rechtsradikale Jugendliche eine Silvesterparty. Zwei Taubstumme und einer ihrer Betreuer wurden verletzt. In Niesky warfen Skinheads Feuerwerkskörper in eine türkische Gaststätte und riefen Parolen wie "Ausländer raus". In der Silvesternacht haben rechte Skinheads in Waren an der Müritz zwei Rußlanddeutsche angegriffen und schwer verletzt. Wie erst mehr als eine Woche später bekannt wurde, traten die Skinheads den beiden 18jährigen Spätaussiedler mit Stiefeln ins Gesicht und prügelten sie bis zur Bewußtlosigkeit. Die Aussiedler mußten mehrfach operiert werden. In einer Nürnberger Straßenbahn haben zwei Deutsche auf zwei Männer aus Sri Lanka eingetreten und sie beraubt. Ein sechsjähriger Junge starb und eine schwangere Frau verlor ihr Kind, als in der Nacht zum 14. Januar in einem Wohnhaus in der Innenstadt von Bielefeld ein Brand gelegt wurde.

# Ende der Diplomarbeit: Februar 1998

## Gegen das Verdrängen einer traurigen Realität !
## Gegen Rassismus !

# 1 Problemstellung

Beim rechnergestützten Vergleich von strukturierten Objekten aus unserer Umwelt treten immer wieder Probleme auf - während wir Menschen sofort gleiche Strukturen in verschiedenen Objekten erkennen, bedeutet der Versuch der rechnerischen Ermittlung die Lösung von NP-harten Problemen, also nicht mit sinnvollem Aufwand zu lösenden Aufgaben.

Ein Ausweg bietet an dieser Stelle die Abbildung des Matchingproblems auf ein Rückkopplungsnetz, wobei versucht wird, über lokal konsistente Variablenbelegungen zu einer globalen Lösung des Problems zu kommen.

Ein solches Netz bildet Constraints auf excitatorische und inhibitorische Kanten ab. Der stabile Endzustand der Propagation des Netzes stellt dann die Lösung des Matchingproblems dar - alle aktiven Neuronen repräsentieren Zuordnungen von Elementarobjekten der verglichenen Originale. Das zugrundeliegende Prinzip, nach welchem Neuronen mit großer Unterstützung die zugehörigen Neuronen 'mitziehen' und sämtliche nicht passenden Lösungen behindern, kommt auch im Namen dieser Netze zum Ausdruck - Winner takes all- Netze (WTA).

Dennoch sind mit der Nutzung dieser Netze nicht alle Probleme des Vergleichs strukturierter Objekte gelöst. Bei solchen Vergleichen werden zuerst die realen Objekte und ihre Beziehungen untereinander in markierte Graphen abgebildet. Ist diese Abbildung erfolgt, werden üblicherweise Knoten und Kanten, welche identische Markierungen haben, einander zugeordnet und im Kompatibilitätsgraph vermerkt, welcher wiederum die Basis für die Bildung des WTA-Netzes darstellt.

Beim menschlichen Vergleichen strukturierter Objekte werden nun jedoch nicht nur identische Objekte einander zugeordnet, vielmehr spielen Ähnlichkeiten eine wichtigere Rolle. Bestimmte Merkmale werden geringer gewichtet als andere, und falls Übereinstimmung in einer Kernstruktur gefunden wurde, wird diese durch ähnliche Strukturen erweitert, bis sinnvolle Ergebnisstrukturen gefunden werden. Ich spreche hier von sinnvollen Ergebnisstrukturen, weil die Gewichtung der Merkmale sowie die Beurteilung des Ergebnisses natürlich stark vom Problem abhängig ist.

Meine Aufgabe war nun, die lokalen Ähnlichkeiten strukturierter Objekte bei der Bildung des Kompatibilitätsgraphen und damit natürlich auch bei der Bildung des WTA- Netzes zu berücksichtigen, damit nicht nur identische Zuordnungen im Ergebnis auftreten konnten.

Um dies zu realisieren, war zu allererst eine Quantisierung von Ähnlichkeiten erforderlich (Abschnitt 2.) und danach mußten Möglichkeiten gefunden werden, diese Ähnlichkeiten im Netz abzubilden (Abschnitt 3.). Um sinnvolle Ergebnisse zu erhalten, mußten natürlich Generalisierungsvorschriften erarbeitet werden, mit denen ähnliche Objektmarkierungen zu Ergebnismarkierungen gewandelt werden konnten.

Nach diesen theorhetischen Vorarbeiten konnten Konzepte für die Implementierung entwickelt (Abschnitt 5.) und umgesetzt (Abschnitt 6.) werden.

Zur Nutzung des implementierten Programms wurden zwei verschiedene Wege vorgesehen. Zum einen sollte das Programm durch eine interaktive, menügestützte Oberfläche leicht bedienbar sein, zum anderen sollte eine Schnittstelle in C++ die Möglichkeit bieten, die programmierten Algorithmen in eigene Programme einzubinden (Abschnitt 8.).

Da eine Bewertung der diversen Möglichkeiten der Ähnlichkeitsabbildung nur schwer möglich ist, wurden diese abschließend anhand eines Testdatensatzes kurz bewertet, indem die ermittelten Ähnlichkeiten zu Aufbau von Klassifikatoren genutzt wurden (Abschnitt 10.).

# 2 Merkmalsvergleiche

## 2.1 Graphen und Markierungsstruktur

Um bestimmte Informationen aus unserer Umwelt miteinander zu vergleichen, muß dieses Wissen im Rechner adäquat repräsentiert werden, da nur eine solche Abbildung für rechnergestützte Vergleichsprozesse nutzbar ist. Diese Abbildung sollte alle relevanten Informationen vollständig widerspiegeln, jedoch andererseits möglichst einfache, leicht auf Computern zu verarbeitende Strukturen nutzen.

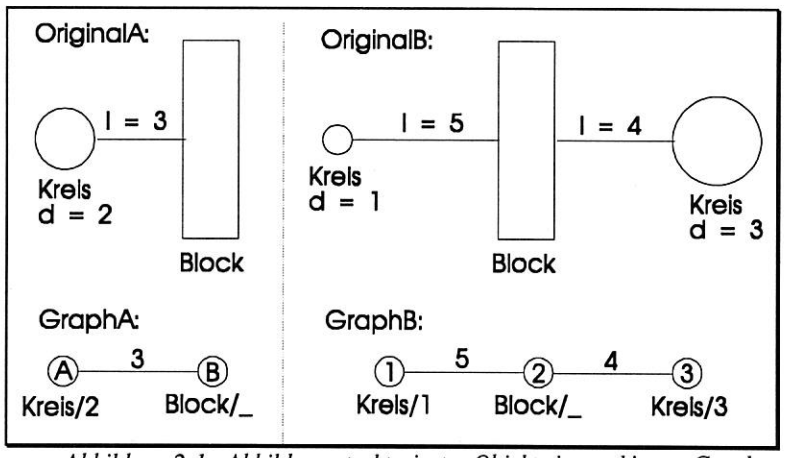



*Abbildung 2-1 - Abbildung strukturierter Objekte in markierten Graphen*

Eine Möglichkeit der Wissensrepräsentation ist die Abbildung strukturierter Objekte auf relationale Beziehungen, welche wiederum in (markierten) Graphen anschaulich dargestellt werden können (Abbildung 2-1).

Ein markierter Graph sei durch G = (N, V, f, g, M, W) definiert mit

N: Knotenmenge, V: Kantenmenge ($V \subseteq N \times N$)

M: Knotenmarkierungen, W: Kantenmarkierungen

f,g: Knoten- bzw. Kantenmarkierungsfunktionen, f: $N \rightarrow M$, g: $V \rightarrow W$.

Da das Programm nicht für ein spezielles Problem entwickelt werden sollte, war es nötig, *genau diese* allgemeine Graphbeschreibung im Programm abzubilden. So werden Mengen von Knoten und Kanten verwaltet, welchen jeweils Markierungen zugeordnet sind. Um Markierungen möglichst universell zu gestalten, andererseits aber die Vergleichbarkeit sicherzustellen, sind die Markierungen beliebig lange Merkmalsvektoren aus elementaren Merkmalen.

Die folgenden Abschnitte sollen beschreiben, welche elementaren Merkmalsarten für die Beschreibung von (allgemeinen) Objektmarkierungen notwendig waren und wie die Generalisierung und die Quantifizierung der Ähnlichkeiten bei diesen Elementarmerkmalen vorgenommen werden kann.

## 2.2 Objekte und Konzepte

Zur deutlicheren Unterscheidung der verschiedenen Arten von Merkmalsausprägungen ist es an dieser Stelle nötig, einige Begrifflichkeiten zu erläutern.

Stellen wir uns einmal den Vergleich zweier Zahlen vor, zum Beispiel der Zahlen 3 und 7. Ohne jetzt genauer darauf einzugehen, wie diese Ähnlichkeit quantifiziert werden kann, soll hier überlegt werden, wie denn das generalisierte Ergebnis aus diesen beiden Ausprägungen aussehen könnte.

Als einfachste Lösung scheint hier das arithmetische Mittel angebracht, also 5. Allerdings würde ein Vergleich mit der Merkmalsausprägung 5 jetzt maximale Ähnlichkeit bescheinigen, obwohl es sich ja nicht um 5 handeln soll, sondern um das Resultat aus dem Vergleich von 3 und 7, was wohl weniger genau der Ausprägung 5 ähnlich sein soll als die 5 sich selbst.

Aus dem eben geschilderten Grund erschien es nötig, als Ergebnis des Vergleichs von 3 und 7 nicht 5, sondern {3,7} anzugeben, also eine Menge aus mehreren (im Beispiel den bisher verglichenen) Zahlen.

Wie wir aber an dieser Stelle sehen können, kann unter bestimmten Bedingungen ein Unterschied zwischen der Ergebnismenge selbst ({3,7}) und der Beschreibung dieser (5) bestehen, da ja auch der Mittelwert der Menge eine Beschreibung derselben darstellt.

Darum soll die *Beschreibung* der Ergebnismenge (ob durch Aufzählung der Elemente oder durch andere beschreibende Methoden) im folgenden als Konzept bezeichnet werden. Im Gegensatz dazu stehen die Ausgangsmerkmalsausprägungen als Objekte.

## 2.3 Generalisierung von Merkmalsausprägungen

Zur Beschreibung von Markierungen, welche uns im Alltag konfrontieren und somit in irgendeiner Form in einen Graphen abgebildet werden können, wurde eine Unterscheidung in fünf Hauptgruppen vorgenommen, aus denen (fast) alle existierenden Merkmale zusammengesetzt werden können[1].

| Merkmalsart |
|---|
| reellwertig |
| ganzzahlig |
| Element einer geordneten Menge |
| Symbole |
| Begriff aus einer Hierarchie |

*Tabelle 2-1 - Arten der Merkmale*

Zum einen gibt es Merkmale, welche sich direkt durch Zahlen abbilden lassen und somit für die Quantisierung leicht faßbar sind. Zum anderen kann das Merkmal ein Element aus einer geordneten Menge sein, es kann ein Begriff aus einer Begriffshierarchie vorliegen oder einfach dieser Begriff als Symbol betrachtet werden, welches nur zu sich selbst ähnlich ist. Natürlich wird oft erst durch die Kombination dieser Merkmale eine originalgetreue Beschreibung möglich.(Tabelle 2-1)

Eine Generalisierung von Merkmalen soll allgemein immer eine Obermenge der generalisierten Merkmale beschreiben. Zusätzlich ist es wichtig und sinnvoll darauf zu achten, daß das generalisierte Ergebnis eine gleiche oder zumindest ähnliche Qualität wie die zu generalisierenden Merkmale hat.

In den folgenden Abschnitten werden für alle in Tabelle 2-1 genannten Merkmalsarten Generalisierungen entwickelt. Darauf aufbauend wird erarbeitet, ob es bei den entsprechenden Merkmalen wichtig ist, Unterschiede zwischen Konzepten und Objekten zu berücksichtigen.

Danach werden für alle Merkmalsarten Quantifizierungsverfahren für Vergleiche beschrieben, falls Konzepte und Objekte verschiedene Qualität besitzen wird natürlich an dieser Stelle auch die Ähnlichkeitsquantifizierung von Konzepten beschrieben.

[1] Sollten diese Merkmalsarten für eine bestimmte Anwendung nicht ausreichen, so kann das System aufgrund der objektorientierten Implementierung einfach erweitert werden.

### 2.3.1 Generalisierung reellwertiger Merkmale

Stellen wir uns an einem Beispiel die Generalisierung von zwei reellen Zahlen, 3.5 und 5.5 vor. Als Ergebnis der Generalisierung bietet sich hier also eine Menge mit beiden Elementen an (siehe Abschnitt 2.2). Dadurch wird gleichzeitig festgelegt, daß, da die Ergebniskonzepte sich von den Ausgangsobjekten unterscheiden (Menge vs. Element), solche auch als Basis des Vergleichs berücksichtigt werden müssen.

Sollen die generalisierten Beschreibungen irgendwann wieder auf ein Merkmal gleicher Qualität zurückgeführt werden, können statistische Verfahren genutzt werden. Jedoch ist dies nur möglich, wenn immer *alle* betrachteten Elemente in einem Multiset gesammelt werden, also auch doppelte Elemente aufgehoben werden.

Ist eine solche 'Rücktransformation' auf eine einzelne Zahl nicht erwünscht, oder kann davon ausgegangen werden, daß statistische Verteilungen die Elemente im Lösungskonzept nicht beschreiben, kann auf die Registrierung von doppelten Elementen in der Menge verzichtet werden. Für Vergleiche bieten sich in diesem Fall zum Beispiel Verfahren aus dem 'fuzzy'- Bereich an.

Diese Trennung im Umgang mit reellwertigen Merkmalsausprägungen wird auch in zwei verschiedenen implementierten Verfahren reflektiert.

#### *2.3.1.1 statistisches Verfahren*

Das hier geschilderte Verfahren geht davon aus, daß die zur Generalisierung 'angebotenen' Elemente normalverteilt sind. Die zu generalisierenden Elemente beschreiben somit eine Verteilung, während diese in ihrer Gesamtheit beim 'fuzzy'- Ansatz lediglich als Zugehörigkeitsfunktion zu betrachten sind.

Als Generalisierung werden also alle Zahlen der beiden Ausgangskonzepte oder -objekte in einem Multiset gesammelt. So ist immer auch eine Rekonstruktion einer reellen Zahl aus dieser Generalisierung durch eine Berechnung des Mittelwertes möglich. (siehe auch Quantifizierung Abschnitt 2.4.1.2)

#### *2.3.1.2 fuzzy Verfahren*

Soll die oben beschriebene Generalisierung keine Normalverteilung widerspiegeln, so reicht es an dieser Stelle aus, als Generalisierung jeweils eine Menge aller Zahlen der Ausgangsobjekte bzw. -konzepte zu bilden. Die Generalisierung enthält also alle generalisierten Zahlen. (siehe auch Quantifizierung Abschnitt 2.4.1.3)

Zusätzlich erscheint es bei dieser Generalisierung sinnvoll, bei Unterschreitung bestimmter Abstände zwischen den betrachteten Zahlen automatisch Intervalle zu bilden. So kann die Generalisierung an den gewünschten oder erwarteten Lösungsbereich angepaßt werden, da nicht alle Differenzen zwischen Zahlen als solche betrachtet werden (Abbildung 2-2).

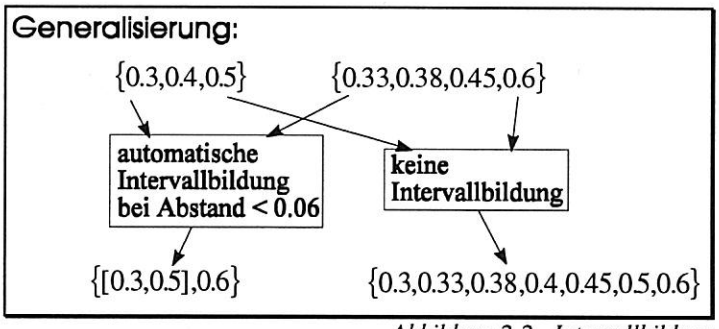



*Abbildung 2-2 - Intervallbildung*

### 2.3.2 Generalisierung ganzzahliger Merkmale

Ganze Zahlen sind als Teilmenge der reellen Zahlen natürlich auf genau dieselbe, eben beschriebene Weise generalisierbar.

Dies bedeutet, daß auch bei ganzen Zahlen Konzepte als Generalisierung auftreten können, welche von den generalisierten Objekten verschieden sind und somit zum Vergleich zugelassen werden müssen. Obwohl sich bei einem statistischen Verfahren nicht sicherstellen läßt, daß sich die Ergebnisverteilung im Multiset auf eine ganze Zahl reduzieren läßt, soll dieses Verfahren auch hier die Generalisierungsart, bei welcher alle Elemente in einer Menge (ohne doppelte Zahlen) gesammelt werden, ergänzen.

### 2.3.3 Generalisierung geordneter Mengen

Um den Ablauf der Generalisierung von geordneten Mengen zu verstehen, soll hier kurz die Interpretation dieser erläutert werden.

Eine geordnete Menge läßt sich immer auf eine andere geordnete Menge abbilden, falls die Anzahl der Elemente in beiden gleich ist, wobei Elemente mit gleichen Indizes einander zugeordnet werden.

$$A = \{x_1, x_2, \ldots, x_n\}$$
$$B = \{y_1, y_2, \ldots, y_m\} \qquad (2\text{-}1.)$$
$$C = \{(x,y)_1, (x,y)_2, \ldots, (x,y)_n\} \text{ mit } (x,y)_i = (x_i, y_i), \text{ wenn } n = m$$

So läßt sich natürlich jede geordnete Menge auf die Menge der ganzen Zahlen von 1 bis zur Anzahl der Elemente abbilden (und eigentlich sind die Indizes, welche die Ordnung der Mengen in den oberen Formeln verdeutlichen sollen, bereits eine solche Abbildung). Zum Vergleich und zur Generalisierung von Elementen einer geordneten Menge werden darum nicht die Elemente selbst genutzt, sondern immer deren Abbild auf ganze Zahlen (bzw. deren Indizes).

Die Generalisierung erfolgt daher genau wie bei ganzen Zahlen mit eben der dort genannten Unterscheidung in die Registrierung der Elemente in einer Menge oder einem Multiset.

Wie bei ganzen Zahlen sind darum auch hier, um dies nochmals explizit zu erwähnen, die Ergebniskonzepte als Mengen von einer anderen Qualität als die Vergleichsobjekte (Elemente).

### 2.3.4 Generalisierung von Symbolen

Sollen verschiedene Symbole generalisiert werden, so besteht lediglich die Möglichkeit, alle zu generalisierenden Einzelsymbole in einer Menge zu sammeln.

Andere Methoden, welche zum Beispiel statistische Verfahren nutzen, wurden an dieser Stelle nicht berücksichtigt, da die zu wählenden statistischen Vorgehensweisen sehr stark vom Problem abhängig wären.

### 2.3.5 Generalisierung von Begriffen einer Hierarchie

Auch die Generalisierung von Begriffen einer Hierarchie soll an dieser Stelle beschrieben werden.

Offensichtlich ist hierbei, daß der Oberbegriff dieser beiden Elemente genau diese Generalisierung der beiden Begriffe darstellt. Natürlich können dabei Oberbegriff und zu vergleichende Begriffe in beliebiger Hinsicht identisch sein, fest steht nur, daß das Ergebnis der Generalisierung schon aufgrund der Ordnung in der Hierarchie ebenfalls ein Begriff derselben Hierarchie sein muß, also dieselbe Qualität hat.

Da der Oberbegriff zweier Begriffe auch ein einzelner Begriff ist, sind bei dieser Merkmalsart Konzepte als Ergebnis nicht gesondert zu betrachten.

| Merkmalsart | Verfahren | Generalisierung | Konzept-Objekt-Unterschiede | Ergebnisqualität |
|---|---|---|---|---|
| reell | statistisch | Multiset aller Zahlen | ja | reelle Zahl |
| reell | fuzzy | Menge aller Zahlen | ja | Menge aller Zahlen |
| ganz | statistisch | Multiset aller Zahlen | ja | ganze Zahl |
| ganz | fuzzy | Menge aller Zahlen | ja | Menge aller Zahlen |
| geordnete Menge | statistisch | Multiset aller Elemente | ja | ein (oder zwei) Element(e) |
| geordnete Menge | fuzzy | Menge aller Elemente | ja | Menge aller Elemente |
| Symbole | | Menge aller Symbole | ja | Menge aller Symbole |
| Begriffshierarchie | | Oberbegriff | nein | Begriff |

*Tabelle 2-2 - Generalisierung der Einzelmerkmale*

## 2.4 Quantifizierung von Merkmalsähnlichkeiten

Um Ähnlichkeiten zwischen Merkmalen in einem Computer zu verarbeiten, müssen diese zuerst auf Zahlen abgebildet werden. Diese recht simpel klingende Feststellung birgt jedoch einige Probleme in sich, da wir Menschen normalerweise Ähnlichkeit sehr einfach feststellen, ja sogar den Grad der Ähnlichkeit ganz gut einschätzen können. Sollen diese Ähnlichkeitswerte nun auf exakte Zahlen zwischen 0 (maximal unähnlich) und 1 (identisch) abgebildet werden, treten Probleme auf. Zahlen sind zu fest, und Ähnlichkeiten als unscharfe, kontextabhängige Beziehungen auf feste Werte zu reduzieren bringt Schwierigkeiten mit sich - während eine Ähnlichkeit von 0.5 vielleicht noch verstanden werden kann, ist dies bei der Ähnlichkeit 0.762 schon etwas komplizierter.

Warum diese Erklärung?

Da ich die Merkmalsquantifizierung nicht für eine bestimmte Aufgabenstellung vornehmen sollte, mußte ich versuchen, allgemeine (das heißt Alltags-) Ähnlichkeiten in Zahlen umzusetzen. Da für die Ähnlichkeit von Begriffen aus Begriffshierarchien zum Beispiel keine typischen Quantisierungsvorgaben existieren, spiegeln die verwendeten Verfahren natürlich auch immer *meine* Vorstellung der Anwendung dieser Vergleiche wieder. Sollten also einige der folgenden Verfahren für ein ganz spezielles Problem nicht geeignet sein, so kann ich nur empfehlen, die implementierten Versionen zu verändern.

### 2.4.1 Reellwertige Merkmale

Obwohl, wie bereits in Abschnitt 2.3.1 erwähnt, zwei Implementierungen für die Quantisierung von reellwertigen Merkmalsausprägungen existieren, unterscheiden diese sich lediglich im Umgang mit Konzepten. Sollen einzelne Zahlen miteinander verglichen werden, ist ein statistischer Ansatz nicht zu verwenden und auch das statistische Verfahren nutzt an dieser Stelle den im folgenden erklärten 'fuzzy'- Vergleich für Objekte.

#### *2.4.1.1 Objekt- Objekt- Vergleich:*

Sollen zwei Zahlen miteinander verglichen werden, steht fest, daß bei Gleichheit beider Zahlen die Ähnlichkeit mit 1 (also Identität) bestätigt werden muß.

Um aber Ähnlichkeit zum Ausdruck zu bringen, muß eine Zahl , welche nicht identisch der anderen ist, dieser trotzdem ähnlich sein (können). Diese Ähnlichkeit sollte sich verringern, je größer der Abstand beider Zahlen ist. Um den gesamten Vorgang überschaubarer zu halten und den Anwender bzw. die Anwenderin bei der Parameterwahl zu unterstützen, empfiehlt sich eine lineare Abnahme der Ähnlichkeit, wie sie mit folgender Formel im Programm implementiert wurde (HardOr - Vergleich nach [Lui 1992]):

$$\gamma = \gamma(x_a, x_b) = \varphi[1 - abs(\frac{x_a - x_b}{l})], \quad \varphi(x) = \begin{cases} 0 & x < 0 \\ x & \text{sonst} \end{cases} \qquad (2\text{-}2.)$$

Ein Beispielvergleich von $x$ mit 5 ist in Abbildung 2-3 dargestellt.

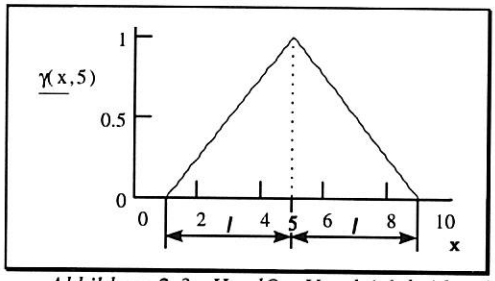



*Abbildung 2-3 - HardOr- Vergleich bei l = 4*

$l$ ist der Ähnlichkeitsabstand (im Beispiel l = 4), welcher bestimmt, auf welche reelle Entfernung die Ähnlichkeit der beiden Zahlen von 1 bis auf 0 absinkt. Dieser kann als Parameter dem Programm übergeben werden und muß also von der Nutzerin bzw. dem Nutzer selbst anhand des Problems gewählt werden. (siehe Parametertabelle 7.3.)

Die Umgangsweise mit Konzepten ist an dieser Stelle abhängig von der Betrachtung der Daten. Können diese als auf einen Wert weisend angenommen werden, so läßt sich aus allen Elementen des Konzeptes eine Normalverteilung ermitteln. Sind verschiedene Elemente aber auch als solche zu berücksichtigen, ist ein Vergleich über Fuzzy- Methoden eher angebracht.

### *2.4.1.2 Statistisches Vergleichsverfahren*

Bei diesem Verfahren werden alle Objekte (auch die innerhalb von Konzepten), welche jemals zum Vergleich angeboten wurden, im Vergleichsergebnis mitgeführt. Der Vorteil dieser Methode ist die mögliche Berechnung einer einzelnen Zahl aus allen Werten des Ergebnisses (z.B. Mittelwert), wobei *dann* das Resultat die gleiche Qualität hat wie die Ausgangsobjekte (einzelne Zahlen).

Ein weiterer Vorteil dieser Methode liegt in der Berücksichtigung der Gesamtverteilung der Elemente der Konzepte bei den Ähnlichkeitsquantisierungen - im Gegensatz zu 'fuzzy'- Methoden, bei welchen nur die näheren Objekte Einfluß auf das Ergebnis haben. Dies ist natürlich nur dann sinnvoll, wenn die Elemente auch als Verteilung betrachtet werden können.

#### 2.4.1.2.1 **Objekt- Konzept- Vergleich** (siehe [Clauss 1970])

Es sei $x_a$ das Objekt und $\{x_1, x_2, \ldots, x_n\}$ das Konzept, welche verglichen werden sollen.

Da die Elemente des Konzepts als auf eine Zahl hin weisend angenommen werden, werden aus dem Konzept zuerst der Mittelwert $\bar{x}$ und die Varianz $s^2$ berechnet, da angenommen wird, daß die Elemente im Konzept normalverteilt sind.

Berechnung der Varianz: $s^2 = \frac{1}{n-1} * \sum_{i=1}^{n} (x_i - \bar{x})$ (2-3.)

Nun sind ein Mittelwert eines Konzeptes und eine Zahl miteinander zu vergleichen. Wie eben schon angenommen wurde, soll das Konzept eine Normalverteilung beschreiben. Eine Ähnlichkeit kann damit mit der Wahrscheinlichkeit des Auftretens von $x_a$ in der durch Mittelwert und Varianz beschriebenen Verteilung beschrieben werden.

Berechnung der Ähnlichkeit: $y = e^{-\frac{1}{2}*\frac{(x_a-\bar{x})^2}{s^2}}$ (Normalverteilung, Abbildung 2-4) (2-4.)

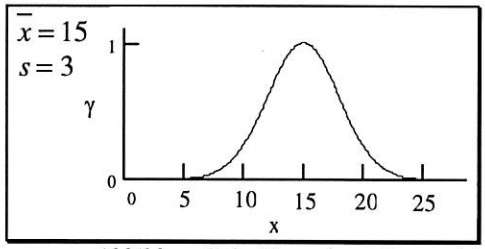



*Abbildung 2-4 - Normalverteilung, y(x)*

#### 2.4.1.2.2 Konzept- Konzept- Vergleich:

Es seien A und B zwei Konzepte, deren Mittelwerte ($\overline{x_A}, \overline{x_B}$) und Varianzen ($s_A^2, s_B^2$) berechnet wurden.

Fest steht, daß bei gleichen Verteilungen ($\overline{x_A} = \overline{x_B}$ und $s_A^2, = s_B^2$) die Identität der Konzepte mit 1 bestätigt werden muß. Sind die Verteilungsparameter verschieden, so soll die Ähnlichkeit abnehmen - weil diese Abnahme schon beim Objekt- Konzept- Vergleich (2.4.1.2.1) exponentiell erfolgte, ist es an dieser Stelle ebenfalls sinnvoll, eine exponentielle Abnahme der Ähnlichkeit vorzusehen.

Die eben beschriebenen Anforderungen lassen sich durch folgende Formel realisieren:

$$\gamma' = e^{-(\overline{x_A}-\overline{x_B})^2-(s_A-s_B)^2} \qquad (2\text{-}5.)$$

Ein weiteres Problem ist die Dimensionsabhängigkeit der ermittelten Ähnlichkeit. Es muß sichergestellt werden, daß gleiche Ähnlichkeiten errechnet werden, falls die Elemente in beiden Konzepten in eine andere Dimension verschoben (zum Beispiel verzehnfacht) werden. Dies konnte durch Skalierung der Mittelwertdifferenz mit der Summe der Standardabweichungen (Standardabweichung = $s = \sqrt{s^2} = \sqrt{\text{Varianz}}$) und umgekehrt erfolgen.

$$\gamma = e^{-\left(\frac{\overline{x_A}-\overline{x_B}}{s_A+s_B}\right)^2-200*\left(\frac{s_A-s_B}{x_A+x_B}\right)^2} \qquad (2\text{-}6.)$$

Der Faktor '200' bestimmt das Verhältnis des Einflusses der Differenzen der Mittelwerte zu dem der Differenzen der Standardabweichungen und wurde empirisch ermittelt (siehe Abbildung 2-5).

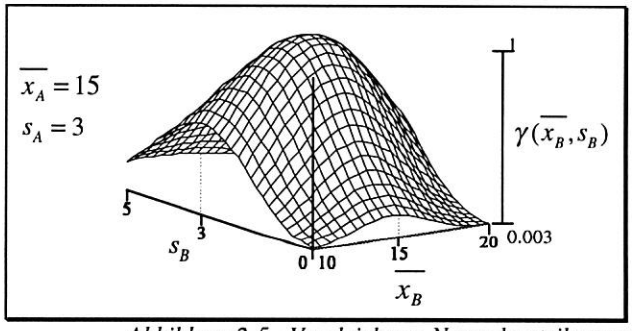



*Abbildung 2-5 - Vergleich von Normalverteilungen*

### *2.4.1.3 Fuzzy- Vergleich*

Ist anhand der Daten vorgegeben oder erwünscht, statistische Verfahren zur Ähnlichkeitsbestimmung auszuschließen, können die nun folgenden fuzzy- Verfahren (nach [LUI 1992]) genutzt werden.

#### 2.4.1.3.1 Objekt- Konzept- Vergleich:

Beim Objekt- Konzept- Vergleich ist an dieser Stelle festgelegt, daß, falls das Objekt bereits Element des Konzeptes ist, eine Ähnlichkeit von 1 festgestellt wird.

Dabei ist es wichtig, daß das Vergleichsergebnis von 1 hier nicht etwa Identität bescheinigt, sondern lediglich die größte bei einem solchen Vergleich mögliche Identität feststellt. Andererseits liegt dieser Gegensatz schon im Prinzip von Konzepten begründet, da nicht explizit festgelegt wurde (und werden sollte), ob diese eher als Sammlung von Einzelobjekten oder als Gesamtheit betrachtete werden sollen. Für fuzzy- Vergleiche sollen also im folgenden Konzepte als Sammlung derjenigen Einzelobjekte betrachtet werden, die mit diesem Konzept die Ähnlichkeit 1 haben.

Für den Fall, daß das Einzelobjekt nicht im Konzept auftritt, wird die folgende Berechnung genutzt, wobei für Vergleiche von Objekten wiederum die Berechnung nach Abschnitt 2.4.1.1 verwendet wird.:

Es sei $x$ das Objekt und $A = \{y_1,...\}$ das Konzept, welche verglichen werden sollen.

$$\gamma(\{y_1,y_2\},x) = \gamma(y_1,x) + \gamma(y_2,x) - \gamma(y_1,x) * \gamma(y_2,x)$$

$$\gamma(\{y_1,y_2,...\},x) = \gamma(y_1,x) + \gamma(\{y_2,...\},x) - \gamma(y_1,x) * \gamma(\{y_2,...\},x) \qquad (2\text{-}7.)$$

Diese Ähnlichkeitsberechnung wird in [LUI 1992] als Soft- Or beschrieben (Abbildung 2-6). Sie ergibt für den Fall, daß Objekte im Konzept enthalten sind eine Identität von 1, weshalb dieser Fall nicht gesondert betrachtet werden muß.

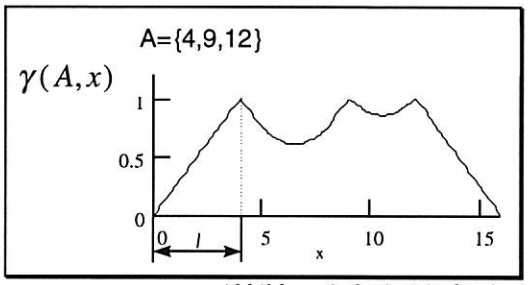



*Abbildung 2-6 - SoftOr bei l=4*

#### 2.4.1.3.2 Konzept- Konzept- Vergleich:

Es seien A und B die zu vergleichenden Konzepte. Dann kann eine Ähnlichkeit dieser Konzepte durch einen Flächenvergleich der eben beschriebenen Soft- Or Funktionen quantifiziert werden.

$$\gamma(A,B) = \frac{\int_{-\infty}^{\infty} \min(\gamma(A,x),\gamma(B,x))}{\max\left(\int_{-\infty}^{\infty}\gamma(A,x), \int_{-\infty}^{\infty}\gamma(B,x)\right)} \qquad (2\text{-}8.)$$

Die Fläche unter den minimalen Funktionswerten der Soft- Or Funktionen wird hierbei ins Verhältnis zur maximalen Fläche unter einer der beiden Soft- Or Funktionen für beide

Konzepte gesetzt. Diese Ähnlichkeit ist nur dann maximal, wenn beide Konzepte die gleichen Elemente aufweisen, da nur dann Dividend und Divisor gleich groß sind (Abbildung 2-7).

Die Flächenberechnung erfolgt programmintern durch lineare Approximation der Funktionen. Die Anzahl der Stützstellen der Berechnung in $l$ ist mit 5 voreingestellt und kann durch Parameter geändert werden (siehe Parametertabelle 7.3.).

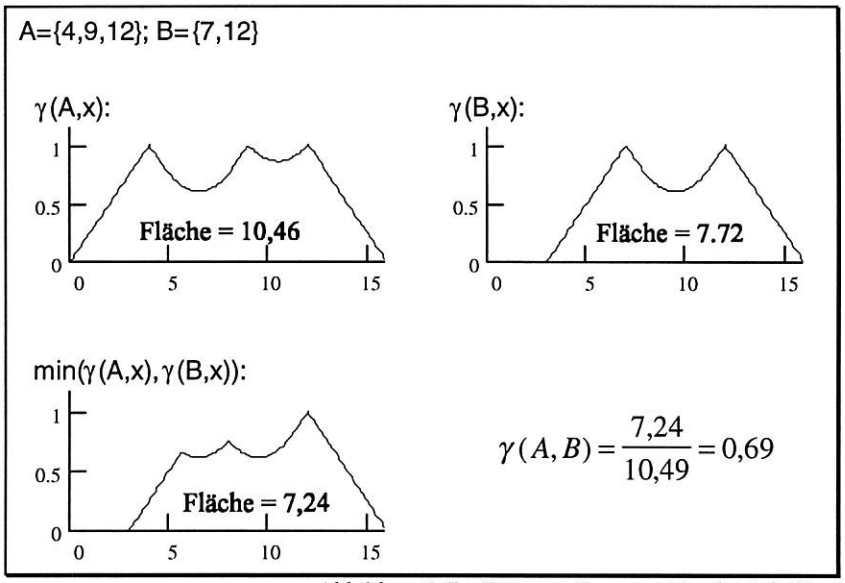



*Abbildung 2-7 - Konzept- Konzept- Vergleich bei l=4*

### 2.4.2 Ganzzahlige Merkmale

Die eben erläuterten Verfahren zum Quantifizieren der Ähnlichkeit von reellen Zahlen bzw. Konzepten eignen sich auch hervorragend, um die Ähnlichkeit ganzer Zahlen bzw. Konzepte zu ermitteln. Lediglich die Menge der möglichen Argumente der genutzten Berechnungsfunktionen verringert sich auf die Menge der ganzen Zahlen. Da die Zahlen als solche im Ergebnis weitergeführt werden, ist nicht einmal hierbei eine exklusive Berücksichtigung ganzer Zahlen nötig.

Im Programm wurde an dieser Stelle einfach eine template[2]- Klasse implementiert, welche einmal für ganze und einmal für reelle Zahlen compiliert wurde.

### 2.4.3 Quantifizierung der Ähnlichkeit von Elementen geordneter Mengen

Wie schon im Abschnitt über die Generalisierung (2.3.3.) von geordneten Mengen erläutert, lassen sich die Elemente der geordneten Menge immer ganzen Zahlen zuordnen.

Programmintern werden daher für die Repräsentation und den Vergleich der Elemente der geordneten Menge lediglich deren Indizes genutzt, bei einer Ausgabe werden den Indizes wieder die Elementbeschreibungen zugeordnet. Daher erfolgt der Vergleich von Elementen geordneter Mengen und dessen Quantifizierung genau wie der ganzer Zahlen, mit denselben Methoden und Vorschriften wie in (2.4.2). Da ganze Zahlen wiederum wie reelle Zahlen behandelt werden, sollten die Verfahren nach (2.4.1) ausreichend Informationen geben.

[2] template- Methoden (generische Methoden) bieten die Möglichkeit, Methoden, welche einmal für bestimmte Klassen geschrieben wurden, für (prinzipiell) alle Klassen zu nutzen, falls diese Methoden für die neuen Klassen compiliert werden und alle Methoden, welche innerhalb dieser Methoden auf die neuen Klassen zugreifen, auch für diese neuen Klassen existieren (zum Beispiel durch überlagerte Operatoren)

Aus dieser Abbildung auf ganze Zahlen ergibt sich natürlich auch, daß geordnete Mengen wiederum anhand statistischer Verfahren oder anhand fuzzy- Methoden verglichen werden können.

### 2.4.4 Merkmal symbol

Das hier beschriebene Vergleichsverfahren dient dem Vergleich beliebiger Symbole.

In diesem Falle gelten Symbole als gleich, wenn die Reihenfolge und Art der Zeichen in beiden zu vergleichenden Symbolen gleich ist. Weitere Zusammenhänge zwischen den Symbolen existieren nicht oder sind nicht bekannt (bzw. werden ignoriert) und können damit keinen Einfluß auf die Ähnlichkeit haben. Weil die als Vergleichsergebnis auftretenden Konzepte sich von den Objekten unterscheiden (2.3.4), müssen diese natürlich auch als Vergleichs'elemente' berücksichtigt werden.

Um hierbei den Gesamtablauf durchschaubarer zu machen, wird in den folgenden Abschnitten immer noch kurz die Generalisierung der Symbole wiederholt, weil diese natürlich immer Rückschlüsse auf die zu vergleichenden Elemente liefert.

#### *2.4.4.1 Objekt- Objekt- Vergleich:*

Da, wie eben erwähnt, keine weiteren Informationen über die Symbole berücksichtigt werden, kann beim Vergleich zweier Symbole nur auf Gleichheit oder Ungleichheit entschieden werden.

Falls die Symbole gleich sind, wird eine 1 als Ähnlichkeit ausgegeben, sonst eine 0. Das Ergebnis der Generalisierung ist ein Konzept mit beiden Symbolen, wenn diese unterschiedlich waren. Waren beide Objekte gleich, wird das Objekt als Generalisierung weitergegeben.

#### *2.4.4.2 Objekt- Konzept- Vergleich:*

Es sei s ein Objekt und A ein Konzept mit Symbolen. Dann wird eine Ähnlichkeit von 1 festgestellt, wenn s in A gefunden werden kann. Sonst beträgt die Ähnlichkeit 0. Das Ergebnis der Generalisierung ist ein Konzept mit allen Symbolen je einmal.

Dies entspricht im übrigen der Herangehensweise beim Quantifizieren von reellwertigen Merkmalen nach dem fuzzy- Verfahren, wo bei Objekt- Konzept- Vergleichen (2.4.1.3.1) ebenfalls eine Ähnlichkeit von 1 bescheinigt wurde, wenn das Objekt im Konzept auftrat.

#### *2.4.4.3 Konzept- Konzept- Vergleich:*

Sind zwei Konzepte mit Symbolen zum Vergleich gegeben, stehen sofort einige Randbedingungen fest. Natürlich muß die Ähnlichkeit bei gleichen Symbolen in beiden Mengen 1 betragen.

Daher kann zur Bewertung das Verhältnis von identischen Symbolen in beiden Mengen zur Anzahl der Symbole in den Mengen genutzt werden. Da die Mengen der Symbole natürlich verschieden groß sein können, ist es sinnvoll, für diese Anzahl der Symbole in den Mengen immer die maximale Anzahl zu nutzen.

Seien nun A und B zwei Konzepte mit Symbolen. Dann wird die Ähnlichkeit der Konzepte nach folgender Formel bestimmt:

$$\gamma = \frac{match}{anz}, \qquad (2\text{-}9.)$$

wobei *match* die Anzahl der übereinstimmenden Symbole und *anz* die maximale Anzahl von Objekten in A und B ist.

### 2.4.5 Merkmal begriff

Zum Abschluß soll nun noch daß Vergleichsverfahren für Begriffe aus einer Hierarchie beschrieben werden.

Da bei Hierarchien Begriffe durch Verallgemeinerung zu Oberbegriffen zusammengefaßt werden, entstehen als Generalisierung immer wieder einzelne Begriffe. Die Konzepte haben somit die gleiche Qualität wie die Objekte und müssen daher beim Vergleich nicht gesondert berücksichtigt werden.

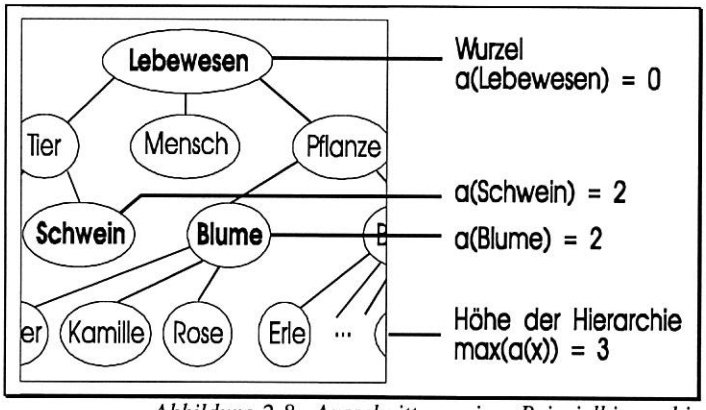



*Abbildung 2-8 - Ausschnitt aus einer Beispielhierarchie*

Für den Vergleich wird jeder Begriff mit seiner Entfernung $a$ (Anzahl der möglichen Oberbegriffe) zur Wurzel der Hierarchie versehen (Abbildung 2-8). Nun sei C der gemeinsame Oberbegriff von A und B. (A, B und C können auch identisch sein). Dann sei der Hierarchieabstand *dist* die minimale Entfernung zwischen einem der beiden Unterbegriffe und dem Oberbegriff.

$$dist = \min(a(A), a(B)) - a(C) \tag{2-10.}$$

Eine Ähnlichkeit läßt sich nun wie folgt quantifizieren:

$$\gamma' = \frac{g - dist}{g}. \tag{2-11.}$$

In der Voreinstellung ist $g$ die maximale Entfernung irgendeines Begriffes der Hierarchie von der Wurzel, also die Höhe der Hierarchie. Dies bedeutet, daß der aktuelle Schritt in der Hierarchie, welcher nötig ist, um zum Oberbegriff zu kommen, ins Verhältnis gesetzt wird zur Größe der Hierarchie.

Als Alternative dazu würde sich ein Verhältnis des aktuellen Schrittes zur Größe der Resthierarchie anbieten, so daß für die Ähnlichkeitsbestimmung nicht mehr die gesamte Hierarchie, sondern nur noch der minimale Teil, welcher zur Beschreibung beider zu vergleichender Begriffe nötig ist, berücksichtigt wird.

Da die Aufgabenstellung an dieser Stelle keine eindeutige Vorgehensweise festlegte und auch in der Literatur keine allgemeingültige Beschreibung gefunden werden konnte, wurden die beiden oben erwähnten Verfahren implementiert und können nun von dem Anwender bzw. der Anwenderin durch Parameter ausgewählt werden (siehe Tabelle 7.3.).

# 3 Abbildung der ermittelten Ähnlichkeiten auf ein WTA- Netz

## *3.1 Das WTA- Netz*

Sollen zwei Graphen miteinander verglichen werden, so kann dies nach [Schädler 1997] durch Abbildung auf ein WTA[3] - Netz erfolgen.

Für die Bildung des WTA- Netzes wird als erstes ein Kompatibilitätsgraph benötigt, welcher die lokalen Ähnlichkeiten zwischen den zu vergleichenden Graphknoten und Graphkanten repräsentiert.

Ein Kompatibilitätsgraph zweier interpretierter gerichteter Graphen (Definition siehe 2.1.) mit gleichen Wertebereichen der Knoten- und Kanteninterpretationen (N, V, f, g, M, W) und (N', V', f', g', M' = M, W' = W) kann also dann erzeugt werden, wenn Vergleichsfunktionen existieren, welche die Markierungen der Knoten bzw. der Kanten beider Graphen vergleichen und das Ergebnis quantifizieren können ( sim(f(n),f'(n')); sim(g(n,m),g'(n',m')) ).

Diese Quantifizierung ist nur im hier betrachteten Fall der Berücksichtigung von lokalen Ähnlichkeiten bedeutsam, bei einer Beschränkung des Vergleichs auf Identitäten kann das Ergebnis der Ähnlichkeitsfunktionen (sim(...,...)) als boolscher Wert angesehen werden.

Der Kompatibilitätsgraph ist dann der folgendermaßen erzeugte Graph (**N**, **V**, **f**, **g**, $\mathbf{M} \subseteq N \times N'$, $\mathbf{W} = \{w, -w_I, 0\}$):

- Knoten des Kompatibilitätsgraphen sind alle Paare von Knoten der beiden Graphen, deren Markierungsvergleich einen Wert über einer Schwelle ergibt: $\mathbf{N} = \{(n,n'): n \in N \wedge n' \in N' \wedge sim(f(n),f'(n')) \geq KnotenSchwelle\}$
- w, $0 < w \leq 1$ markierte Kanten des Kompatibilitätsgraphen sind alle Paare $(\mathbf{n}, \mathbf{m}) = ((n,n'),(m,m'))$ von Knoten $\mathbf{n}, \mathbf{m} \in \mathbf{N}$ des Kompatibilitätsgraphen mit $n \neq m \wedge n' \neq m' \wedge sim(g(n,m),g'(n',m')) \geq KantenSchwelle$
- Mit $-w_I$, $0 < w_I \leq 1$ markierte Kanten des Kompatibilitätsgraphen sind alle Paare $(\mathbf{n}, \mathbf{m}) = ((n,n'),(m,m'))$ von Knoten $\mathbf{n}, \mathbf{m} \in \mathbf{N}$, $\mathbf{n} \neq \mathbf{m}$ des Kompatibilitätsgraphen mit $n = m \vee n' = m'$
- Alle anderen Kanten sind mit 0 markiert

[nach Schädler 1997, S. 3f]

Werden nun die Knoten auf Neuronen, die Kanten entsprechend ihrer Bedeutung auf excitatorische bzw. inhibitorische Kanten des Netzes abgebildet, entsteht ein WTA- Netz.

Betrachten wir an dieser Stelle die in Abbildung 3-1 gezeigten Graphen. Diese stellen die strukturierten Objekte (Originale) in Form markierten Graphen dar. Die Kanten der Graphen sind mit dem Abstand der Originalobjekte markiert (*l*), die Knoten mit dem Typ der Originalobjekte (*{Kreis,Block}*) und - falls dieser im Original angegeben war, also sinnvoll ist - dem Durchmesser (*d*).

[3] Winner takes all

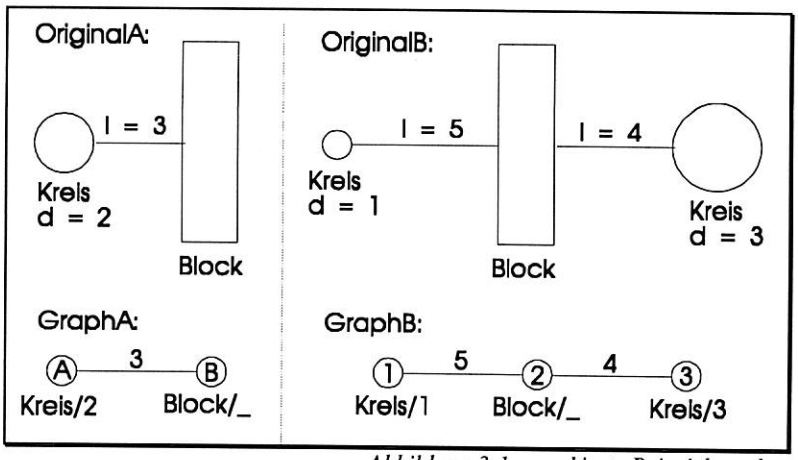



*Abbildung 3-1 - markierte Beispielgraphen*

Soll jetzt ein Kompatibilitätsgraph gebildet werden, sind Funktionen nötig (wie die in Abschnitt 2.4 beschriebenen), welche die Ähnlichkeiten der Knoten- und Kantenmarkierungen quantifizieren können. Im Beispiel sollen Knoten nur unähnlich sein, wenn der Typ der Objekte nicht übereinstimmt - Objekte vom gleichen Typ sind entweder identisch (Ähnlichkeit 1) oder die Ähnlichkeit wird durch das Verhältnis von kleinerem zu größerem Durchmesser bestimmt (auch dies ist eine, wenn auch nicht implementierte Ähnlichkeitsquantifizierung). Für die Ähnlichkeitsbestimmung der Kantenmarkierungen soll hier in gleicher Weise der kleinere Abstand zum Größeren ins Verhältnis gesetzt werden.

Werden nun Knoten- und Kantenschwelle mit 0 festgelegt, so entsteht nach Bildung des Kompatibilitätsgraphen ein WTA- Netz wie in Abbildung 3-2 gezeigt.

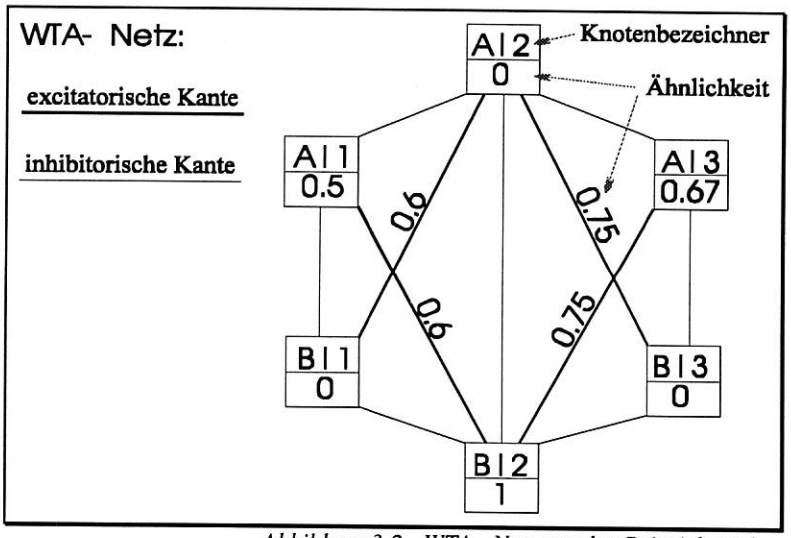



*Abbildung 3-2 - WTA - Netz aus den Beispielgraphen*

Jedem Neuron des WTA- Netzes wird nun zusätzlich zum Bezeichner und der Ähnlichkeit noch eine Aktivität $p$ zugeordnet - diese wird bei der Propagation des Netzes geändert und kennzeichnet bei Stabilität des Netzes die Zugehörigkeit dieses Neurons zur Lösungsmenge. Da die einzelnen Neuronenaktivitäten auf andere Neuronen Einfluß haben, soll durch eine besondere (nichtlineare) Funktion sichergestellt werden, daß sich der auf andere Neuronen wirkende Output $o$ eines Neurons innerhalb eines bestimmten Rahmens bewegt.

$$o = F(p), \quad \text{wobei } F(p) = \begin{cases} 0 & p < 0 \\ 1 & p > 1 \\ p & \text{sonst} \end{cases} \tag{3-1.}$$

Für eine einfachere Beschreibung werden im folgenden die Neuronen des WTA- Netzes mit einer laufenden Nummer versehen. Parameter, welche einem Neuron zugeordnet sind, führen diese Zahl als Index mit, Parameter für Kanten werden durch die Indizes der Neuronen, zwischen welchen diese Kanten verlaufen, gekennzeichnet. So bezeichnet $sim_i$ zum Beispiel die Ähnlichkeit des Neurons i, $Kante_{ij}$ bezeichnet die Kante zwischen den Neuronen i und j (Abbildung 3-3).

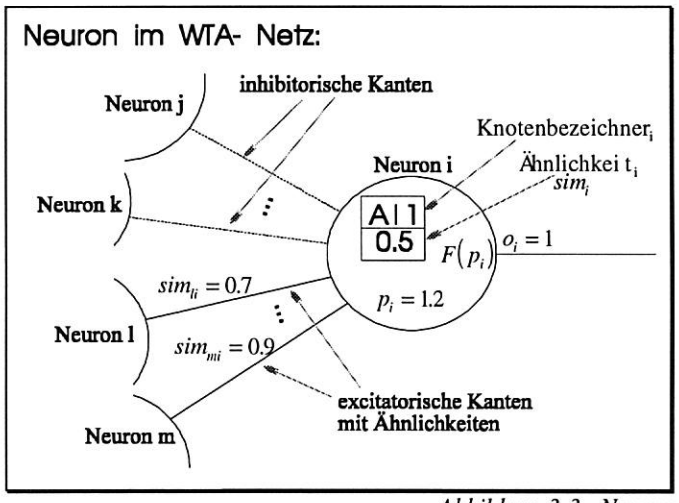



*Abbildung 3-3 - Neuron*

Folgende Propagationsvorschrift ist nun für ein WTA- Netz in Anlehnung an [Wysotzki 1994] gegeben:

$$p_i(t+1) = \sum_j w_{ij} * o_j(t) + (1-d) * p_i(t), \tag{3-2.}$$

$$\text{wobei } w_{ij} = \begin{cases} w & \text{excitatorische Kante}_{ij} \\ -w_I & \text{inhibitorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases}$$

Wird diese Vorschrift bis zum Erlangen eines stabilen Zustandes ausgeführt, ergibt sich bei richtiger Parameterwahl eine Lösung des Constraintproblems als Ergebnis.

Das Problem der richtigen Parameterwahl wurde in [Schädler 1997] ausführlich diskutiert, darum sollen an dieser Stelle nur kurz die zu nutzenden Vorschriften beschrieben werden. Während in [Schädler 1997] jeweils untere oder obere Schranken für die Parameter angegeben werden, werden hier die auch im Programm genutzten direkten Festlegungen angegeben.

$$w_I = r * w = z * w - \frac{s * w}{2}, \qquad z = \max_i \left( \sum_j \begin{cases} o_j(t) & \text{excitatorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases} \right) \tag{3-3.}$$

$$w = \frac{2}{s + \max_i \left( \sum_j \begin{cases} 1 & \text{excitatorische Kante}_{ij} \\ -r & \text{inhibitorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases} \right)} \tag{3-4.}$$

Wie schon aus den Beschreibungen zur Bildung des Kompatibilitätsgraphen hervorging, wurden bisher die Ähnlichkeiten von Knoten- bzw. Kantenpaaren im Kompatibilitätsgraph lediglich gespeichert. Bei der Abbildung des Kompatibilitätsgraphen auf das WTA- Netz sind diese

jedoch bis zu dieser Stelle ignoriert worden. Im folgenden sollen daher verschiedene Verfahren der Ähnlichkeitsabbildung beschrieben werden.

## 3.2 Beeinflussung der Anfangserregung

Alle Neuronen des WTA- Netzes müssen vor der Propagierung des Netzes eine Anfangsaktivierung erhalten. Diese wird üblicherweise anhand der Knotenanzahl im Kompatibilitätsgraphen berechnet. Haben alle Neuronen beim Start der Berechnung die gleiche Aktivierung, so sind auch alle hinsichtlich ihrem Einfluß auf das Ergebnis gleichwertig. Die Herausbildung einer Lösung ist nur abhängig von der Anzahl der Kanten, welche ein Neuron positiv beeinflussen und denen, die es behindern (und natürlich der Aktivität der Neuronen am Ende dieser Kanten).

$$p_i'(0) = p\_init = \frac{\min GraphKnotenAnzahl}{KnotenAnzahl} \qquad (3\text{-}5.)$$

$\min GraphKnotenAnzahl$ Anzahl von Kanten im kleineren von beiden Vergleichsgraphen, entspricht der maximalen Größe des Ergebnisgraphen

$KnotenAnzahl$ Anzahl der Knoten im Kompatibilitätsgraphen

Diese gleichmäßige Verteilung der Anfangsaktivierungen kann durch eine Anfangsaktivierung in Abhängigkeit von der Ähnlichkeit der Knoten, welche dem entsprechenden Neuron zugeordnet sind, ersetzt werden ($sim_i$).

$$p_i(0) = p\_init * sim_i \qquad (3\text{-}6.)$$

Die Skalierung mit *p_init* erfolgt weiterhin, da so vermieden werden kann, daß zu hohe Anfangsaktivierungen zu einfachen, aber unerwünschten Lösungen führen.

Ersichtlich ist ebenfalls, daß mit dieser Vorgehensweise Neuronen, welche eine geringere Ähnlichkeit abbilden, eine geringere Anfangsaktivierung erhalten. Diese Neuronen benötigen zur Beteiligung an einer Lösung entsprechend mehr excitatorische Kanten zu Neuronen mit hoher Aktivität, als dies sonst der Fall wäre. Große Graphen mit geringerer Knotenähnlichkeit werden somit in direkter Konkurrenz zu kleineren Graphen mit höherer Knotenähnlichkeit gestellt. Dabie ist nicht mehr nur die Größe des Ergebnisgraphen relevant für das Ergebnis.

Ein Nachteil dieser Abbildungsmethode ist, daß keine Kantenähnlichkeiten abgebildet werden können - dafür müssen bei Bedarf andere Methoden gewählt werden.

Eine Anpassung der Parameterbestimmung (Formeln 3-3. und 3-4.) für diese Methode ist nicht erforderlich, da keine diesbezüglichen Änderungen unternommen wurden.

## 3.3 Schrittweises Matching

Eine besondere Methode der Abbildung von Ähnlichkeit in das WTA- Netz ist die Methode des schrittweisen Matchings. Die Besonderheit dieser Vorgehensweise besteht darin, daß die Ähnlichkeiten eigentlich gar nicht auf das Netz abgebildet werden, sondern nur die Vergleichsschwellen, bei deren Überschreitung Knoten bzw. Kanten in den Kompatibilitätsgraphen aufgenommen werden, immer weiter schrittweise gesenkt werden.

Im Netz von Abbildung 3-2 würden sich beispielsweise folgende Knotenschwellen anbieten: 1; 0,6; 0,5.

Am Anfang würden also nur Knotenkombinationen in den Kompatibilitätsgraphen aufgenommen, welche mindestens die Ähnlichkeit 1 widerspiegeln. Ist das Netz dann stabil, kann auf

verschiedene Arten mit der Lösung verfahren werden, um danach den kompletten Ablauf für eine geringere Knoten- bzw. Kantenschwelle zu wiederholen.

In den folgenden Abschnitten werden diese verschiedenen Verfahren des schrittweisen Matchings erläutert - dabei wird ein neues Beispielnetz genutzt, da das bisherige zu klein ist, um die schrittweisen Verfahren anschaulich darzustellen. Dieses Beispielnetz ist nur für diese Darstellung konstruiert wurden und repräsentiert keine bestimmte Graphabbildung.

### 3.3.1 Schrittweises loses Matching

Das schrittweise lose Matching erscheint sofort als einfache Möglichkeit, die eben beschriebene Idee fortzusetzen. Dabei wird die alte Lösung als Anfangsaktivierung für das neue, um Neuronen bzw. Kanten, welche eine geringere Ähnlichkeit repräsentieren, erweiterte Netz benutzt.

Neuronen oder Kanten 'geringerer Ähnlichkeit' werden zunächst nicht berücksichtigt und erst nach Bildung einer suboptimalen Lösung hinzugenommen und können diese ergänzen oder Knoten der vorherigen Lösung ersetzen.

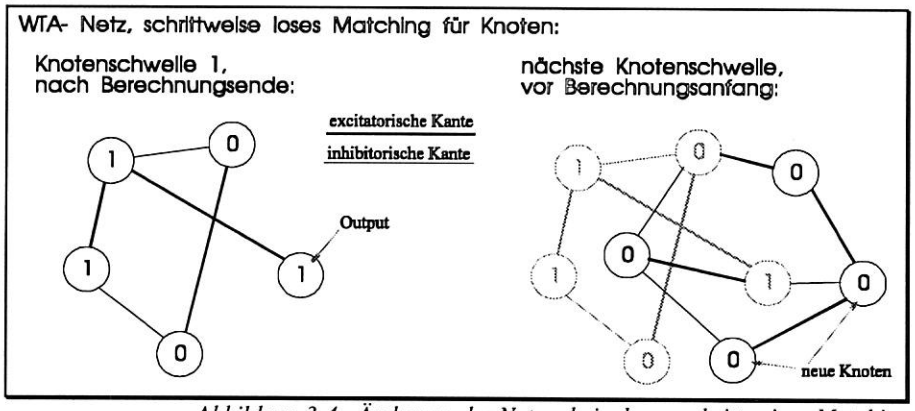



*Abbildung 3-4 - Änderung des Netzes beim losen schrittweisen Matching*

Abbildung 3-4 zeigt links ein beliebiges WTA- Netz, welches mit einer Knotenschwelle von 1 erzeugt wurde und einen stabilen Zustand erreicht hat. Werden nun nach dem Verfahren des losen Matchings neue Neuronen hinzugefügt, entsteht zum Beispiel das auf der rechten Seite gezeigt Netz. Alle Neuronen des alten Netzes bleiben erhalten und die neuen Neuronen erhalten keine Anfangsaktivierung.

### 3.3.2 Schrittweises reduziertes Matching

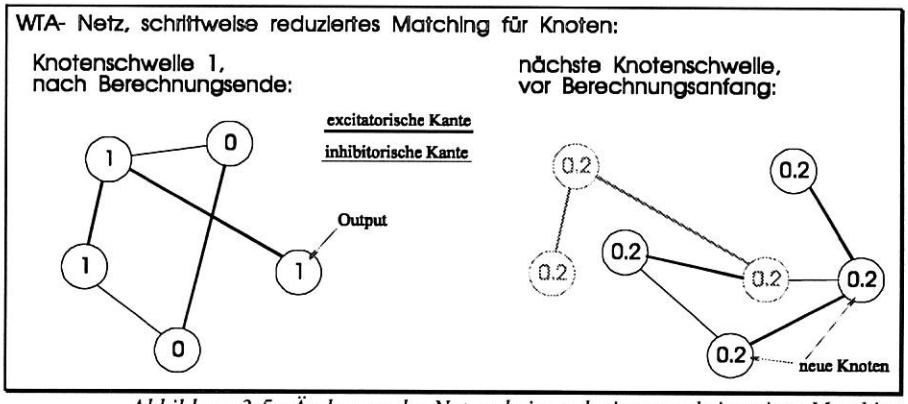



*Abbildung 3-5 - Änderung des Netzes beim reduzierten schrittweisen Matching*

Eine weitere Möglichkeit des schrittweisen Matchings besteht darin, nach dem Erhalten einer Lösung alle Neuronen, die nicht zur Lösung gehören, aus dem Netz zu entfernen. Danach werden alle neuen Neuronen bzw. Kanten hinzugefügt und allen noch vorhandenen Neuronen

(auch denen aus der vorherigen Lösung) wird eine neue Anfangsaktivierung gegeben. Somit werden Knotenkombinationen, welche nicht zur Lösung gehören, aus dem Netz entfernt, bevor dieses zur nächsten Berechnungsstufe gegeben wird.

Die Anzahl der inhibitorischen Kanten, die auf ein Neuron einer Lösung mit großer Ähnlichkeit zeigen, wird somit verringert. Der Lösungsraum wird jedoch dadurch erweitert, da sich auch die Neuronen der vorherigen Lösung in diesem Stadium durch nichts von den Neuronen, welche geringere Ähnlichkeiten repräsentieren, unterscheiden (Abbildung 3-5).

### 3.3.3 Schrittweises fixiertes Matching

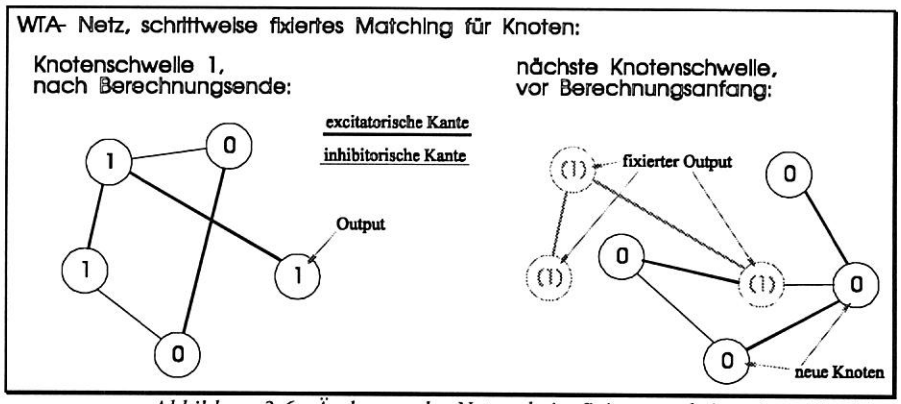



*Abbildung 3-6 - Änderung des Netzes beim fixierten schrittweisen Matching*

Natürlich fehlt nun noch die Erwähnung einer Möglichkeit, bei welcher die alte Lösung als bereits den Ansprüchen genügende Lösung fixiert wird.

Die so festgelegten Neuronenpotentiale bilden einen externen Input für neue Neuronen, welche keine Anfangserregung besitzen. Auf diese Weise wird die bestehende Lösung vorrangig um Neuronen bzw. Kanten geringerer Ähnlichkeit ergänzt, welche zur bestehenden Lösung passen (d.h. keine inhibitorischen Kanten mit Neuronen der Lösung haben).

Alle schrittweisen Matchingalgorithmen lassen sich nicht nur für Knotenähnlichkeiten verwenden, auch für Kantenähnlichkeiten stehen sie in gleicher Weise zur Verfügung.

Obwohl die Verfahren sehr einfach wirken, ist auffallend, daß durch sie die Anzahl der Neuronen im Netz relativ gering gehalten werden kann. Dies ist besonders bei zeitkritischen Vergleichen ein recht wichtiges Kriterium (siehe Ergebnisse in Abschnitt 10.).

Wie auch in der im vorherigen Abschnitt beschriebenen Methode, wird an dieser Stelle die Parameterberechnung für das WTA- Netz (Formeln 3-3. und 3-4.) nicht beeinflußt, die Propagierung der einzelnen Teilnetze erfolgt wie in Formel 3-2. beschrieben.

## 3.4 Beeinflussung eines externen Inputs

Während die eben beschriebenen Abbildungsmethoden die Ähnlichkeit lediglich zur Erstellung des WTA- Netzes nutzen, kann natürlich auch der Ablauf der Propagation selbst geändert werden. So kann die Update- Regel (Formel 3-2.) um einen externen Input erweitert werden, welcher durch die Ähnlichkeit derjenigen Knoten, die in einem Neuron kombiniert wurden, bestimmt wird.

$$p_i(t+1) = \sum_j w_{ij} * o_j(t) + (1-d) * p_i(t) + I_i \qquad (3\text{-}7.)$$

Dieser externe Input ist eigentlich nach der Initialisierung für die gesamte Iterationsdauer konstant. Dieses Verfahren hat somit den Nachteil, daß der externe Input völlig unabhängig von der aktuellen Schrittweite *w* ist. Dies kann bei einem ungünstigen Verhältnis der Schrittweite zum externen Input dazu führen, daß bei der Propagation der Updatevorschrift lediglich die Auswirkungen des externen Inputs, jedoch nicht die der excitatorischen bzw. inhibitorischen Kanten zu bemerken sind. Auf diese Weise kann also keine sinnvolle Lösung des Matchingproblems erzielt werden.

Darum wurde für eine sinnvolle Anwendung dieser Methode der externe Input zusätzlich mit die Schrittweite *w* skaliert:

$$p_i(t+1) = \sum_j w_{ij} * o_j(t) + (1-d) * p_i(t) + \tilde{I}_i * w \qquad (3\text{-}8.)$$

So bleibt der Einfluß des externen Inputs immer im Rahmen der berechneten Schrittweite.

Durch das Skalieren des externen Inputs mit *w* ergibt sich zusätzlich ein Vorteil beim Verständnis seiner Wirkung.

Wird in Formel 3-8. $\tilde{I}_i = 1$ gesetzt, so hat dieses Neuron bildlich gesprochen eine zusätzliche excitatorische Kante zu einem Neuron der Lösungsmenge, also zu einem bias- Neuron (siehe auch Abbildung 3-7).

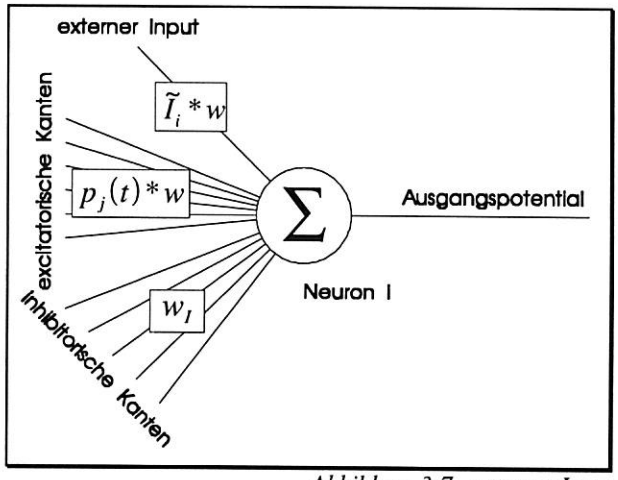



*Abbildung 3-7 - externer Input*

Zur praktischen Realisierung empfiehlt sich daher folgende Abbildung der Ähnlichkeit auf den externen Input:

$$\tilde{I}_i = skal * sim_i \qquad (3\text{-}9.)$$

Die Ähnlichkeit wird mit einem zusätzlichen Faktor skaliert, welcher die Anzahl der zusätzlichen excitatorischen Kanten angibt, die ein Neuron, welches eine Ähnlichkeit von 1 repräsentiert, durch den externen Input erhält. Ist die Ähnlichkeit, die dieses Neuron widerspiegelt, geringer, werden natürlich auch entsprechend weniger zusätzliche Kanten durch den externen Input 'simuliert'.

Bei dieser Methode ist zu beachten, daß durch den externen Input natürlich die maximal ein Neuron unterstützenden Einflüsse erhöht werden. Aus diesem Grund muß bei der Berechnung der Parameter die Formel 3-3. so angepaßt werden, daß dies bei der Größenermittlung von r berücksichtigt wird.

$$w_I = r * w = z * w - \frac{s * w}{2}, \qquad z = \max_i \left( \sum_j \begin{cases} o_j(t) & \text{excitatorische Kante}_{ij} \\ \tilde{I}_i & i = j \\ 0 & \text{sonst} \end{cases} \right) \tag{3-10.}$$

Zu erwähnen ist an dieser Stelle der Vollständigkeit wegen noch, daß auf diese Weise natürlich nur Knotenähnlichkeiten abgebildet werden können.

## 3.5 Skalierung der Kanten des Netzes

Als letzte zu betrachtende Abbildungsmöglichkeit der Ähnlichkeit auf das Netz soll hier die Skalierung der excitatorischen Kanten des WTA- Netzes betrachtet werden.

Diese Methode bietet sich besonders zur Abbildung von Kantenähnlichkeiten an, falls diese nicht durch schrittweises Matching abgebildet werden sollen.

Zwei verschiedene Konzepte sind dabei denkbar. Zum einen kann, wie eben erwähnt, die Kantenähnlichkeit zur Skalierung genutzt werden, zum anderen können noch zusätzlich die Ähnlichkeiten der angrenzenden Knoten berücksichtigt werden.

### 3.5.1 Skalierung anhand der Kantenähnlichkeiten

Die Kantenähnlichkeit kann wie folgt zur Skalierung genutzt werden:

$$p_i(t+1) = \sum_j w_{ij} * o_j(t) + (1-d) * p_i(t), \tag{3-11.}$$

$$\text{wobei } w_{ij} = \begin{cases} w * sim_{ij} & \text{excitatorische Kante}_{ij} \\ -w_I & \text{inhibitorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases}$$

Auch hier muß beachtet werden, daß sich natürlich die Netzparameter ändern, wenn die excitatorischen Kanten neu gewichtet werden.

Darum müssen folgende Berechnungsvorschriften genutzt werden:

$$w_I = r * w = z * w - \frac{s * w}{2}, \qquad z = \max_i \left( \sum_j \begin{cases} sim_{ij} * o_j(t) & \text{excitatorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases} \right) \tag{3-12.}$$

$$w = \frac{2}{s + \max_i \left( \sum_j \begin{cases} sim_{ij} & \text{excitatorische Kante}_{ij} \\ -r & \text{inhibitorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases} \right)} \tag{3-13.}$$

### 3.5.2 Skalierung anhand der Kanten- und Knotenähnlichkeiten

Da eine Kante immer zwischen zwei Knoten verläuft, wird als Skalierungsfaktor bei dieser Methode das Produkt der Ähnlichkeiten, welche durch die beiden verbundenen Neuronen widergespiegelt werden, mit der Ähnlichkeit, welche durch die excitatorische Kante repräsentiert wird, multipliziert.

$$p_i(t+1) = \sum_j w_{ij} * o_j(t) + (1-d) * p_i(t), \tag{3-14.}$$

$$\text{wobei } w_{ij} = \begin{cases} w * sim_{ij} * sim_i * sim_j & \text{excitatorische Kante}_{ij} \\ -w_I & \text{inhibitorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases}$$

Wie im vorigen Abschnitt ist auch hierbei eine Anpassung der Netzparameterberechnung nötig:

$$w_I = r * w = z * w - \frac{s * w}{2},$$

$$z = \max_i \left( \sum_j \begin{cases} sim_{ij} * sim_i * sim_j * o_j(t) & \text{excitatorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases} \right) \tag{3-15.}$$

$$w = \frac{2}{s + \max_i \left( \sum_j \begin{cases} sim_{ij} * sim_i * sim_j & \text{excitatorische Kante}_{ij} \\ -r & \text{inhibitorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases} \right)} \tag{3-16.}$$

## 3.6 Gesamtstruktur und Berechnungsalgorithmen des WTA- Netzes

Da für alle verschiedenen Versionen der Ähnlichkeitsabbildung nur *ein und dasselbe* Netz genutzt werden sollte, mußten alle Berechnungsvorschriften zu einer Gesamtvorschrift verbunden werden.

Update- Regel:

$$p_i(t+1) = \begin{cases} \sum_j w_{ij} * o_j(t) + (1-d) * p_i(t) + \tilde{I}_i * w & \text{wenn} \quad \ddot{a}nderbar_i = 1 \\ p_i(t) & \text{wenn} \quad \ddot{a}nderbar_i = 0 \end{cases}, \tag{3-17.}$$

$$\text{wobei } w_{ij} = \begin{cases} w * \tilde{w}_{ij} & \text{excitatorische Kante}_{ij} \\ -w_I & \text{inhibitorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases}$$

Parameterberechnung:

$$w_I = r * w = z * w - \frac{s * w}{2}, \qquad z = \max_i \left( \sum_j \begin{cases} \tilde{w}_{ij} * o_j(t) & \text{excitatorische Kante}_{ij} \\ \tilde{I}_i & i = j \\ 0 & \text{sonst} \end{cases} \right) \tag{3-18.}$$

$$w = \frac{2}{s + \max_i \left( \sum_j \begin{cases} \tilde{w}_{ij} & \text{excitatorische Kante}_{ij} \\ -r & \text{inhibitorische Kante}_{ij} \\ 0 & \text{sonst} \end{cases} \right)} \tag{3-19.}$$

Mit den nun eingeführten, teilweise neuen Parametern können alle eben beschriebenen Abbildungsverfahren genutzt werden.

Am auffälligsten dürfte an dieser Stelle der (bool´sche) Parameter $änderbar_i$ in Formel 3-17. sein, welcher speziell für die diversen Methoden des schrittweisen Matchings eingeführt wurde. So kann damit das schrittweise fixierte Matching realisiert werden, indem bei den Neuronen der Lösung $änderbar_i = false$ (0) gesetzt wird. Auch das 'Entfernen' von Neuronen aus dem Netz kann mit diesem Parameter bewerkstelligt werden. Dafür wird die Aktivität der Neuronen zuerst auf Null gesetzt und danach werden diese auf $änderbar_i = false$ (0) gesetzt.

Ein weiterer Parameter ist $\tilde{w}_{ij}$, welcher für die Kantenskalierung genutzt wird. Sollen die Kanten nicht skaliert werden, muß $\tilde{w}_{ij}$ auf 1 festgelegt werden. Ist jedoch eine Skalierung der Kanten anhand bestimmter Werte geplant, so müssen diese Werte in $\tilde{w}_{ij}$ eingetragen werden (siehe Formeln 3-11. und 3-14.).

Zur Nutzung eines externen Inputs wurde der Parameter $\tilde{I}_i$ vorgesehen, welcher bei Berechnungen ohne externen Input auf 0 gesetzt werden muß, sonst jedoch den Skalierungsfaktor (siehe Formel 3-9.) widerspiegeln sollte.

# 4 Gütebewertung des Matches / Energieberechnung

Um die Qualität des Matches zu bewerten und gleichzeitig stabile Endzustände erkennen zu können, wird bei jedem Berechnungsschritt die Energie des Netzes bestimmt. Da die Propagation des Netzes einem Gradientenabstieg auf einer Energiefunktion entspricht, kann die Suche nach einer Lösung des Graphmatchingproblems also auch einer Suche nach einem (lokalen) Minimum der Energiefunktion gleichgesetzt werden.

Die Berechnung der Energie erfolgt in Anlehnung an [Schädler 1997] nach folgender Formel:

$$E(t) = -\frac{1}{2}\sum_{i,j} o_i(t) * o_j(t) * \begin{Bmatrix} \tilde{w}_{ij} & \text{excitatorische Kante}_{ij} \\ r & \text{inhibitorische Kante}_{ij} \\ 0 & \text{sonst} \end{Bmatrix} - \sum_i \tilde{I}_i * o_i(t) \qquad (4\text{-}1.)$$

Eine so berechnete Energie gibt jedoch nur begrenzte Informationen über die Ähnlichkeit der beiden zu vergleichenden Ausgangsgraphen. Um ein anschauliches Maß für die Quantifizierung der Graphähnlichkeiten zu erhalten, bietet sich zusätzlich folgende Gütebestimmung an:

$$Güte1 = \frac{-E}{\frac{1}{2}n(n-1) + \max_i(\tilde{I}_i) * n} \quad ,n = \text{maximale Knotenzahl im zu erwartenden Graph} \qquad (4\text{-}2.)$$

Auffallend in Formel 4-2. ist die Verwendung von $n$ - der maximalen Knotenanzahl im zu erwartenden Ergebnisgraphen. Da die tatsächliche Knotenzahl natürlich vor Ermittlung des Ergebnisgraphen nicht bekannt ist, wird hier immer die maximale Knotenanzahl in den zu vergleichenden Graphen genutzt. Nur in dem Fall, daß einer der beiden verglichenen Graphen in den Markierungen Konzepte enthält (also selbst das Ergebnis eines Vergleichs ist), der andere jedoch nicht, wird davon ausgegangen, daß Graphen generalisiert werden sollen. Der Graph mit den Konzepten in den Merkmalen wird dann als Generalisierungsziel betrachtet und dessen Knotenzahl wird als erwartete Knotenzahl für $n$ verwendet.

Bei bestimmten Ähnlichkeitsabbildungen, zum Beispiel beim schrittweisen Matching, kann auch diese Formel nur unzureichende Aussagen über die Ähnlichkeit der Knoten- und Kantenabbildungen im Ergebnisgraphen treffen, da deren Ähnlichkeit nicht direkt in die Energieberechnung eingeht. (Die Ähnlichkeit wird bei einigen Verfahren nicht direkt auf die Netzparameter abgebildet).

Aus diesem Grund wurden zwei weitere Gütebewertungsfunktionen vorgesehen, welche die Ähnlichkeit explizit mit in die Berechnung einfließen lassen (mit $n$ aus Formel 4-2.):

$$Güte2 = \frac{\sum_i sim_i * o_i(t)}{n} \qquad (4\text{-}3.)$$

$$Güte3 = \frac{1}{2}\frac{\sum_{i,j} sim_{ij} * o_i(t) * o_j(t)}{\frac{1}{2}n(n-1)} \qquad (4\text{-}4.)$$

Während die Energieberechnung entscheidend für die Feststellung des Berechnungsendes ist, sind die oben beschriebenen Güteberechnungen ausschließlich für die Matchbewertung durch den Nutzer bzw. die Nutzerin gedacht. Andere Gütemaße sind durchaus vorstellbar und können natürlich unter Umständen für bestimmte Aufgaben sinnvoller sein, dennoch spiegeln die obigen Maße die Qualität des Matches gut wieder.

# 5 Grundkonzepte und Datentypen

Da für das Programm keine Einschränkungen bezüglich der Ein- und Ausgabe oder genauere Informationen über die Graphstrukturen gegeben waren, sollte zuallererst ein flexibel nutzbares Grundkonzept für das Programm entwickelt werden.

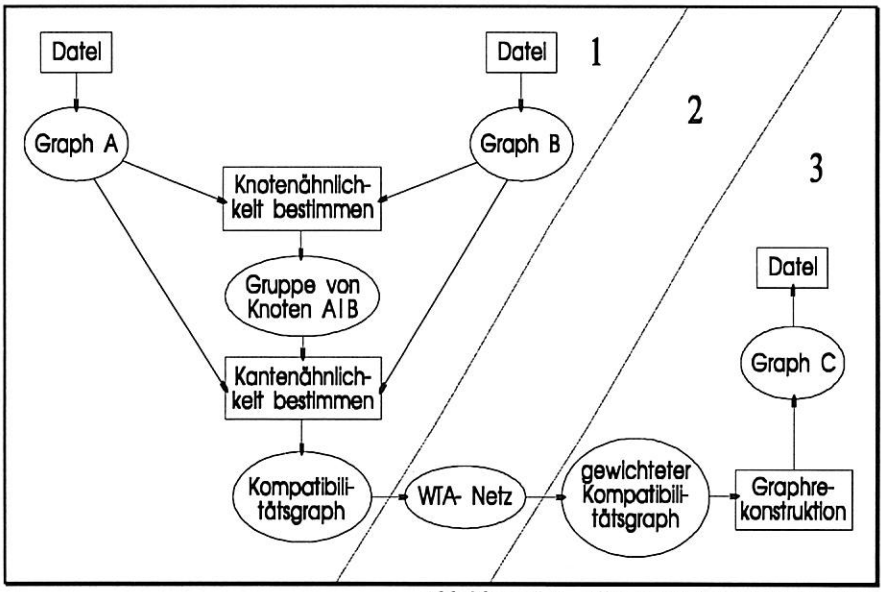



*Abbildung 5-1 - allgemeine Programmstruktur*

Zur Eingabe der Graphen sollten diesem Konzept zufolge Dateien genutzt werden, die mit üblichen Texteditoren bearbeitet werden können und sämtliche Graphinformationen in verständlicher Form enthalten. Diese Vorgehensweise ermöglichte zum einen die Einsparung eines eigenen Grapheditier- Programmbausteins, zum anderen ist eine lesbare Graphbeschreibung leicht zu konvertieren und anderen Tools zur Verfügung zu stellen.

Nachteilig dabei war, daß sich momentan keine übliche Graphbeschreibungssprache etabliert hat, und daß somit ein Neuentwurf eine speziell auf dieses Programm zugeschnittene Version enthalten mußte.

Als nächstes war klar, daß der Ergebnisgraph natürlich in genau der selben Qualität wie die Eingabegraphen vorliegen sollte, also ebenfalls in einer Datei gespeichert werden sollte und wiederum als Vergleichsgraph zur Verfügung stehen sollte.

Es konnten nun drei wesentliche Programmteile unterschieden werden. Als erstes mußten die zu vergleichenden Graphen auf den Kompatibilitätsgraphen abgebildet werden, um danach das WTA- Netz bis zu einer stabilen Lösung zu propagieren. Anschließend sollte ein Ergebnisgraph aus dem Kompatibilitätsgraphen rekonstruiert werden (Abbildung 5-1).

Zusätzlich war durch die recht offene Problembeschreibung festgelegt, daß sämtliche Strukturen nicht vorher in ihrer Größe festgelegt werden konnten. Es mußte daher weitgehend das Prinzip einer dynamischen Speicherverwaltung genutzt werden, was bedeutet, daß fast alle Klassen erst während der Programmausführung den erforderlichen Speicher vom System anfordern.

Die nachfolgenden Kapitel beschreiben Programmstrukturen, welche nötig sind, um Graphen abzubilden und zu vergleichen. Die Reihenfolge der Beschreibung entspricht dabei der Reihenfolge des Entwurfs. Dabei wurde das klassische Verfahren des bottom- up Entwurfs genutzt, daß heißt es wurde prinzipiell stets von unter- zu übergeordneten Strukturen hin programmiert.

So war es zum Beispiel nötig, zuerst die Merkmalsabbildungen zu definieren, bevor diese den Knoten- bzw. Kantenmarkierungen untergeordnet werden konnten, da für mich ganz wesentlich das Problem der Merkmalsvergleiche und dessen Umsetzung in C++ war. Erst durch die (fast) komplette Implementierung der Merkmalsabbildung und einiger Vergleiche konnte der gewählte objektorientierte Ansatz bestätigt werden, so daß danach 'einhüllende' Objekte darauf aufbauen konnten.

## 5.1 Merkmalsabbildung

### 5.1.1 Markierungsstrukturbeschreibung

Um die Vergleiche von Knoten- und Kantenmarkierungen effektiv durchführen zu können, mußten die Merkmalsverktoren in bestimmte Einzelmerkmale zerteilt werden, für die ein jeweils eigener Vergleich implementiert werden konnte. Die Beschreibung, aus welchen Basismerkmalen sich die Knoten- und Kantenmarkierungen zusammensetzen, wird für alle zu vergleichenden Graphen aus einer Datei eingelesen - so können beliebige Merkmalsvektoren zur Programmlaufzeit vor dem Graphvergleich festgelegt werden.

Diese Struktur kann für verschiedene Graphen und verschiedene Vergleiche gewechselt werden, es können jedoch nur Graphen mit gleicher Markierungsstrukturbeschreibung verglichen werden.

### 5.1.2 Knotenmarkierungen

Jeder Knoten kann natürlich nicht nur durch ein Merkmal, sondern durch beliebig viele verschiedene Einzelmerkmale markiert werden. Diese werden zu einem Merkmalsvektor zusammengefaßt. Ein Vergleich zweier Knoten besteht darum in dem Vergleich jeweils aller Einzelmerkmalsausprägungen beider Knoten.

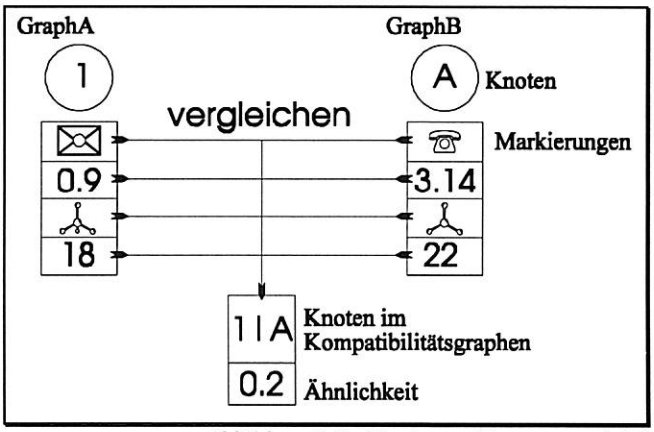



*Abbildung 5-2 - Knotenmarkierungsvergleich*

In Abbildung 5-2 ist an einem Beispiel der Vergleichsablauf von Knotenmerkmalen dargestellt, zwei Knoten von zwei verschiedenen Graphen sollen für die Aufnahme in den Kompatibilitätsgraphen verglichen werden.

Dabei fällt sofort auf, daß natürlich beide Knoten die gleiche Anordnung von Einzelmerkmalen in den Merkmalsvektoren (Markierungen) enthalten müssen. Nur so können diese anhand der Informationen in der Merkmalstrukturbeschreibung verglichen werden.

Zuerst werden alle Einzelmarkierungen miteinander verglichen, danach werden alle Einzelvergleichsergebnisse zusammengefaßt und dann wird (falls das Ergebnis über einer Schwelle liegt) einem Kompatibilitätsgraphknoten das Vergleichsergebnis zugeordnet.

Eine für dieses Beispiel gültige Strukturbeschreibung wäre zum Beispiel Symbol/Reell/Symbol/Ganz, was bedeuten soll, daß alle Knoten aller im Moment betrachteten Graphen (also auch des Ergebnisgraphes) als erstes durch ein Symbol, gefolgt von einer reellen Zahl, einem weiteren Symbol und einer ganzen Zahl markiert sind.

### 5.1.3 Kantenmarkierungen

Im Gegensatz zu Knoten ergibt sich bei Kanten das Problem, daß diese unter bestimmten Umständen gerichtet sein können. Um dies in allgemeiner Form im Graph abzubilden, bieten sich diverse Lösungen an.

So könnten zum Beispiel verschiedene Arten von Kanten vorgesehen werden, welche für besondere Richtungen oder Ungerichtetheit stehen. Der Nachteil einer solchen Vorgehensweise besteht in den zusätzlichen Unterschieden, die bei der Bearbeitung der Kanten gemacht werden müssen. Ein Graph besteht dann nicht nur aus Knoten und Kanten, sondern aus Knoten, ungerichteten Kanten und gerichteten Kanten.

Daher war es angebracht, diese Einteilungen nicht auf Graphebene, sondern auf Kantenebene vorzunehmen. Jede Kante verkörpert im genutzten Modell alle drei möglichen Kantenarten, also ungerichtete, ankommende und abgehende Kanten. All diesen integrierten Kantenarten ist jeweils eine Markierung zugeordnet, so daß durch bestimmte Merkmale innerhalb der Merkmalsvektoren gerichtete und ungerichtete Kanten unterschieden werden können.

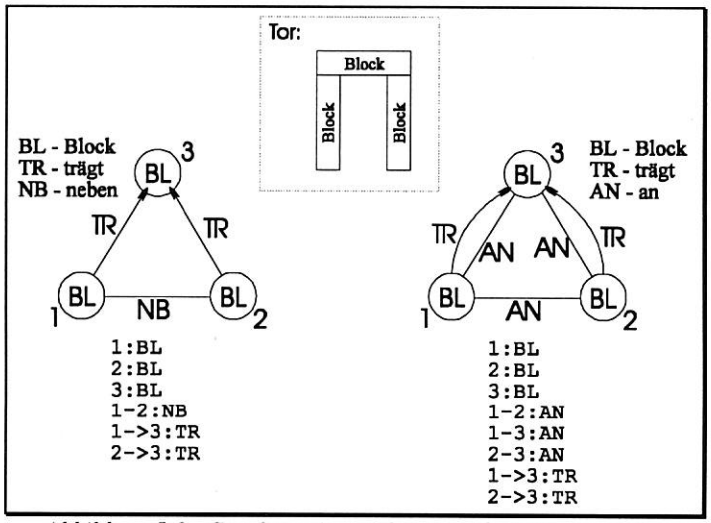



*Abbildung 5-3 - Graphen mit gerichteten und ungerichteten Kanten*

In Abbildung 5-3 ist eine Situationsbeschreibung mit gerichteten Graphen am klassischen Torbeispiel verdeutlicht. [nach Wysotzki 1994, S. 5-11]

Sofern beim Vergleichen die verschiedenen Kantenarten berücksichtigt werden können, ist offensichtlich, daß bei Vergleich des linken Graphen mit sich selbst niemals die Kante 1 -2 auf die Kante 2 -3 abgebildet werden könnte, da diese von verschiedener Art sind.

Schwieriger wird der Vergleich jedoch schon dann, wenn der Bezeichner 'NB' (neben) durch 'AN' (an) ersetzt wird (rechter Graph in Abbildung 5-3). Der nun folgende Vergleich könnte natürlich die ungerichteten Kanten zwischen 1 -2 und 2 -3 aufeinander abbilden, und nur die Tatsache, daß die gerichtete Kante zwischen 2- 3 (TR) nicht abgebildet werden kann, schließt diese Kantenzuordnung aus.

Dies bedeutet, daß für einen solchen Vergleich von Kanten immer die Gesamtheit aller Kanten berücksichtigt werden müßte, was nicht sehr effektiv ist.

Der in Abbildung 5-4 gezeigte Graph beschreibt immer noch das Tor aus dem vorigen Beispiel (Abbildung 5-3). Jedoch wurden nun *alle* Kanten per Definition als dreigeteilt angenommen - alle Kanten bestehen aus einem ungerichteten und zwei entgegengesetzt gerichteten Teilen.

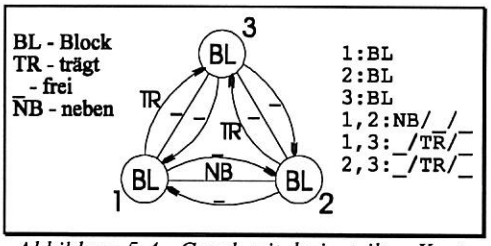



*Abbildung 5-4 - Graph mit dreigeteilten Kanten*

Um die gerichteten Teile voneinander zu unterscheiden, wird eine Aufteilung in abgehende und ankommende Markierungen unternommen, welche aufgrund der Symmetrie der Kantenbeschreibung lediglich vom *betrachteten* Ausgangspunkt der Kante abhängig sind.

Die Symmetrie der gerichteten Kantenmarkierungen, welche durch die Merkmalsstruktur sichergestellt worden ist, hat nun zur Folge, daß ein und dieselbe Kante natürlich auch in der entgegengesetzten Richtung zum Vergleich genutzt werden kann. Sollen also Kanten verglichen werden, so ist gleichzeitig wichtig, von welchem Knoten diese (im Moment genau dieses Vergleiches) ausgehen, weshalb bei allen Zugriffen auf gerichtete Merkmale der Kanten gleichzeitig die Ausgangsknotenbezeichner dieser Kanten übergeben werden müssen (siehe Programm 6.4.).

Der Vergleich von Kanten kann jetzt auf fast genau die gleiche, einfache Art vorgenommen werden wie der von Knoten.

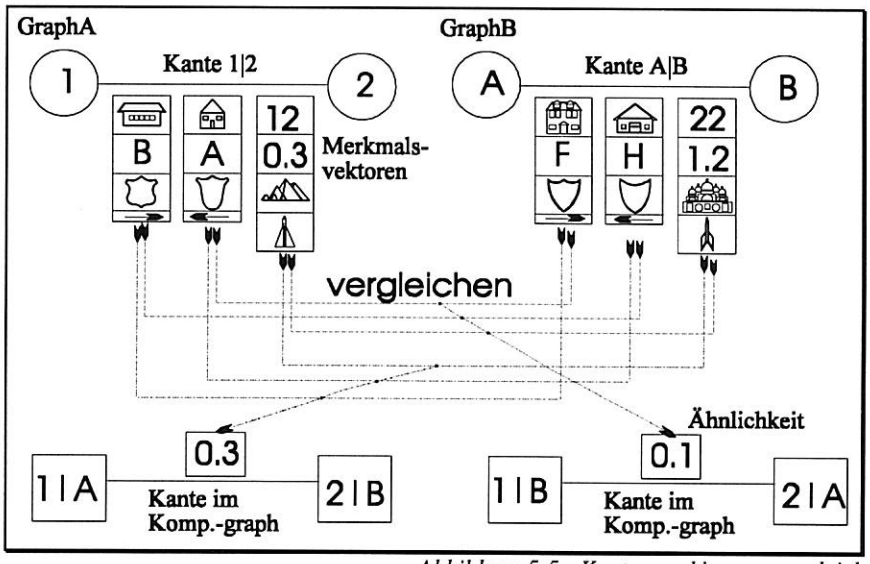



*Abbildung 5-5 - Kantenmarkierungsvergleich*

In Abbildung 5-5 werden zwei Kanten miteinander verglichen. Jeder Kante sind dabei 3 Merkmalsvektoren zugeordnet, 2 entgegengesetzt gerichtete und ein ungerichteter. Für einen Vergleich dieser Kanten sind zwei verschiedene Anordnungen möglich (Kante 1-2 auf A-B oder Kante 1-2 auf B-A). Da bei diesen beiden verschiedenen Anordnungen also auch verschiedene gerichtete Kantenmarkierungen miteinander verglichen werden, folgen auch unterschiedliche Kantenähnlichkeiten (hier pauschal angegeben) aus diesen Zuordnungen.

Die Grafik verdeutlicht, daß aufgrund verschiedener Knotenzuordnungen die Struktur der gerichteten Markierungen unabhängig von der Richtung immer gleich sein muß, sich jedoch von der Struktur der ungerichteten Markierung unterscheiden kann.

## 5.2 Graphabbildung

Es sollen Graphen genutzt werden, welche der in Abschnitt 2.1. getroffenen Definition genügen müssen, G = (N, V, f, g, M, W).

Genau diese allgemeine Graphbeschreibung umzusetzen, war also das Ziel des Entwurfs der Graphimplementierung.

Ein Graph besteht demnach im Programm aus (Zeigern auf) Knoten und Kanten. Wichtig dabei ist, daß die Registrierung der Kanten am Graphen lediglich für die Graphausgabe und das Freigeben des genutzten Speichers sinnvoll ist, für Vergleiche ist ein Zugriff auf die Kanten besser über die Knoten zu realisieren (siehe auch nächsten Abschnitt). Die einzelnen Markierungen sind den Knoten bzw. Kanten entsprechend den im vorigen Abschnitt erläuterten Ansätzen zugeordnet.

### 5.2.1 Graphknoten

Da die Knoten eines Graphen die Objekte abbilden, Kanten aber Relationen zwischen diesen Objekten, ist sofort ersichtlich, daß die Bearbeitung von Graphen immer auch anhand der Knoten erfolgen muß. Eine Relation ohne Objekte existiert nicht, also sind auch alle Kanten lediglich als Beziehungen zwischen Knoten interessant.

Alle Knoten enthalten außer ihrem Bezeichner und der dem Knoten zugeordneten Markierung Zeiger auf alle Kanten, welche diesen Knoten berühren. So ist ein Zugriff auf eine bestimmte Kante direkt über den meist vorher bestimmten Ausgangsknoten möglich, indem die Ziele aller Kanten von diesem Knoten aus nach dem gewünschten Kantenziel durchsucht werden.

Durch die gewählte Abbildungsweise werden beide Möglichkeiten der Kantensuche, die in einer Menge von Kanten und die von einem Knoten ausgehende, unterstützt und genutzt.

### 5.2.2 Graphkanten

Wie bereits in Abschnitt 5.1.3 erläutert, besteht im genutzten Modell jede Graphkante gleichzeitig aus einem ungerichteten Merkmalsvektor und zwei entgegengesetzt gerichteten Markierungen.

An jeder Kante muß vermerkt sein, zwischen welchen Knoten sie verläuft. Dafür müssen die Knotenbezeichner der Knoten in der Kante registriert werden.

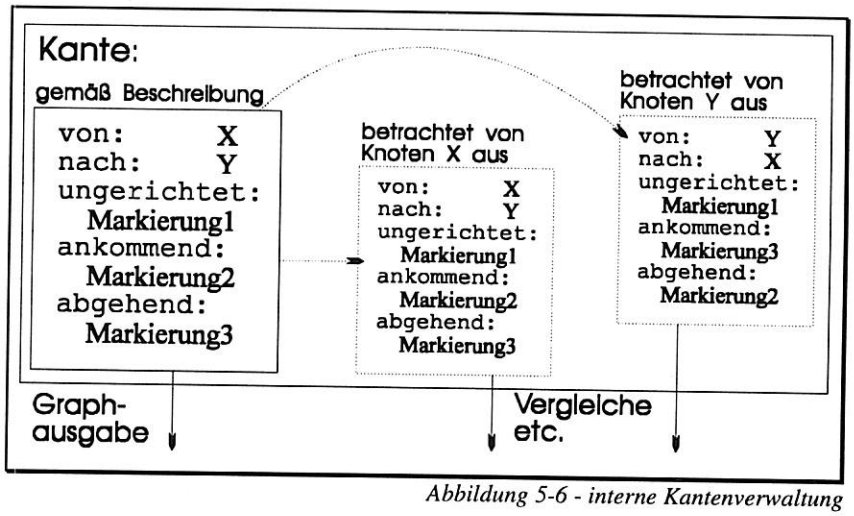



*Abbildung 5-6 - interne Kantenverwaltung*

Jede Kante wird gemäß ihrer Initialisierung als gerichtete Kante von X nach Y betrachtet, und enthält einen ungerichteten Merkmalsvektor, einen abgehenden (von X nach Y) und einen ankommenden (von Y nach X) Merkmalsvektor. Soll der Graph in eine Datei ausgegeben wer-

den, werden alle im Graph registrierte Kanten gemäß ihrer Initialisierung- das heißt so, wie diese in der Graphbeschreibungsdatei (Abschnitt 7.2.) beschrieben wurden- ausgegeben.

Sollen Kanten jedoch verglichen werden, so wird die Symmetrie der Beschreibung genutzt und entsprechend dem Ausgangsknoten werden ankommende bzw. abgehende Kantenmarkierungen für diesen Vergleich neu zugeordnet (Abbildung 5-6).

## 5.3 Kompatibilitätsgraph / WTA- Netz

Aus Sicht der Implementierung gibt es keine Unterscheidung zwischen Kompatibilitätsgraph und WTA- Netz. Dennoch wird an dieser Stelle von einem Kompatibilitätsgraph gesprochen, da dieser die Ähnlichkeiten der beiden Graphen widerspiegeln soll.

Üblicherweise wird der Kompatibilitätsgraph als Matrix zwischen allen Knotenzuordnungen abgebildet. Diese Implementierungsart hat aber verschiedene Nachteile.

Ein Knoten im Kompatibilitätsgraph hat einige excitatorische und einige inhibitorische Verbindungen zu anderen Knoten - die meisten Verbindungen zu anderen Knotenzuordnungen haben jedoch keinerlei Einfluß auf den Graphen und werden darum mit '0' markiert. Wird also der Kompatibilitätsgraph in einer Matrix abgebildet, wird sehr viel Speicher für unbedeutsame Matrixelemente verschwendet.

Ein weiterer Nachteil einer Matrix ist die Schwierigkeit, die Matrix bei Bedarf zu vergrößern oder bestimmte Elemente zu entfernen.

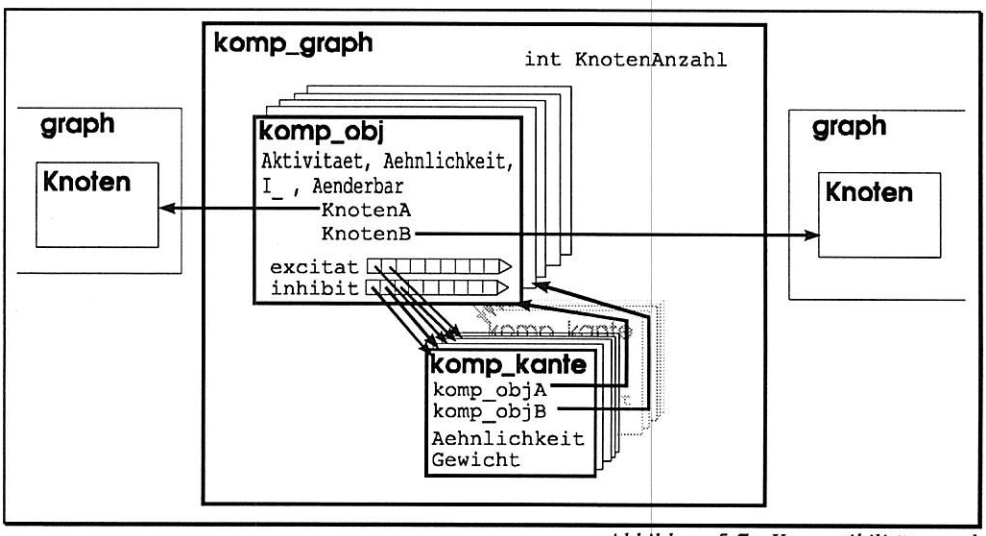



*Abbildung 5-7 - Kompatibilitätsgraph*

Darum wurde für die Implementierung des Kompatibilitätsgraphen ein Modell genutzt, welches Knoten (komp_obj) und Kanten (komp_kante) verwaltet (Abbildung 5-7).

Da, ähnlich der Registrierung von Kanten am Graph, auch beim Kompatibilitätsgraph der Zugriff auf die Kanten hauptsächlich über die Knoten erfolgt, sind an jedem Knoten die excitatorischen und inhibitorischen Kanten in Listen gesammelt. Auch hier sind die Kanten symmetrisch, also eine excitatorische Kante zwischen Al1 und Al2 ist einmal vorhanden und über entsprechende Methoden kann die Sicht von jedem Knoten aus simuliert werden.

Natürlich müssen die Kompatibilitätsgraphknoten gemäß den Abschnitten 3.1 und 3.6 die Ähnlichkeit, den externen Input und die Aktivität speichern können. Zusätzlich ist die (bool´sche) Variable 'Aenderbar' vorgesehen.

Zum WTA- Netz gehören nun die Methoden, welche für die Initialisierung und die Propagation des Netzes zuständig sind. Abschließend ist eine Methode vorzusehen, welche die Lösung

des Netzes, also die Aktivierung der Knoten des Kompatibilitätsgraphen nach einem stabilen Netzzustand in einen Graphen zurückwandelt.

## 5.4 C++ - Programmierschnittstelle

Durch eine Programmierschnittstelle in C++ sollte eine flexible Nutzung in Anwendungsprogrammen möglich sein. Dabei sollten die Anwender bzw. Anwenderinnen einen Zugriff auf alle wichtigen Systemklassen haben, andererseits sollte die Nutzung der Schnittstelle so einfach wie möglich erscheinen.

Daher ist eine Schnittstellenklasse angebracht, welche zwei Ausgangsgraphen sowie einen Ergebnisgraphen enthält. Über ein WTA- Netz können die Ausgangsgraphen zu einem Ergebnisgraphen überführt werden. Alle Methoden zur Graphein- und -ausgabe sowie zur Parameterbeeinflussung sollten dabei der Anwenderin bzw. dem Anwender zur Verfügung stehen.

Diese Methoden sollten unabhängig von ihrer internen Realisierung einfach auf die Klasse angewandt werden können und somit eine einheitliche Bedienungsschnittstelle schaffen.

## 5.5 menügestützte Oberfläche

Für die effektive Nutzung des Gesamtprogramms war eine Erweiterung der Programmstruktur nach Abbildung 5-1 wünschenswert. Da der Ergebnisgraph die gleiche Qualität wie die Ausgangsgraphen hat, sollten alle Graphen generell in einer Menge gehalten werden, zu welcher durch einen Vergleich einfach der Ergebnisgraph hinzugefügt wird. Alle Graphen dieser Menge können ausgegeben oder zum Vergleich genutzt werden.

Außerdem sollten natürlich auch alle Netzparameter über ein Menü komfortabel geändert werden können.

Um die Qualität von Netzparametereinstellungen zu bewerten, können Vergleichsmatrizen von Graphen genutzt werden. Dies bedeutet, daß zwischen allen möglichen Kombinationen zweier Graphen aus einer Menge von Graphen die Ähnlichkeiten bestimmt werden (und nur diese, die Ergebnisgraphen werden ignoriert).

Die erzeugte Matrix hat dann $n$ Zeilen und Spalten ($n$ gleich Anzahl der Graphen). Jedes Matrixelement $m_{ij}$ $(i \leq n, j \leq n, i \neq j)$ enthält dann die drei Vergleichsgüten für den Vergleich von $Graph_i$ und $Graph_j$ gemäß Abschnitt 4. Da eine solche Matrix symmetrisch wäre, wird hier nur der Teil der Matrix mit $m_{ij}$ $(i < n, i < j \leq n)$ berechnet.

Solche Matrizen können dann zur Klassifikation genutzt werden (Abschnitt 10.). Eine komfortable Unterstützung der Matrixbildung, welche auch aufgrund ihres Rechenaufwandes enorme Anforderungen an das System stellt, wurde darum zusätzlich implementiert (Abbildung 5-8). Diese enthält unter anderem Möglichkeiten zum Verteilen des Rechenaufwandes auf mehrere Rechner. Auch dafür war eine Steuerung der Eingabe (für Referenzdateien, siehe 6.5.2.3.) und der Ausgabe (der Matrix oder der Referenzdateien) durch das Menü nötig.

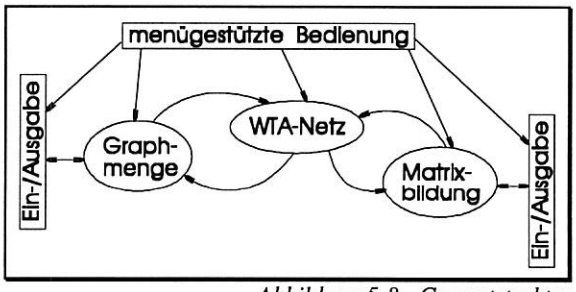



*Abbildung 5-8 - Gesamtstruktur*

# 6 Implementierung

Das gesamte Programm wurde in C++ implementiert - dabei wurde aufgrund der freien Verfügbarkeit der GNU C++ Compiler genutzt. Darum kann die Nutzung des Quellcodes mit anderen Compilern Probleme bereiten oder ist nicht möglich. Da jedoch der GNU- Compiler auf den verschiedensten Plattformen zugänglich ist, dürfte dies im Moment keine wesentliche Einschränkung darstellen- getestet wurde das Programm unter LINUX (gnu g++ v.2.7.0) und UNIX (sparc-sun-solaris 2.6, gnu g++ v.2.7.2.3).

Zur Verdeutlichung der implementierten Programmklassen werden diese in diesem Abschnitt in fünf verschiedene Teile getrennt, welche durch die Rolle der verschiedenen Klassen im Programm beschrieben werden. Diese Trennung erfolgte (zumindest in *dieser* Form) nicht während der Entwicklung des Programms, so daß zwischen den einzelnen Subsystemen durchaus wichtige Verbindungen bestehen und diese teilweise nur *miteinander* sinnvoll zu nutzen sind. (siehe Tabelle 6-1)

| Programmteile |
|---|
| Listenverwaltung |
| Input- System |
| Basissystem |
| Porgrammierschnittstelle |
| Bedienoberfläche |

*Tabelle 6-1 - Programmteile*

Im folgenden werden aus Platzgründen nicht alle beschriebenen Programmdateien abgebildet, dafür wird jeweils der Verweis auf den entsprechenden Abschnitt des Anhangs angegeben, in welchem die kompletten Header- Dateien zu finden sind..

## 6.1 Listenverwaltung

| Dateien | Anhang |
|---|---|
| basis_liste.* | A I |
| menge.* | A II |
| menge_sort.* | A III |
| multiset.* | A IV |
| multiset_sort.* | A V |

*Tabelle 6-2 - Listenverwaltung*

Schon am Anfang der Implementierung wurde deutlich, daß zum Beispiel für Merkmalsausprägungen, aber auch für viele dynamisch zu speichernde Strukturen besondere Listen benötigt werden würden.

Da keine Klassen in den Standardbibliotheken existierten, welche die Ansprüche (zum Beispiel automatische Intervallbildung) an diese Listen erfüllen konnten, war es nötig, eigene Klassen für die Listenverwaltung zu entwickeln.

Alle dafür implementierten Klassen sind von der Klasse 'basis_liste' abgeleitet (siehe Abbildung 6-1), welche die grundlegenden, gemeinsamen Klassenmethoden beinhaltet. So wird die Struktur eines Listenelements in dieser Klasse definiert, eine grundlegende Methode zum Hinzufügen neuer Elemente sowie mehrere Zeiger auf Listenelemente ergänzen diese Klasse.

Die Klasse 'basis_liste' verwaltet fünf interne Zeiger. Zwei davon werden für die Kennzeichnung des Listenanfangs und des Listenendes benötigt, die anderen dienen zur Nutzer(-innen)-gesteuerten Arbeit in den Listen.

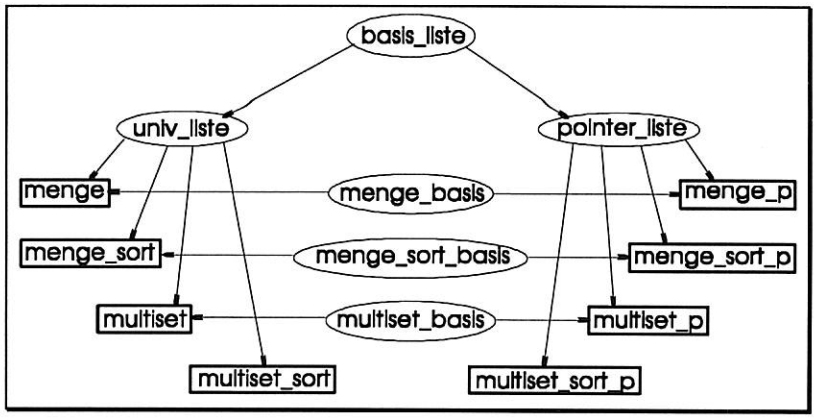



*Abbildung 6-1 - Vererbungsstruktur der Listen*

Da die in den Listen gespeicherten Daten universell verwendbar sein sollten, wurden für sämtliche Implementierungen, welche auf die Daten zugreifen, template- Methoden genutzt.

Weil dadurch vorher nicht bekannt ist, ob die Liste Pointer oder Elemente verwalten soll, treten hier einige Probleme auf. Um zum Beispiel die Elemente in den Listen zu vergleichen, müssen spezielle Vergleichsfunktionen genutzt werden. Die meisten Vergleichsoperatoren lassen sich aber nicht mit Operatoren, welche zwischen Pointern arbeiten, überlagern- zumindest nicht, wenn die ausgeführte Funktion auch noch abhängig vom Ziel des Pointers sein soll.

Aus diesem Grund werden im Rahmen aller Pointer- Sublisten bei Vergleichen oder zur Ausgabe die Pointer dereferenziert. Dies bedeutet, daß die Elemente, auf welche die in der Liste gesammelten Pointer zeigen, übergeben werden. Sind die überlagerten Funktionen beendet, werden nun die übergebenen Parameter vom System gelöscht, das heißt, daß in diesem Moment die Destruktoren der Klassen, auf welche die Pointer zeigten, aufgerufen werden - sofern diese vorhanden sind. Falls also die Speicherfreigabe innerhalb der Destruktoren erfolgt, ist nach einem Aufruf einer solchen überlagerten Funktion mit einem dereferenzierten Pointer das Element gelöscht. *Darum dürfen die Zielklassen der Pointer, welche in diesen Listen gesammelt werden, die Speicherfreigabe nicht in den Destruktoren vornehmen.* Um dennoch eine Speicherfreigabe zu gewährleisten, müssen solche Elemente vor dem Entfernen aus den Listen gelöscht werden.

Ein weiterer Unterschied zwischen den Klassen, welche von 'pointer_liste' und andererseits von 'univ_liste' abgeleitet sind, besteht im Leeren der Liste ('basis_liste::free()'). *Während alle 'univ_liste'- Erben lediglich die Listenelemente löschen, wird durch alle Subklassen von 'pointer_liste' auch noch die Klasseninstanz, auf welche der Pointer in der Liste zeigt, entfernt* ('univ_liste::del()','pointer_liste::del()').

## 6.2 Input- System

| Dateien | Anhang |
|---|---|
| file.* | A VI |
| graph_file.* | A VII |
| terminal.* | A VIII |

*Tabelle 6-3 - Input- System*

Die Klassen zur Eingabe teilen sich in zwei prinzipielle Teile auf - zum einen die Klassen für das Einlesen von Dateien sowie die für die Eingabe von der Standardeingabe aus.

### 6.2.1 Einlesen von Dateien

Die für die Eingabe von Dateien zuständige Klasse 'readfile' enthält alle Methoden, welche zum Einlesen von Dateien nötig sind. Zusätzlich wurde eine Kommentarerkennung (alles von '#' bis '#' ist ein Kommentar und wird ignoriert) sowie eine Statuserkennung implementiert.

Eine weitere Besonderheit dieser Klasse ist die Tatsache, daß über die einzige Zugriffsmethode auf die gelesenen Daten ('nextWord()') komplette Wörter zurückgegeben werden (der Speicher dafür wird mit 'new' vom System angefordert). Dafür wurde eine Wortendeerkennung implementiert. Außerdem können verschiedene Sonderzeichen im Datensatz den Einlesestatus ändern, was wiederum von anderen Programmklassen genutzt werden kann.

| Zeichen | Bedeutung |
|---|---|
| **Zeichen mit Wortendekennung:** | |
| ',' | Worttrennung |
| '(' | Parameteranfang |
| '/' | Merkmalstrennung bei Kantenmerkmalen |
| ':' | Knoten- bzw. Kantenbezeichnerende |
| cr = (int) 10 | Return = Zeilenende |
| **Zeichen ohne Wortendekennung:** | |
| ')' | Parameterende |
| '{' | Konzeptanfang |
| '}' | Konzeptende |
| '[' | Intervallanfang |
| ']' | Intevallende |
| '#' | Kommentaranfang bzw. -ende |

*Tabelle 6-4 - Sonderzeichen*

Direkt genutzt werden diese Stati von der abgeleiteten Klasse 'g_file'. Diese enthält Methoden, welche den nächsten Knotenbezeichner oder den nächsten Kantenbezeichner suchen und zurückgeben. (siehe dazu Syntax der Dateien, Abschnitt 7)

Von der Klasse 'g_file' ist wiederum die Klasse 's_file' abgeleitet, welche zusätzliche Methoden zum Einlesen der Strukturbeschreibung enthält.

### 6.2.2 Einlesen der Standardeingabe

Auch zum Einlesen der Standardeingabe ist eine extra Klasse - 'tty' - implementiert worden. Diese Klasse wird in der Instanz 'term' einfach im Programm gehalten und über diese Instanz werden alle Eingaben abgewickelt. Dies war nötig, da festgestellt wurde, daß unter bestimmten Bedingungen die in den Standardbibliotheken implementierten Versionen für die Eingabe nicht problemlos funktionierten.

Ein weiterer Grund war der undefinierte Zustand des Terminals, auf dem das Programm läuft. Zwei Zustände sind an dieser Stelle interessant: bei einem werden alle Zeichen, welche am Terminal eingegeben werden, sofort und ohne Berücksichtigung von Sonderzeichen an das Programm gegeben. Dies ist sinnvoll und nötig, wenn zum Beispiel das Menü mit einem Tastendruck bedient werden soll. Im alternativen Zustand wird erst beim Zeilenendezeichen die komplette Zeile vom Terminal zum Programm geschickt, was den Vorteil hat, daß Steuerzeichen erkannt werden können und darum eine Zeile mit den üblichen Tasten solange editiert werden kann, bis Return gedrückt wird - also sinnvoll und nötig, falls zum Beispiel Dateinamen eingegeben werden sollen.

Da kein definierter Programmstartstatus vorliegt und eine Umschaltung anhand der aktuellen Erfordernisse sinnvoll erscheint, wurde diese Klasse implementiert.

Sie enthält alle Methoden zur Eingabe von Daten und zusätzlich einige Möglichkeiten, um Bildschirmausgaben zu beeinflussen oder Makrodateien einzulesen.

## 6.3 Basissystem

Zum Basissystem gehören alle die Programmteile, welche direkt für die Arbeit des WTA- Netzes zuständig sind. Das sind zum einen alle Klassen, welche das Netz direkt beschreiben. Zum Basissystem gehören jedoch auch die Klassen zur Beschreibung der Graphen, zur Beschreibung der Markierungen und deren Vergleiche sowie Generalisierungen.

Zum Verständnis der Implementierung des WTA- Netzes ist es sinnvoll, bereits eine Vorstellung über die Graphimplementation sowie den Markierungsvergleich zu haben - daher werde ich hier erst auf die Graphen, dann auf die Vergleiche und abschließend auf das Netz eingehen.

### 6.3.1 Graphimplementierung

| Graphimplementierung |
|---|
| merkstrukt.* |
| graph.* |
| knoten.* |
| kanten.* |

*Tabelle 6-5 - Graphimplementierung*

#### *6.3.1.1 Markierungsstrukturbeschreibung*

| Dateien | Anhang |
|---|---|
| merkstrukt.* | A IX |

*Tabelle 6-6 - Strukturbeschreibung*

Bevor mit Graphen gearbeitet werden kann muß gemäß Abschnitt 5.1.1. der Aufbau der Markierungen von Knoten und Kanten aus Basismerkmalen beschrieben werden. Erst dann können Graphen eingegeben oder generalisiert werden, da an den Knoten und Kanten dieser Graphen die Ausprägungen der Markierungen gespeichert werden müssen, und dies nur möglich ist, wenn deren Größe und Struktur vorher festgelegt wurde.

Diese Struktur muß immer aus einer Datei eingelesen werden, was durch Methoden in 's_file.*' unterstützt wird.

Da es im Graph drei verschiedene Markierungen gibt (siehe Abschnitt 5.1 - eine für Knoten, eine für ungerichtete und eine für gerichtete Kanten) sind in der Klasse merkmal_struktur alle wesentlichen Beschreibungen 3- fach vorhanden. Zusätzlich zur Information, welche Basismerkmale zu Gesamtmarkierungen zusammengefaßt werden, werden auch die jeweiligen Parameter für die entsprechenden Basismerkmalsvergleiche in dieser Klasse gespeichert.

Für alle gespeicherten Informationen existieren Methoden, welche diese anhand ihrer Position innerhalb der Markierung übergeben.

#### *6.3.1.2 Graphimplementierung*

| Dateien | Anhang |
|---|---|
| graph.* | A X |
| knoten.* | A XI |
| kanten.* | A XII |

*Tabelle 6-7 - Graphbeschreibung*

Die Graphimplementierung selbst besteht, wie schon in Abschnitt 5.2 erwähnt, aus Zeigern auf Knoten und Kanten.

Weil die Größe der Graphen fest ist (das heißt zu einem Graphen kommen während der Abarbeitungszeit keine Knoten oder Kanten hinzu - dies wäre ein neuer Graph) erschien es sinnvoll, die Knoten- bzw. Kantenzeiger nicht in Listen, sondern in einem Array zu speichern.

Da andererseits die Größe des Graphen natürlich frei bestimmbar sein soll, enthält der Graph vor seiner Initialisierung nur Pointer, welche später auf zu erstellende Arrays zeigen können. Zusätzlich werden die Anzahl der Knoten und Kanten registriert (um ein Überschreiten des Arrays zu verhindern) und zwei Positionszeiger unterstützen die Arbeit in diesen Arrays.

Wenn wir die allgemeine Programmstruktur betrachten (Abbildung 5-1) fällt auf, daß Graphen entweder aus Dateien gelesen werden oder durch Graphrekonstruktion aus dem Kompatibilitätsgraphen gewonnen werden können. Für beide Situationen wurden hier Methoden implementiert.

Die Größe der Markierungen von Knoten und Kanten des Graphen müssen bereits in der Markierungsbeschreibung festgelegt sein, daher können Knoten und Kanten sofort bei der Grapherstellung in ihrer Größe festgelegt werden. Die einzelnen Merkmale, welche die Markierungen bilden, werden bei Erstellung von Knoten bzw. Kanten initialisiert.

### 6.3.2 Merkmale

| Dateien | Anhang |
|---|---|
| basismerkmal.* | A XIII |
| m_symbol.* | A XIV |
| m_zahl.* | A XV |
| m_gmenge.* | A XVI |
| gmenge.* | A XVII |
| m_hierarchie.* | A XVIII |
| hierarchie.* | A XIX |

*Tabelle 6-8 - Merkmale*

Alle implementierten Merkmalklassen sind (und alle neuen werden) von der Klasse Basismerkmal abgeleitet. Diese Klasse enthält lediglich abstrakte virtuelle Methoden. Die einzige Funktion dieser Klasse ist die sich dadurch ergebende Möglichkeit, alle Merkmalsklassen auf einer (dieser) höheren Abstraktionsebene gleich zu behandeln. In den Knoten und Kanten werden für die Markierungen nur Zeiger auf Basismerkmale bereitgestellt und berücksichtigt, das tatsächliche Ziel dieser Zeiger, ein einzelnes spezielles Merkmal, ist dabei unwichtig.

```
class Basismerkmal
{
 public:
   virtual bool addIntervall(char* a, char* b)
     {
       // Intervall in Beschreibung fuer Merkmal ohne Intervalle
       cout << "\nWARNING: Intervall [" << a << "," << b
         << "] kann dem Merkmal nicht zugeordnet\n"
         << "          werden und wird ignoriert\n";
       return 1;
     };
   virtual void addMerkmal(char*) = 0;
   void gibAusMerkmal(void)
     {gibAusMerkmal(cout);};
   virtual void gibAusMerkmal(ostream) = 0;
   virtual ~Basismerkmal(void) = 0;
};
```

*Programm 6-1 - Klasse basismerkmal*

Die Klasse Basismerkmal (Programm 6-1) zeigt damit sehr deutlich die Anforderungen an die Merkmalsklassen. Diese müssen eine Methode zum Hinzufügen von Einzelausprägungen haben und eine weitere Methode, welche das Merkmal (in welcher Form auch immer) nach 'ostream'

ausgibt. Soll das Merkmal Intervalle berücksichtigen, so ist zusätzlich eine Methode zum hinzufügen von Intervallen zu implementieren. Wichtig ist außerdem bei allen Merkmalsklassen ein Destruktor, welcher den belegten Speicher beim Entfernen der Klasse freigibt.

Dennoch sind bisher nicht alle Anforderungen an die Merkmalsklassen erwähnt. Für Vergleiche ist es zusätzlich wichtig, eine Funktion zu implementieren, die als Resultat des Vergleichs eine reelle Zahl liefert. Und für die Generalisierung ist ebenfalls eine zusätzliche Funktion vorzusehen, welche die entsprechenden Ausprägungen zum generalisierten Merkmal generalisiert. Diese Funktionen sind jedoch nicht auf der übergeordneten Ebene des Basismerkmals nötig, sondern nur speziell bei allen Merkmalen (siehe 6.3.2.x und 6.3.3.)

### *6.3.2.1 Symbolische Merkmale*

| Dateien | Anhang |
|---|---|
| m_symbol.* | A XIII |

*Tabelle 6-9 - symbolische Merkmale*

Die einfachste Implementierung eines Merkmals ist die des symbolischen Merkmals. Die Klasse enthält, wie in Programm 6-2 zu sehen, lediglich die geforderten Methoden (Intervalle werden nicht unterstützt) und bietet der Vergleichs- und der Generalisierungsfunktion Zugriff auf die Symbole, welche in einer Menge gesammelt werden.

```
class symbl
{
 public:
   char* Name;
};

class symbol_merkmal : public Basismerkmal
{
   menge_p<symbl*> daten;
 public:
   void addMerkmal(char*);
   void gibAusMerkmal(ostream);
   friend float cmp(symbol_merkmal*, symbol_merkmal*);
   friend float cmpObjKon(symbl*, symbol_merkmal*);
   friend float cmpKonKon(symbol_merkmal*, symbol_merkmal*);
   friend void general(symbol_merkmal*, symbol_merkmal*, symbol_merkmal*);
   ~symbol_merkmal(void);
};

float cmp(symbol_merkmal*, symbol_merkmal*);
void general(symbol_merkmal*, symbol_merkmal*, symbol_merkmal*);

int operator ==(class symbl a, class symbl b);
void operator <<(class ostream ausgabestrom, class symbl a);
```

*Programm 6-2 - Klassen für Symbole*

Da die Einzelsymbole in einer eigenen Klasse gespeichert werden ('symbl'), wird als Menge der Symbole eine Pointer- Liste verwaltet (siehe Programm 6-2). Daher können zum Vergleich von Elementen dieser Mengen überladene Operatoren genutzt werden, in diesem Fall

`'int operator ==(class symbl a, class symbl b)'`

Diese Operator ist als Stringvergleich der Symbole implementiert. Wäre statt dessen eine Menge von 'char*' verwaltet worden, so würde Gleichheit bei identischem Pointerziel bestätigt, und nicht bei identischen Strings.

Zum Verständnis der Generalisierungsfunktion ist vielleicht wichtig zu erwähnen, daß immer aus den beiden ersten Merkmalen das dritte Merkmal generalisiert wird, was heißt, daß diesem Merkmal im Verlauf der Generalisierung die 'passenden' Ausprägungen zugeordnet werden.

### 6.3.2.2 *reelle und ganzzahlige Merkmale*

| Dateien | Anhang |
|---|---|
| m_zahl.* | A XV |

*Tabelle 6-10 - Merkmale für Zahlen*

Dieses Merkmal besteht im wesentlichen aus einer Liste, welche die Zahlen sammelt. Das Merkmal wurde als template- Klasse implementiert, da auf diese Weise genau die gleiche Implementation für reelle wie für ganze Zahlen genutzt werden kann.

Weiterhin ist wichtig, daß die Unterscheidung zwischen fuzzy und statistischen Merkmalen (siehe Abschnitte 2.3.1 und 2.4.1) nicht durch verschiedene abgeleitete Klassen vorgenommen wurde, sondern anhand eines Enumerators. Dies ist sinnvoll, da die Methoden der Klasse keinen Bezug auf den Enumerator legen, sondern diese Information nur für die (externen) Vergleichs- und Generalisierungsfunktionen wichtig ist. Die abgeleiteten Klassen wären also völlig identisch gewesen, und die in 'kanten.cpp', 'knoten.cpp', 'cmp.cpp' und 'general.cpp' zu verwaltenden Merkmalsarten wären zusätzlich erhöht wurden.

Ansonsten enthält die Klasse alle eben beschriebenen Methoden, zusätzlich noch eine ('getListe()') für die Übergabe der Datenliste, da diese Merkmalsklasse auch in der Merkmalsklasse für geordnete Mengen genutzt wird.

### 6.3.2.3 *Merkmale geordneter Mengen*

| Dateien | Anhang |
|---|---|
| m_gmenge.* | A XVI |
| gmenge.* | A XVII |

*Tabelle 6-11 - geordnete Mengen*

Um geordnete Mengen zu bewerten, reicht es nicht, ein Element zu betrachten. Es muß allgemein (oder hier: im System) bekannt sein, wie diese geordneten Mengen aufgebaut sind - die Ordnung muß beschrieben sein.

```
class gmenge_elem
{
 public:
   char* Begriff;  // aus Datei gelesener Name, muss am Ende durch
                   // free() geloescht werden
   int Rang;
   void free(void);
};

class gmenge
{
   char* Name;                              // Name der geordneten Menge
   multiset_sort_p<gmenge_elem*> *GMengeListe;// Liste mit Elementen
 public:
   gmenge(char*);                           // Konstruktor (Parameter Name)
   int initGMenge(void);                    // liest die gMenge aus der Datei
   char* getName(void) {return Name;};      // gebe gMengenName zurueck
   char* getBegriff(const int);             // suche Name zu Rang
   int getRang(const char*);                // suche Rang zu Name
   void gibAusGMenge(void);                 // gebe geordnete Menge aus
   void free(void);                         // gib belegten Speicher frei
};
```

*Programm 6-3 - Programmausschnitt gmenge.h*

Dies wird auch in der Beschreibung des Merkmals 'm_gmenge.h' deutlich. Zum einen existiert ein Merkmal, zum anderen aber noch eine Sammlung geordneter Mengen. Da - sofern das oben genannte Hintergrundwissen über die geordnete Menge existiert - jedem Element eine

ganze Zahl zugeordnet werden kann (siehe Abschnitt 2..3.3) besteht das Merkmal hauptsächlich aus einem Merkmal für ganze Zahlen und einigen zusätzlichen Methoden, mit denen die übergebenen Elemente oder Parameter beeinflußt werden können.

Weil dieses Merkmal ansonsten keine besondere Struktur hat, dürften zum Verständnis die Beschreibungen in den vorangegangenen Abschnitten ausreichen.

Wichtiger ist an dieser Stelle wohl der Aufbau der Sammlung der Beschreibung der geordneten Mengen. Diese enthält einfach eine Liste (Menge) mit Beschreibungen geordneter Mengen, wie sie in der Klasse 'gmenge' enthalten sind (Programm 6-3).

Die Klasse 'gmenge' wiederum besteht aus einer Liste, in der alle Elemente der geordneten Menge mit ihrem Index bzw. Rang registriert sind. Ergänzende Methoden erlauben das Ermitteln des Ranges zu einem gegebenen Begriff oder umgekehrt.

### *6.3.2.4 Merkmale einer Begriffshierarchie*

| Dateien | Anhang |
|---|---|
| m_hierarchie.* | A XVIII |
| hierarchie.* | A XIX |

*Tabelle 6-12 - Begriffshierarchie*

Ebenso wie bei einer geordneten Menge ist auch bei Begriffen aus einer Hierarchie nur dann eine sinnvolle Verarbeitung möglich, wenn irgendwo im System die nötige Hintergrundinformation zur Verfügung steht. Das Merkmal an sich enthält nur den Namen der Hierarchie und den Begriff, also die aktuelle Merkmalsausprägung.

Die systemweite Sammlung der Hierarchien wird adäquat zu 6.3.2.3 durch eine Menge von Pointern auf 'hierarchie'- Klassen dargestellt. Prinzipiell waren für die Entwicklung der internen Hierarchieabbildung zwei Ansprüche entscheidend, zum einen sollte zu einem Begriff die Stufe in der Hierarchie angegeben werden können ('int stufe(char*)'), zum anderen sollte zu zwei Begriffen ein generalisierter Begriff zurückgegeben werden ('char* general(char*,char*)').

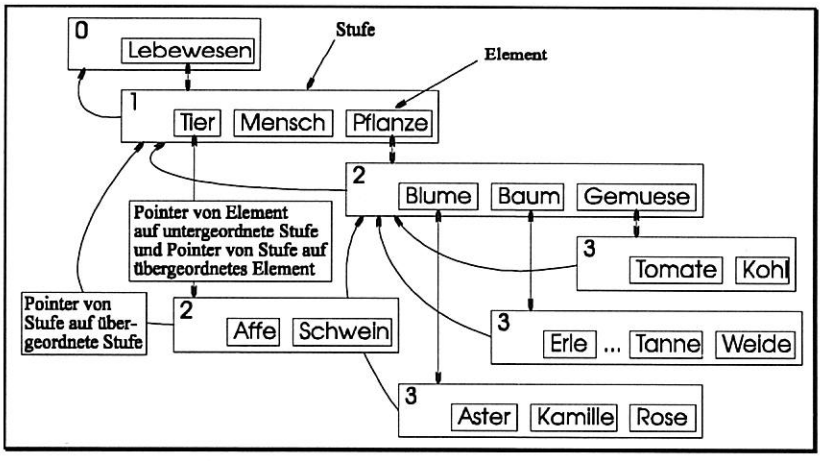



*Abbildung 6-2 - interne Hierarchiedarstellung*

Die implementierte Klasse verwaltet eigentlich eine Liste mit Hierarchieelementen. Jedes dieser Elemente entspricht einem Begriff. Zusätzlich wird nun angenommen, daß jedes Hierarchieelement gleichzeitig Teil einer Hierarchiestufe ist. Diese (gesamte) Hierarchiestufe mit mehreren Begriffen verweist auf einen Oberbegriff, der selbst wiederum Element einer Stufe ist, welche ebenfalls einem Oberbegriff zugeordnet ist (sofern dies nicht die Wurzel ist). Außerdem kann das oben bezeichnete Element natürlich - wie jedes andere Element auch - ebenfalls ein Oberbegriff einer niedrigeren Hierarchiestufe sein (siehe dazu auch Abbildung 6-2)

Auf diese Weise konnte eine Struktur implementiert werden, welche sehr schnell Generalisierungen zuläßt oder die Positionen der Stufen anzugeben vermag.

### 6.3.3 Vergleich und Generalisierung

| Dateien | Anhang |
|---|---|
| cmp.* | A XX |
| general.* | A XXI |

*Tabelle 6-13 - Vergleich und Generalisierung*

Da der prinzipielle Aufbau der Vergleichs- und Generalisierungsverfahren gleich ist, werde ich im folgenden erst auf den Vergleich eingehen und lediglich am Ende des Abschnittes einige Anmerkungen zu den Generalisierungsverfahren geben.

Vordergründig ist beim Vergleich das Problem, daß der Vergleich zweier Knoten bzw. zweier Kanten keiner einzelnen Klasseninstanz zuzuordnen ist - er erfolgt bei Bilden des Kompatibilitätsgraphen als Vergleich *zwischen* zwei Graphen.

```
float cmp(knoten_struktur*, knoten_struktur*);

float cmp(kanten_struktur*, char*,
          kanten_struktur*, char*);
```

*Programm 6-4 - Vergleichsfunktion*

Genau so unabhängig sind die Vergleichsfunktionen auch aufgebaut. Die eine Funktion erhält zwei Knotenstrukturen und gibt die Ähnlichkeit als reelle Zahl zurück, die andere bekommt zwei Kantenstrukturen und deren Ausgangspunkte (siehe Programm 6-4) und liefert ebenfalls ein reelles Ergebnis.

```
float cmp(const Basismerkmal* a, const Basismerkmal* b,
          const int MerkmalPos, const int Typ)
{
   if (t) cout << ">>> cmp cmp(Basismerkmal*,Basismerkmal*,int,int)\n";
   switch(merkstruct->Merkmal(MerkmalPos,Typ))
     {
//     ...
       case reell_2:
        if (v) cout << "reell ";
        return cmp((merkmal<float>*)a,(merkmal<float>*)b,
                   merkstruct->Param(MerkmalPos,Typ));
       case ganz_1:
        if (v) cout << "ganz ";
        return cmp((merkmal<int>*)a,(merkmal<int>*)b,
                   merkstruct->Param(MerkmalPos,Typ));
//     ...
     }
   if (v) cout << "MerkmalTyp nicht in Verteiler cmp.cpp:cmp(Basismerkmal*,"
     << "Basismerkmal*, int, int)\n";
   return 0;
};
```

*Programm 6-5 - Ausschnitt aus cmp.cpp*

Um diesen Vergleich zu realisieren, müssen natürlich alle Einzelmerkmale miteinander verglichen und die Ergebnisse entsprechend der Merkmalsstruktur gewichtet werden. Da hierfür eine etwas schwer durchschaubare Pointertransformation vorgenommen werden mußte, soll dies genauer erläutert werden. Vergleiche von Kanten werden nicht speziell erläutert, da diese nach derselben Methode ablaufen.

Die Knotenmarkierungen sind, wie in Abschnitt 6.3.2 erwähnt, Arrays mit Pointern auf Basismerkmale. Sollen nun zwei dieser Basismerkmale verglichen werden, kommt es darauf an, die

richtige Vergleichsfunktion - nämlich die für genau diese speziellen Merkmalsarten - aufzurufen.

Diese Funktion übernimmt hierbei 'cmp(basismerkmal*.....)', (siehe Programm 6-5). Anhand der Beschreibung innerhalb der Markierungsstruktur werden die Pointer auf Basismerkmale zu Pointern auf die speziellen Merkmalsklassen transformiert und danach mit den genau diesen Merkmalen zugeordneten Vergleichsfunktion verglichen. (Der Integer 'Typ' gibt hierbei nur an, ob es sich um Knoten-, Kanten- oder gerichtete Kantenmerkmale handelt).

Anschließend werden alle Einzelvergleichsergebnisse zu einem Gesamtergebnis verknüpft und als Resultat zurückgegeben. Die Generalisierung erfolgt - wie anfangs erwähnt - mit denselben Prinzipen der Pointerkonversion, da nur so eine Lösung des oben geschilderten Problems möglich war (siehe Programm 6-6).

```
void general(knoten_struktur*, knoten_struktur*, knoten_struktur*);

void general(kanten_struktur*, char*,
             kanten_struktur*, char*, kanten_struktur*);
```

*Programm 6-6 - Generalisierung*

Hierbei wird jedoch nicht eine Zahl zurückgeliefert, sondern der letzte übergebene Strukturparameter enthält nach Aufruf der Funktion die Generalisierung der ersten beiden.

### 6.3.4 Das Netz

| Dateien | Anhang |
|---|---|
| komp_graph.* | A XXII |
| wta_params.* | A XXIII |
| wta_init.* | A XXIV |
| wta_iterate.* | A XXV |
| wta_netz.* | A XXVI |

*Tabelle 6-14 - WTA-Netz*

Das WTA- Netz besteht aus fünf Teilklassen, welche voneinander abgeleitet sind. Obwohl die wesentliche Nutzung dieser Klassen nur in Ihrer Gesamtheit besteht, kann durch diese Vererbung deutlich zwischen verschiedenen Funktionszusammenhängen unterschieden werden. (Abbildung 6-3)

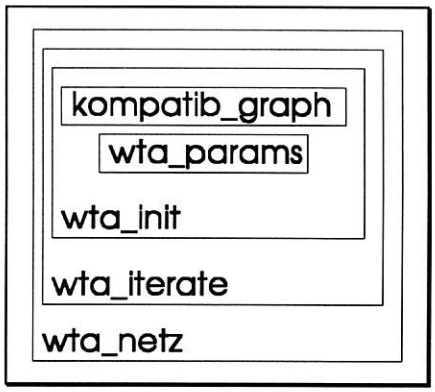



*Abbildung 6-3 - Klassenstruktur*

#### *6.3.4.1 Die Klasse 'kompatib_graph' (komp_graph.*)*

Der Kompatibilitätsgraph besteht, wie schon im Abschnitt 5.3 dargestellt, aus zwei Listen mit Zeigern auf Knoten bzw. Kanten. Zusätzlich sind noch einige Informationen über die Anzahl der Knoten bzw. Kanten und über die Größe der Graphen, welche zum Bilden des Kompatibilitätsgraphen genutzt wurden, gespeichert.

Die Kompatibilitätsgraphknoten enthalten zusätzlich eine bool´sche Variable und einen Wert für den externen Input. Auch die Ähnlichkeit der Graphknoten wird im Kompatibilitätsgraphknoten vermerkt. Die Knotenaktivität ist ein Feld mit zwei Werten, da so bei der Berechnung der neuen Aktivitäten (für das gesamte Netz) die alten Werte noch zur Verfügung stehen. Außerdem sind zwei Listen für excitatorische und inhibitorische Kanten in den Knoten vorhanden. Während bei inhibitorischen Kanten keine weiteren Informationen als 'Start' und 'Ziel' wichtig sind, müssen excitatorische Kanten zusätzlich das Gewicht ($\tilde{w}_{ij}$) und die Ähnlichkeit ($sim_{ij}$) der Kante registrieren (siehe auch Abschnitt 3.).

Die wichtigsten Zugriffsmethoden ('initKompatGraph(...)' und 'getErgGraph()') spiegeln genau die Anforderungen wieder, welche an einen Kompatibilitätsgraph gestellt werden müssen - eine bildet zwei Graphen auf den Kompatibilitätsgraphen ab, die andere erzeugt aus dem Kompatibilitätsgraph einen Ergebnisgraphen. Außerdem ist noch eine Methode implementiert, welche aus zwei Knotennamen einen neuen formt und eine Methode, welche den Kompatibilitätsgraph ausgibt.

#### *6.3.4.2 **Die Klasse 'wta_params'** (wta_params.*, enthält Vergleichsgüte)*

Diese Klasse 'wta_params' enthält alle Netzparameter, welche für den Iterationsablauf bzw. die Abbildung der Ähnlichkeit auf die Parameter des Netzes wichtig sind. Natürlich sind auch alle Methoden zur Änderung dieser Parameter implementiert und zusätzlich eine Methode, welche genau die aktuelle Parameterklasse kopiert (mit allen aktuellen Einstellungen) und als neue Klasse zurückgibt. Dies kann sinnvoll genutzt werden (in 'metaclass'), um sich zu einem erzeugten Graphen die Parametereinstellungen zu merken. Da dies vor allem *nach* einem Vergleich wichtig ist, werden in dieser Klasse auch die Vergleichsgüten nach Abschnitt 4 registriert und können so dem Graph zugeordnet werden.

#### *6.3.4.3 **Die Klasse 'wta_init'** (wta_init.*)*

Die Initialisierung der Neuronen kann in zwei Teile geteilt werden. Zum einen müssen diese mit einer Anfangserregung versehen werden, was 'initAnfAkt()' übernimmt. Außerdem müssen in diesem besonderen Fall eines WTA- Netzes die externen Inputs sowie alle Kantenskalierungen den geforderten Einstellungen angepaßt werden. Dies erledigt die Methode 'init_I_w()'.

Die Methode 'initAll()' ruft lediglich beide eben genannten Methoden nacheinander auf.

#### *6.3.4.4 **Die Klasse 'wta_iterate'** (wta_iterate.*)*

Die wichtigste Zugriffsmethode der Klasse 'wta_iterate' ist 'int iterate(int, float)' Weil diese Methode den Kern des Programms darstellt und entsprechend oft aufgerufen wird, wurde versucht, den Rechenaufwand etwas zu verringern, was leider die Übersichtlichkeit des Programmcodes beeinträchtigt hat. Da bezüglich des Rechenaufwandes Verbesserungen wichtig und sinnvoll erscheinen, werde ich bei dieser Methode etwas genauer auf die Anordnungen der einzelnen Aufrufe eingehen. Dazu werde ich eine Pseudoprogrammiersprache nutzen, da so am anschaulichsten die wichtigen Elemente hervorgehoben werden können.

Der implementierte Ablauf ist als Ergebnis einer Änderung zu betrachten, welche folgendes (prinzipielles) Programm als Ausgangspunkt hatte:

```
iterate()
{
   wiederhole bis Abbruch
   beruecksichtige r und Energie
   {
      neues r;
      *{berechne Zmax}; // Berechnung der Schrittweite
      neues w;
      *{verschiebe Aktivitaet}; // (Aktivitaet[1] = Aktivitaet[0])
      *{fuege Rauschen hinzu};
      updateNet();
      *{Energieberechnung};
   }
}
```

*Programm 6-7 - Prinzip 1*

Wie in Programm 6-7 sieht üblicherweise der Update- Algorithmus des WTA- Netzes aus. Die Kernprozedur ist dabei 'updateNet()', in ihr werden die neuen Knotenaktivierungen anhand der excitatorischen und inhibitorischen Verbindungen laut Update- Vorschrift berechnet (siehe Formel 3-17).

Die Aufrufe vor 'updateNet()' bereiten das Netz für das nächste Update vor - zum Beispiel werden die Parametereinstellungen aktualisiert. Nachdem die Neuronenaktivitäten neu berechnet wurden, muß noch die Energie des Netzes berechnet werden und ein kompletter Durchlauf ist erfolgt.

Für alle mit '*' markierten Aufrufe muß das Programm dabei alle Knoten aus der Knotenliste des Kompatibilitätsgraphen holen, um diese zu bearbeiten oder deren Informationen zu nutzen. Weil eine solche Abarbeitung in der Summe zu rechenzeitaufwendig war, wurden einige Aufrufe zusammengefaßt und die Energieberechnung erfolgt jetzt gleichzeitig mit der Vorbereitung des Netzes für den nächsten 'updateNet()'- Aufruf (Programm 6-8).

```
changeEnvironment()
{
   neues r (merke altes r in old_r);
   *{
      Energieberechnung;
      berechne Zmax; // Berechnung der Schrittweite
      verschiebe Aktivitaet;
      fuege Rauschen hinzu;
    }
   neues w;
}

iterate()
{
   changeEnvironment();
   wiederhole bis Abbruch
   beruecksichtige old_r und Energie
   {
      updateNet();
      changeEnvironment();
   }
   restoreErgebnis(); // da Aktivitaeten schon verschoben
}
```

*Programm 6-8 - Prinzip 2*

Aufgrund dieser Änderungen kann nun erst nach der neuen Netzvorbereitung entschieden werden, ob die Berechnungen abgebrochen werden sollen, daher muß nach einem Abbruch der vorherige Netzstatus wieder rekonstruiert werden.

In den gewählten Bezeichnungen kommen nun auch schon die Rollen der einzelnen Methoden dieser Klasse zum Ausdruck. Die Methode 'updateNet()' berechnet die neuen

Neuronenaktivitäten und die Methode 'changeEnvironment()' stellt die neuen Parameter ein und berechnet gleichzeitig die Energie des Netzes (mit Hilfe der Methode 'newEnergyZmax()').

In Programm 6-8 ist auch ein Punkt zu sehen, welcher bisher noch nicht besprochen wurde- in jedem Iterationsdurchlauf wird auf alle Neuronen des Netzes ein Rauschen gelegt.

Bei teilweise symmetrischen Graphen können die Neuronen im Netz absolut identische Umgebungen haben. Falls diese Neuronen nun zusätzlich noch mit einer inhibitorischen Kante verbunden sind, kann erst bei einem minimalen Potentialunterschied zwischen den Neuronen eine bessere, weitergehende Lösung gefunden werden, sonst behindern sich beide Lösungen. Um genau diese minimalen Potentialunterschiede zu erzeugen, wurde ein Rauschen geringer Intensität auf alle Neuronenaktivitäten gesetzt- so wird sichergestellt, daß fast nie zwei Neuronen exakt die gleiche Aktivität besitzen. Folgende Vorschrift wird für das Rauschen genutzt:

Aktivität += (Aktivität * drand48()) / 1e7;

Die Funktion drand48() liefert dabei einen Zufallswert aus dem Intervall [0,1), der Wert 1e7 dient zur Skalierung des Rauschens und wurde empirisch durch Vergleich der Ergebnisse dieses Netzes mit denen einer Matrixabbildung des Netzes ermittelt.

Das Hinzufügen des Rauschens entspricht im übrigen einem biologischen Verständnis der Neuronen- da natürliche Potentiale immer analog sind, gibt es auch keine absolute Gleichheit der Neuronenpotentiale.

#### *6.3.4.5* ***Die Klasse 'wta_netz'*** *(wta_netz.*, enthält Programmversionsnummer)*

In dieser Klasse ('wta_netz') ist die eigentliche Schnittstelle des Netzes implementiert ('initIterate(graph*,graph*)'). Durch den Aufruf dieser Methode werden der Kompatibilitätsgraph und alle dadurch erzeugten Neuronen initialisiert. Danach wird das Netz bis zur Stabilität iteriert. Dies wird für schrittweises Matching auch mit verschiedenen Schwellwerten bei der Kompatibilitätsgraphbildung wiederholt, falls es gewünscht war. Abschließend werden über eine interne Methode ('calcGuete()') die 3 in Abschnitt 4 beschriebenen Gütemaße berechnet und in wta_params gespeichert.

Die Methode liefert die Anzahl der erfolgten Berechnungen zurück. Wurde die intern gesetzte maximale Berechnungsanzahl erreicht (10 * Neuronenanzahl), wird 0 zurückgegeben.

Zusätzlich enthält diese Klasse noch eine Methode zur Ausgabe der Neuronenaktivitäten.

In der Datei 'wta_netz.cpp' wird noch die Programmversionsnummer geführt, welche unter anderem im Menü oder in Kommentaren gespeicherter Graphen angegeben wird.

## *6.4 Programmierschnittstelle*

| Dateien | Anhang |
|---|---|
| wta_work.* | A XXVII |

*Tabelle 6-15 - Programmierschnittstelle*

Die Programmierschnittstelle enthält alle für die Arbeit mit dem Programm nötigen Basisklassen (vergleiche Programm 6-9) - ein WTA- Netz sowie zwei Eingabegraphen und einen Ausgabegraph.

Mit den zur Klasse gehörenden Methoden kann das Netz bedient werden. Außerdem sind Methoden vorhanden, mit denen alle in Abschnitt 3 beschriebenen Verfahren der Ähnlichkeitsabbildung ausgewählt werden können. Zum genauen Verständnis der Nutzung dieser Schnittstelle empfiehlt es sich, die Bedienungsbeschreibung in Abschnitt 8.1 zu berücksichtigen.

```
enum s_art{lose,fixiert,reduziert};
enum s_was{knot,kant,beide};
enum anf_art{interaktiv,graphGroesse,fest,aehnlichkeit,listenwerte};
enum exI_art{ohne,prozentGraphGroesse,anzahlKanten};
enum kanSk_art{keine,kanten,kanten_u_knoten};

extern class wta_work work;

class wta_work
{
   class wta_netz *Netz;
   class graph *graphA, *graphB, *ErgGraph;
 public:
   wta_work(void);
// Ein-/ Ausgabemethoden
   bool leseStruktur(char*);
   bool leseGraphA(char*);
   bool leseGraphB(char*);
   int compare(void);
   void speicherErgebnisGraph(char*);
// Kontrollmethoden
   void gibAusStruktur(void);
   void gibAusGraphA(void);
   void gibAusGraphB(void);
   void gibAusErgGraph(void);
   void gibAusParams(void);
   void gibAusAnfAktListe(void);
// Methoden zur Parameteraenderung
   void setAnfAktArt(enum anf_art);
   void setInit(float);
   void resetAnfAktListe(void);
   bool addAnfAkt(char*,char*,float);
   void setExtInpArt(enum exI_art);
   void setSkal(float);
   void setKanSkalArt(enum kanSk_art);
   void setSchrittArt(enum s_art);
   void setSchrittWas(enum s_was);
   void setAnzBerech(int);
   void setS(float);
   void setKnotenSchwelle(float);
   void addKnotenSchwelle(float);
   void setKantenSchwelle(float);
   void addKantenSchwelle(float);
   ~wta_work(void);
};
```

*Programm 6-9 - wta_work.h*

## 6.5 Bedienoberfläche

| Dateien | Anhang |
|---|---|
| menu.* | A XXVIII |
| metaclass.* | A XXIX |
| metawta.* | A XXX |
| matrix.* | A XXXI |
| main.cpp | A XXXII |

*Tabelle 6-16 - Bedienoberfläche*

Die menügestützte Oberfläche wurde auf das bestehende System aufgesetzt, um allgemeine Abläufe in der Arbeit mit dem WTA- Netz bedienungsfreundlicher zu gestalten.

### 6.5.1 Die Klasse 'menu' (menu.*)

Für die Erzeugung eines Menüs bot sich die Erstellung einer eigenen Klasse an, da so immer wiederkehrende Formatierungsaufrufe an einer Stelle strukturiert programmiert werden konnten. Dies erhöhte zum einen die Funktionssicherheit, andererseits auch die Änderbarkeit sowie die Lesbarkeit der Programme.

```
Bitte waehlen Sie (Cursor up/down + Return oder Taste)

   ** Titel ***********
   *                   *
   *       Subtitel    *
   *     a - Punkt1    *
   *     b - Punkt2    *
   *                   *
   *     c - Punkt3    *
   *                   *
   ************ V0.97 *

Titelfunktionsausgabe

   ********************
```

*Ausgabeprotokoll 6-1 - Menü*

Das in der Klasse menu implementierte Menü besteht aus verschiedenen Menüpunkten und kann mit den Cursortasten oder durch Betätigung eines Kurzbezeichners bedient werden. Ein Teil der implementierten Methoden unterstützt die Initialisierung von sämtlichen Menüpunkten, dem Menü kann ein Titel gegeben werden und eine Funktionsausgabe kann in das Menü integriert werden (Ausgabeprotokoll 6-1).

Zur Nutzung des Menüs sind zwei Methoden vorhanden. Die eine gibt das Menü auf dem Bildschirm aus ('gibAusMenu()'), die andere ist eine Abfragemethode welche solange verweilt, bis ein Menüpunkt ausgewählt wurde und dann den Kurzbezeichner für diesen Menüpunkt zurück ('char auswahl(int)') gibt.

### 6.5.2 Metaklassen

Doch ein Menü reichte allein natürlich noch nicht, um eine bessere Programmbedienbarkeit zu gewährleisten. So war es nötig, das bestehende System mit einem weiteren zu umgeben, welches für Abläufe höherer Ordnung eine besondere Umgebung bieten kann. Es war erwünscht, beliebig viele Graphen im System verwalten zu können und diesen gleichzeitig, falls sie Ergebnis von Vergleichen waren, die Vergleichsparameter zuzuordnen.

Außerdem sollte die Möglichkeit des Vergleichs einer Gesamtmenge von Graphen untereinander implementiert werden.

Alle eben genannten Anforderungen wurden darum in speziellen 'Metaklassen' implementiert, welche übergeordnete Abläufe verwalten können.

#### *6.5.2.1 Die Klasse 'metaclass' (metaclass.*)*

Diese Klasse enthält, wie in Programm 6-10 zu sehen ist, als wesentlichen Bestandteil eine Liste mit Pointern auf Graphstrukturen ('graph_struct').

```
class graph_struct
{
 public:
   graph_struct(void);
   graph  *Graph;
   char   *Name;
   char   *DateiName;
   wta_params *wtaParam;
   void getFileName(char*, char*);
   void free(void);
};

class metaclass : public sys_file_io
{
   ...
 protected:
   wta_netz *Netz;
   graph_struct *graphA, *graphB;
   menge_sort_p<graph_struct*> graph_liste;
   int maxGNameL;
 public:
   ...
};
```

*Programm 6-10 - Ausschnitt aus metaclass.h*

In diesen Strukturen wird nicht nur der Graph selbst verwaltet. Zusätzlich wird dessen Name im System und der Name der Datei, aus welcher der Graph gelesen wurde bzw. in welche er zuletzt geschrieben wurde vermerkt. Falls der Graph Ergebnis eines Vergleichs ist, werden in 'wtaParam' die Parameter des Netzes registriert, diese werden zum Beispiel beim Speichern des Graphen als Kommentar hinzugefügt.

Durch die Liste dieser Strukturen ist ein Verwalten von beliebig vielen Graphen im System möglich, Graphen können also fortwährend verglichen werden, ohne daß vorherige Ergebnisse verloren gehen.

Die Klasse enthält außerdem einen Zeiger auf ein WTA- Netz sowie alle Methoden, welche für das Arbeiten mit dem Netz selbst erforderlich sind. Diese Methoden entsprechen weitestgehend den Aufrufen, welche in 'main.cpp' durchgeführt werden, wenn ein Menüpunkt ausgewählt wurde.

#### *6.5.2.2 Die Klasse 'meta_wta' (metawta.*)*

Diese Klasse besteht nur aus Methoden und dient zur menügestützten Änderung der Netzparameter des Netzes in der Klasse 'metaclass'. Die Aufrufmethode ('void changeDef(void)') erstellt unter Zuhilfenahme der Klasse menu ein Menü und bietet Änderungen der Parameter an. Diese Einstellungen werden in der Klasse, auf die der Pointer 'Netz' in metaclass zeigt, sofort eingestellt.

#### *6.5.2.3 Die Klasse 'matrix' (matrix.*)*

Zum Verständnis des Aufbaus dieser Klasse ist es nötig, zuerst einige Worte über den Zweck von 'Matrixberechnungen' zu verlieren (siehe auch Abschnitt 5.5).

Wenn ein Vergleichsverfahren wie das implementierte WTA- Netz bewertet werden soll, ist es sinnvoll, für eine bestimmte Anzahl von Testgraphen sämtliche Ähnlichkeiten der Graphen untereinander zu bestimmen und in einer Matrix auszugeben. Wichtig als Ergebnis sind bei den jeweiligen Vergleichen nur die Gütemaße, nicht die Ergebnisgraphen.

Die Klasse Matrix enthält dafür drei Methoden - eine zum Bilden, eine zum Anzeigen und eine zum Speichern der Matrix.

| Dateiname | Inhalt |
|---|---|
| .wta_matrix.matrix | Informationen über Anzahl der Prozesse, Anzahl der Graphen und Anzahl der Berechnungen je Prozeß |
| .wta_matrix.graphen | enthält alle zu vergleichenden Graphnamen |
| .wta_matrix.netparams | Netzparameter |
| .wta_matrix.structur | Markierungsstruktur |
| .wta_matrix.daten_0 bis .wta_matrix.daten_999 | errechnete Gütewerte |

*Tabelle 6-17 - Matrixdateien*

Da eine solche Matrixberechnung sehr aufwendig sein kann (z.B. erfordern 188 Graphen über 17000 Vergleiche) war es wichtig, die aktuellen Berechnungsergebnisse sofort in Dateien zu speichern. Aus diesen Dateien wird am Berechnungsende die Matrix rekonstruiert (Tabelle 6-17).

Zusätzlich ergab sich dabei die Möglichkeit, die Vergleiche in Gruppen zu teilen und auf verschiedenen Rechnern rechnen zu lassen, um schneller zu Ergebnissen zu kommen. Aus diesem Grund sind einige 'ExpertInnenMethoden' vorgesehen. Der erste Problemteil kann berechnet werden, ein neuer Berechnungsteil begonnen oder ein alter fortgesetzt werden.

Die in Tabelle 6-17 genannten Dateien (natürlich außer den Gütewerten) werden dafür durch den ersten Berechnungsaufruf, bei welchem auch die Anzahl der Berechnungen festgelegt wird, erzeugt, um die Berechnungsumgebung für alle weiteren Berechnungsteile rekonstruieren zu können.

Außerdem enthält die Klasse noch eine weitere Methode, welche immer wieder neue Prozesse startet, bis keine weiteren Vergleiche offen sind. So kann die Berechnungsaufgabe in sehr viele kleine Berechnungsteile geteilt werden, und Rechner, welche ihren Berechnungsteil bereits abgeschlossen haben können beim nächsten fortfahren (siehe 8.2.3).

### 6.5.3 Programmstart main()

| Datei | Anhang |
|---|---|
| main.cpp | A XXXII |

*Tabelle 6-18 - Hauptprogramm*

Die in 'main.cpp' implementierte Funktion 'main()' ist der Startpunkt für den menügestützten Betrieb des WTA- Netzes. Durch diese Funktion (und einige ergänzende) wird ein Bedienmenü aufgebaut, welches bei Aufruf eines Menüpunktes zu Methoden der Metaklassen verzweigt.

Da natürlich alle derartigen Methoden nur auf einer Metaklasseninstanz arbeiten, wird diese als 'basis'- Instanz gebildet. Außerdem werden systemweite Instanzen (oder Zeiger auf diese) bereitgestellt.

## 6.6 Programmgenerierung

Für jede Art der Nutzung, unabhängig ob mit Programmierschnittstelle oder mit Bedienoberfläche, müssen natürlich erst alle Implementationsdateien compiliert werden.

```
> g++ -c *.cpp
```

Liegen dann alle benötigten Objektfiles vor, kann das Basissystem (ergänzt durch die Listenverwaltung und die Eingabeklassen), welches für jede Anwendungsart nötig ist, gelinkt werden.

```
> ld -r basis_liste.o basismerkmal.o cmp.o file.o general.o gmenge.o
graph.o graph_file.o hierarchie.o knoten.o kanten.o komp_graph.o m_gmenge.o
m_hierarchie.o m_symbol.o m_zahl.o menge.o menge_sort.o merkstrukt.o
multiset.o multiset_sort.o terminal.o wta_init.o wta_iterate.o wta_netz.o
wta_params.o -o BasisSys.o
```

Als Ergebnis des Aufrufs entsteht ein Objektfile, welches das gesamte Basissystem beinhaltet.

### 6.6.1 Nutzung der Schnittstelle

Soll nun die Programmierschnittstelle genutzt werden, muß das Basissystem noch um die Schnittstelle erweitert werden:

```
> ld -r BasisSys.o wta_work.o -o wta_work.o
```

Das nun vorliegende Objektfile 'wta_work.o' (welches an dieser Stelle das Objektfile von 'wta_work.cpp' überschreibt) kann nun in Zusammenhang mit 'wta_work.h' als komplettes Programm- Objektfile inklusive Schnittstellenbeschreibung betrachtet werden. Soll ein für die Nutzung der Schnittstelle geschriebenes Programm 'main2.cpp' genutzt werden, kann dies nach dessen Compilierung ('g++ -c main2.cpp') wie folgt geschehen:

```
> g++ main2.o wta_work.o -o wta
```

Das komplette Programm steht nun mit dem Aufruf 'wta' zur Verfügung.

### 6.6.2 Nutzung der Bedienoberfläche

Nachdem auch hierfür bereits BasisSystem.o erzeugt wurde, müssen nun alle Schnittstellenobjektfiles gelinkt werden:

```
> ld -r basis_liste_ext.o matrix.o menge_sort_ext.o menu.o metaclass.o
metawta.o multiset_ext.o -o IntAktSys.o
```

Nun können BasisSystem, Interaktive Systemoberfläche und Hauptprogramm ('main.o') zusammengelinkt werden:

```
> g++ BasisSys.o IntAktSys.o main.o -o wta
```

Als Programm wta steht nun das WTA- Netz mit Bedienoberfläche zur Verfügung.

# 7 Beschreibung der Eingabeformate

Wie schon bei der Vorstellung der Programmstruktur erwähnt (Abschnitt 5.), war für eine bedienfreundliche Nutzung des Systems die Entwicklung eigener Eingabeformate erforderlich.

Da alle einzulesenden Dateien durch eine Klasse (readfile) bzw. deren Unterklassen eingelesen werden (siehe auch 6.2.1.), sind in *allen* Dateien die folgenden Zeichen als Sonderzeichen zu betrachten, ganz gleich, ob diese nun Graphen oder Hierarchien oder geordnete Mengen beschreiben (Tabelle 7-1).

| Zeichen | Bedeutung |
|---|---|
| **Zeichen mit Wortendekennung:** | |
| ',' | Worttrennung |
| '(' | Parameteranfang |
| '/' | Merkmalstrennung bei Kantenmerkmalen |
| ':' | Knoten- bzw. Kantenbezeichnerende |
| cr = (int) 10 | Return = Zeilenende |
| **Zeichen ohne Wortendekennung:** | |
| ')' | Parameterende |
| '{' | Konzeptanfang |
| '}' | Konzeptende |
| '[' | Intervallanfang |
| ']' | Intervallende |
| '#' | Kommentaranfang bzw. -ende |

*Tabelle 7-1 - Sonderzeichen- Übersicht*

Kommentare können in allen einlesbaren Dateien vorkommen, diese müssen in '#' eingeschlossen sein. Oder anders gesagt, alle Zeichen innerhalb zweier '#' werden als Kommentar angesehen und darum ignoriert.

ACHTUNG: Die Knotenbezeichner 'Knoten:' und 'Kante:' (siehe 7.2.1) sind als Schlüsselwörter für die Markierungsbeschreibungen reserviert (siehe folgenden Abschnitt). Dies bedeutet, daß einerseits keinerlei Knoten den Name 'Knoten' bzw. 'Kante' haben dürfen ('KnotenA' ist jedoch erlaubt), andererseits kann die Markierungsbeschreibung somit auch am Anfang (!) einer Graphbeschreibungsdatei stehen. In diesem Fall muß die Gesamtbeschreibungsdatei zuerst als Markierungsbeschreibung und danach als Graphbeschreibung eingelesen werden.

## *7.1 Markierungsbeschreibungsdatei*

Bevor irgendwelche Graphen miteinander verglichen werden können, muß dem Programm mitgeteilt werden, welche Merkmale in welcher Reihenfolge den Graph beschreiben und wie diese verglichen werden sollen. Diese Aufgabe übernimmt die Markierungsbeschreibungsdatei, welche im folgenden beschrieben werden soll.

Die Markierungsbeschreibungsdatei muß die Schlüsselwörter 'Knoten:' und 'Kante:' in dieser Reihenfolge enthalten und kann auf diese Schlüsselwörter folgend beliebig viele Merkmalsbeschreibungen enthalten, welche durch Komma und/oder Return voneinander getrennt werden müssen.

Eine Merkmalsbeschreibung besteht aus einem Merkmalsbezeichner (siehe Tabelle 7-2) und optional darauf folgend einer Liste von Parametern in Klammern '(' bzw. ')'.

| Merkmal-bezeichner | Maximalzahl Parameter | Merkmalsausprägung |
|---|---|---|
| ganz_1 | 2 | ganzzahlig; statistischer Vergleich |
| ganz_2 | 4 | ganzzahlig; 'fuzzy'- Vergleich |
| reell_1 | 2 | reell; statistischer Vergleich |
| reell_2 | 4 | reell; 'fuzzy'- Vergleich |
| symbol | 1 | Symbole |
| begriff | 4 | Begriffe aus einer Hierarchie |
| gmenge_1 | 3 | Begriffe aus einer geordneten Menge; statistischer Vergleich |
| gmenge_2 | 5 | Begriffe aus einer geordneten Menge; 'fuzzy'- Vergleich |

*Tabelle 7-2 - implementierte Merkmalsarten*

Der erste Parameter entspricht dem Gewicht des Merkmals, mit welchem dieses in die Gesamtbewertung für den Knotenvergleich bzw. die Kantenvergleiche eingeht. Ist dieser größer als 1 oder wird er weggelassen, wird bei einem Einzelvergleichsergebnis ungleich 1 das Gesamtvergleichsergebnis auf 0 gesetzt, was bedeutet, daß für genau dieses Merkmal Identität verlangt wird.

| Parameter | Typ | Default | Bedeutung |
|---|---|---|---|
| 1. | float | >1 | Gewichtung des Merkmals, bei Wert größer 1 wird Identität der Merkmale gefordert |
| **Merkmal ganz_1:** | | | |
| 2. | float | 1 | Ähnlichkeitsabstand *l* für Objekt-Objekt- Vergleich |
| **Merkmal ganz_2:** (unterstützt Intervalle) | | | |
| 2. | float | 1 | Ähnlichkeitsabstand *l* |
| 3. | float | 0 | Abstand, bei dessen Unterschreitung benachbarte Elemente zu Intervallen zusammengefaßt werden (0 = keine Intervallbildung) |
| 4. | int | 5 | Anzahl der Stützstellen in *l* für die lineare Approximation der Integrale |
| **Merkmal reell_1:** | | | |
| Bedeutung und Art der Merkmale siehe ganz_1 | | | |
| **Merkmal reell_2:** (unterstützt Intervalle) | | | |
| Bedeutung und Art der Merkmale siehe ganz_2 | | | |
| **Merkmal symbol:** | | | |
| keine weiteren Parameter | | | |
| **Merkmal begriff:** | | | |
| 2. | char* | notwendig | Dateiname, aus der die Hierarchie gelesen werden soll |
| 3. | 'Gesamt' / 'Teil' | 'Gesamt' | "Gesamt" - Berücksichtigung der Höhe der Gesamthierarchie<br>"Teil" - Berücksichtigung der Resthierarchiehöhe |
| 4. | float | 0 | kleinster Wert, den der Vergleich annehmen kann, muß zwischen 0 und 1 liegen |
| **Merkmal gmenge_1:** | | | |
| 2. | char* | notwendig | Dateiname, aus der die geordnete Menge gelesen werden soll |
| 3. | siehe 2. Parameter bei ganz_1 | | |
| **Merkmal gmenge_2:** (unterstützt Intervalle) | | | |
| 2. | char* | notwendig | Dateiname, aus der die geordnete Menge gelesen werden soll |
| x. (2 < x < 6) | siehe (x-1). Parameter bei ganz_2 | | |

*Tabelle 7-3 - Merkmalsparameter*

Die genauen Verfahren, mit welchen die einzelnen Merkmalsausprägungen verglichen werden, wurden bereits in Abschnitt 2.4. beschrieben - dennoch soll hier die Reihenfolge der möglichen Parameter für die einzelnen Vergleichsverfahren aufgezeigt werden (siehe Tabelle 7-3).

Parameter können weggelassen werden, sofern sie nicht unbedingt für den Programmablauf nötig sind (und keine folgenden Parameter übergeben werden sollen oder müssen).

Knotenmerkmale sind generell ungerichtet (wohin auch?), Kantenmerkmale können jedoch auch gerichtet sein. Daher werden bei der Beschreibung der Merkmale für Kanten zunächst die ungerichteten Merkmale angegeben, nach dem Schlüsselzeichen '/' jedoch Merkmalsbeschreibungen für gerichtete Merkmale angegeben.

ACHTUNG: Während die Merkmalsausprägungen natürlich in jede Richtung verschieden sein können, müssen alle gerichteten Gesamtmerkmale identisch strukturiert sein, weshalb natürlich die Struktur nur für eine Richtung angegeben werden muß (und darf).

Zur Verdeutlichung nun einige mögliche Markierungsbeschreibungen:

### 7.1.1 Beispiel 1

```
Knoten:                  # SchluesselWort fuer KnotenMerkmale #
Kante:                   # SchluesselWort fuer KantenMerkmale #
```

*Programm 7-1 - einfachste Markierungsbeschreibung*
*( Kommentare in '#' können weggelassen werden )*

Die in Tabelle 7-1 angegebene Struktur hat keinerlei Merkmale, was bedeutet, daß jeder Knoten mit jedem anderen Knoten und jede Kante mit jeder anderen Kante matcht.

eine dazu passende Knotenbeschreibung:

```
A:
```

eine dazu passende Kantenbeschreibung:

```
A,B:
```

### 7.1.2 Beispiel 2

```
# Strukturbeschreibung #

Knoten  :                #Schluesselwort fuer KnotenBeschreibung   #
ganz_2(0.8,4,0,7)
begriff(0.2,testhier,Teil)
reell_1(0.3,3)

Kante:                   # Schluesselwort fuer KantenBeschreibung #
symbol(0.8,8)            # ungerichtetes Merkmal                  #
gmenge_1(0.5,testgmenge,4)
/                        # Trennzeichen                           #
ganz_1(0.7)              # gerichtetes Merkmal                    #
```

*Programm 7-2 - komplexere Markierungsbeschreibung*

eine dazu passende Knotenbeschreibung wäre:

```
A:{2,5,4},Linde,3   # KnotenName A                                  #
```

eine dazu passende Kantenbeschreibung wäre:

```
A,B:rot,{drei,vier}/1/2    # Kante zwischen A und B, das erste ungerichtete     #
                    # Merkmal hat die Auspraegung rot, das zweite unge-        #
                    # richtete Merkmal hat die Auspraegung {drei,vier},        #
                    # gerichtetes Merkmal von A nach B hat Auspraegung 1       #
                    # und das von B nach A hat Auspraegung 2                   #
```

## 7.2 Graphbeschreibungsdatei

Jeder Graph wird natürlich auch mit einer Datei beschrieben. Ein Graph besteht aus Knoten und Kanten, denen jeweils verschiedene Merkmale zugeordnet sind. Die Reihenfolge von Knoten- und Kantenbeschreibungen ist beliebig, da die Datei in zwei Durchläufen (erst die Knoten und dann die Kanten) eingelesen wird.

### 7.2.1 Knotenbeschreibung

Betrachten wir zuerst die eben (siehe 7.1.2) gegebene Knotenbeschreibung in Zusammenhang mit der dort gegebenen Markierungsbeschreibung.

```
A:{2,5,4},Linde,3   # KnotenName A                                    #
```

Hierbei handelt es sich um den Knoten mit dem Name A (Dieser wird gespeichert und z.B. für das Finden der Kanten gematcht), welchem verschiedene Merkmalsausprägungen zugeordnet sind. Gleich am Anfang ist die Ausprägung ein Konzept (siehe dazu auch 2.2.) daß heißt dem Merkmal ('ganz_2') sind mehrere einzelne ganze Zahlen zugeordnet.

Ein Konzept wird durch '{' eingeleitet und durch '}' abgeschlossen. Im obigen Beispiel werden also die drei ganzen Zahlen 2, 5 und 4 dem Merkmal 'ganz_2' zugeordnet.

Zusätzlich können bei einigen Merkmalsarten (z.B. 'ganz_2') auch Intervalle angegeben werden, welche durch '[' eingeleitet und durch ']' abgeschlossen werden. Intervalle müssen immer innerhalb von Konzepten auftreten. (Beispiele für Intervalle: {[1,5]},{1,2,3,[4,5],6,7,8})

Das zweite Merkmal ist von der Art 'begriff', ist also ein Element einer Begriffshierarchie. Dem Merkmal wird nun der Begriff 'Linde' zugeordnet. Falls das Wort Linde nicht in der speziellen, durch die Markierungsstruktur benannten Begriffshierarchie auftreten würde, würde das Programm eine Warnung ausgeben und beim Vergleich dieses Merkmals mit einem anderen mit einer Fehlermeldung abbrechen.

Beim dritten Merkmal des Knotens handelt es sich um 'reell_1', dessen Ausprägung mit 3 angegeben wurde.

### 7.2.2 Kantenbeschreibung

Nun betrachten wir die Kantenbeschreibung zur in Abschnitt 7.1.2 gegebenen Markierungsbeschreibung genauer:

```
A,B:rot,{drei,vier}/1/2
```

Am Anfang stehen die Knoten, zwischen denen die Kante verläuft. Dies wäre in unserem Beispiel nur gültig, wenn auch der Knoten B vorhanden wäre, da ansonsten kein Ziel der Kante bekannt wäre. Die Reihenfolge der Nennung der Knoten ist nicht egal, da innerhalb der Kantenbeschreibung auch gerichtete Merkmale auftreten können, vorerst schauen wir uns jedoch den ungerichteten Beschreibungsteil an.

Das erste ungerichtete Merkmal war von der Art 'symbol' (siehe Programm 7-2) was bedeutet, daß die eigentliche Zeichenfolge als Symbol gewertet und verglichen wird. Die Ausprägung der Kante A,B in diesem Merkmal ist 'rot'.

Das zweite ungerichtete Merkmal war 'gmenge_1', also müssen die Elemente im folgenden Konzept (drei,vier) natürlich in der Beschreibungsdatei der geordneten Menge vorhanden sein. Anderenfalls wird mit einer Fehlermeldung abgebrochen.

Der nach Abschluß des Konzepts folgende '/' zeigt an (vor allem bestätigt, festgelegt ist dies sowieso durch die Strukturbeschreibung), daß nun zu dem gerichteten Merkmal übergegangen

wird. Dies ist von der Art 'ganz_1' und hat in Richtung von A nach B die Ausprägung 1, in Richtung von B nach A die Ausprägung 2.

```
# KnotenBeschreibung #
A:{2,5,4},Linde,3
B:{[1,5]},Schwein,{1,3,4,4,5}
C:{1,4,6},Tier,{1,3,4}
D:1,Aster,{1,2,3}

# KantenBeschreibung #
A,B:rot,{drei,vier}/1/2
A,C:{rot,gruen,blau},zwei/1/{1,2,3}
C,B:rosa,acht/1/{1,2,3,3}
D,C:{lila,gruen},eins/1/{1,2}
A,D:{blau,rot},drei/{1,2,3}/{3,3,3}
B,D:{gruen,rot},drei/1/{1,2,2}
```

*Programm 7-3 - Graphbeschreibungsdatei*

ACHTUNG: Zuerst folgen bei der Beschreibung einer Kante immer *alle* ungerichteten Merkmalsausprägungen, danach (nach einem /') *alle* in 'Hin'- Richtung (im Beispiel von A nach B) und zum Schluß (nach dem nächsten '/') *alle* in Rückrichtung (von B nach A, siehe Programm 7-3).

## 7.3 Datei zur Beschreibung einer Begriffshierarchie

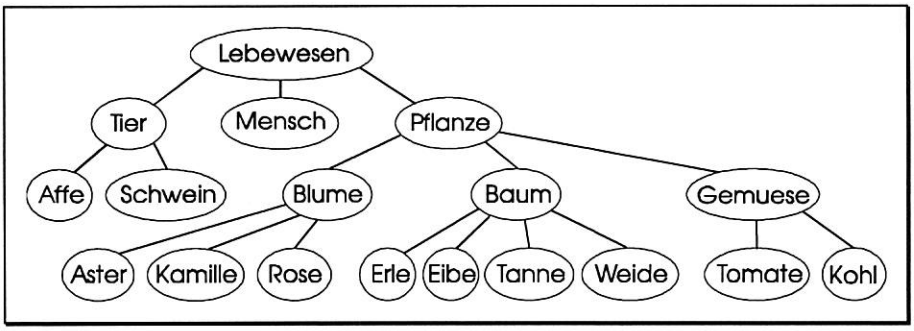



*Abbildung 7-1 - Beispielhierarchie*

Zur Eingabe einer Begriffshierarchie (Abbildung 7-1) muß eine Datei zur Beschreibung dieser entworfen werden. Dabei wird immer ein Oberbegriff aufgeführt, und nach einem ':' zur Trennung werden bis zum nächsten Oberbegriff bzw. Dateiende die direkt untergeordneten Begriffe aufgeführt (durch ',' oder Return getrennt, siehe Programm 7-4).

```
# HierarchieBeschreibung --------------------------#
# Oberbegriff: Unterbegriffe, durch Komma getrennt  #
# Alle Oberbegriffe, ausser Wurzel der Hierarchie
  muessen vorher bereits als Unterbegriffe aufge-
  zaehlt worden sein.                               #

Lebewesen:Mensch,Tier,Pflanze
Pflanze:Blume,Baum,Gemuese
Blume:Aster,Kamille,Rose
Baum:Erle,Eibe,Tanne,Weide
Gemuese:Tomate,Kohl
Tier:Affe,Schwein
```

*Programm 7-4 - Beschreibung einer Begriffshierarchie*

Wichtig ist, daß alle Oberbegriffe (außer die Wurzel der Hierarchie) vor dem Auftreten als Oberbegriff bereits als Unterbegriff eines anderen Oberbegriffs vorgekommen sein müssen. Nur so ist eine korrekte Konstruktion der Hierarchie möglich, in jedem anderen Fall wird eine Fehlermeldung ausgegeben und das Programm abgebrochen.

## 7.4 Datei zur Beschreibung einer geordneten Menge

```
eins,zwei,drei,vier,fuenf,sechs,sieben,acht,neun,zehn
```

*Programm 7-5 - Datei zur Beschreibung einer geordneten Menge*

Diese Datei ist sehr einfach strukturiert, sie enthält alle Elemente der geordneten Menge in aufsteigender Reihenfolge, jeweils durch Komma oder Return getrennt (Programm 7-5).

# 8 Bedienungsbeschreibung

## 8.1 Nutzung der Programmierschnittstelle

### 8.1.1 Beispielprogramm

In diesem kurzen Programm (Programm 8-1) wird die Nutzung der Schnittstelle gezeigt.

Wenden wir uns gleich dem Hauptprogramm (main()) zu. Die Klasse 'work' ist extern definiert und beinhaltet alle Methoden, welche zur Nutzung des WTA- Netzes nötig sind. Zuerst wird die Struktur der Graphen eingelesen - das sind Informationen darüber, durch welche Merkmale die Knoten und Kanten beschrieben werden. Dies muß unbedingt am Anfang erfolgen, da sonst die Merkmalsausprägungen der einzelnen Graphen nicht richtig zugeordnet werden können.

```
#include <iostream.h>
#include "wta_work.h"

main()
{
   if (!work.leseStruktur("bsp1.graph"))
     {
       cout << "Da ist was schiefgelaufen...\n";
       exit(1);
     }
   if (!work.leseGraphA("bsp1.graph"))
     {
       cout << "Da ist was schiefgelaufen...\n";
       exit(1);
     }
   if (!work.leseGraphB("bsp2.graph"))
     {
       cout << "Da ist was schiefgelaufen...\n";
       exit(1);
     }
   work.compare();
   work.speicherErgebnisGraph("erg.graph");
}
```

*Programm 8-1 - main.cpp*

Danach werden im Beispiel zwei verschiedene Graphen eingelesen, welche wiederum verglichen werden. Das Ergebnis dieses Vergleichs wird am Ende gespeichert und kann für neue Vergleiche genutzt werden.

Betrachten wir nun noch kurz die '#include'- Anweisungen am Anfang des Programms. Während das Einbinden von 'iostream.h' nur für die Ausgabeoperationen im Programm main() nötig ist, bindet 'wta_work.h' die eigentliche Schnittstelle ein.

Diese Schnittstelle besteht aus einer Klasse (wta_work, siehe Programm 8-2), welche Methoden enthält, mit denen alle wesentlichen Ein-/ Ausgabeoperationen durchgeführt werden können und andere, mit denen die Parameter des WTA- Netzes geändert werden können (Diese wurden im Beispielprogramm nicht genutzt).

```
#ifndef WTA_WORK_H
#define WTA_WORK_H


#ifndef WTA_PARAMS_ENUM
#define WTA_PARAMS_ENUM
enum s_art{lose,fixiert,reduziert};
enum s_was{knot,kant,beide};
enum anf_art{interaktiv,graphGroesse,fest,aehnlichkeit,listenwerte};
enum exI_art{ohne,prozentGraphGroesse,anzahlKanten};
enum kanSk_art{keine,kanten,kanten_u_knoten};
#endif

extern class wta_work work;

class wta_work
{
   class wta_netz *Netz;
   class graph *graphA, *graphB, *ErgGraph;
 public:
   wta_work(void);

// Ein-/ Ausgabemethoden
   bool leseStruktur(char*);
   bool leseGraphA(char*);
   bool leseGraphB(char*);
   void speicherErgebnisGraph(char*);

// Vergleichsmethode
   int compare(void);

// Kontrollmethoden
   void gibAusStruktur(void);
   void gibAusGraphA(void);
   void gibAusGraphB(void);
   void gibAusErgGraph(void);
   void gibAusParams(void);
   void gibAusAnfAktListe(void);

// Methoden zur Parameteraenderung
   void setAnfAktArt(enum anf_art);
   void setInit(float);
   void resetAnfAktListe(void);
   bool addAnfAkt(char*,char*,float);
   void setExtInpArt(enum exI_art);
   void setSkal(float);
   void setKanSkalArt(enum kanSk_art);
   void setSchrittArt(enum s_art);
   void setSchrittWas(enum s_was);
   void setAnzBerech(int);
   void setS(float);
   void setKnotenSchwelle(float);
   void addKnotenSchwelle(float);
   void setKantenSchwelle(float);
   void addKantenSchwelle(float);

   ~wta_work(void);
};

#endif
```

*Programm 8-2 - wta_work.h*

## 8.1.2 Ein-/ Ausgabemethoden

### *8.1.2.1 Methode bool work.leseStruktur(char* Dateiname)*

Bevor irgendwelche Graphen eingelesen werden können, müssen dem System Informationen über die Bedeutung und die Anzahl der Merkmale an Knoten und Kanten gegeben werden. Dies erfolgt einmal für alle Graphen gleicher Struktur durch eine Strukturbeschreibungsdatei (z.B. main.struct, zur Syntax dieser siehe 7.1). Natürlich ist nur ein Vergleich von Graphen mit

gleicher Merkmalstruktur sinnvoll und möglich, als Ergebnis entsteht ein Graph mit eben dieser Merkmalstruktur. Daher bedingt das Einlesen einer neuen Strukturbeschreibung auch das Löschen *aller* vorhandenen Graphen.

Als Parameter wird dieser Methode der Dateiname der Strukturbeschreibung übergeben.

Rückgabewert: true = 1 bei erfolgreicher Ausführung, sonst false = 0.

#### *8.1.2.2 Methoden* *bool work.leseGraphA(char* Dateiname)*<br>*bool work.leseGraphB(char* Dateiname)*

Diese Methoden lesen die Beschreibungen der Graphen ein. Zur Syntax der Graphbeschreibungsdatei sollte Abschnitt 7.2 beachtet werden.

Rückgabewert: true = 1 bei erfolgreicher Abarbeitung, sonst false = 0 oder Programmabbruch (exit(1)) bei schwerem Fehler.

#### *8.1.2.3 Methode* *work.speicherErgebnisGraph(char* Dateiname)*

Um das Vergleichsergebnis auch nutzen zu können, muß dieses abgespeichert werden. Die oben genannte Methode dient genau dazu und speichert den Graph als Graphbeschreibungsdatei ab. Diese kann wieder eingelesen werden (als GraphA oder GraphB), wenn die gleiche Strukturbeschreibung wie bei den zu vergleichenden Graphen das System vorher initialisiert hat.

ACHTUNG: Vor dem Abspeichern prüft diese Methode *nicht* den Zugriff auf die Datei. So kann es vorkommen, daß ungewollt ältere Dateien überschrieben werden oder ein Abspeichern wegen falschen Zugriffsrechten oder ungültigen Dateinamen unterbleibt. Es ist daher sinnvoll, mit eigenen Routinen den Dateizugriff zu prüfen.

### 8.1.3 Vergleichsmethode int work.compare(void)

Das ist natürlich das Herzstück des Programms, der eigentliche Vergleich findet hier statt. Das bedeutet auch, daß die Struktur und zwei Graphen vorher eingelesen sein müssen. Falls Nicht-Standard- Parameter genutzt werden sollen, müssen diese auch vor Aufruf dieser Methode geändert werden.

Rückgabewert: 0 bei Erreichen des Durchlaufendes (d.h. es wurde keine Lösung gefunden, Durchlaufende ist 10 * Anzahl der Knoten im Kompatibilitätsgraphen), sonst Anzahl der Durchläufe bis zu einem stabilen Zustand des Netzes.

### 8.1.4 Kontrollmethoden

Die folgenden Methoden sind nicht unbedingt für einen korrekten Programmablauf nötig. Jedoch ist es (besonders bei unverständlichen Programmabläufen) manchmal sinnvoll, die Übernahme der Dateien oder anderen Informationen ins System zu überprüfen. Alle Methoden in diesem Abschnitt geben bestimmte Informationen auf dem Terminal aus.

#### *8.1.4.1 Methode* *work.gibAusStruktur(void)*

Diese Methode gibt die Beschreibung der eingelesenen Merkmalstruktur in der Syntax der Markierungsbeschreibungsdatei (siehe Abschnitt 7.1) aus.

#### *8.1.4.2 Methoden* *work.gibAusGraphA(void)*
*work.gibAusGraphB(void)*
*work.gibAusErgGraph(void)*

Die Methoden dienen zur Ausgabe der im Namen der Methoden genannten Graphen. Diese werden in der Syntax der Graphbeschreibungsdatei (siehe Abschnitt 7.2) auf dem Terminal ausgegeben.

#### *8.1.4.3 Methode* *work.gibAusParams(void)*

Durch Aufruf dieser Methode wird auf dem Bildschirm die aktuelle Parametereinstellung in lesbarer Form ausgegeben (Ausgabeprotokoll 8-1).

```
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 1
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert
```

*Ausgabeprotokoll 8-1 - Ausgabe von work.gibAusParams() - Parameterinitialeinstellung*

#### *8.1.4.4 Methode* *work.gibAusAnfAktListe(void)*

Falls die Anfangsaktivierungen durch den Inhalt einer Liste bestimmt werden (work.setAnfAktArt(listenwerte) - siehe 8.1.5.1) können alle Anfangsaktivierungen dieser Liste mit dieser Methode auf dem Bildschirm ausgegeben werden. Ist eine andere Anfangsaktivierungsart eingestellt, wird die Liste geleert und diese Methode gibt daher nichts aus.

### 8.1.5 Methoden zur Parameteränderung

Zum korrekten Ablauf des Programms müssen die Parameter nicht unbedingt geändert werden, die Einstellung nach dem Programmstart ist im Ausgabeprotokoll 8-1 zu finden. Zur Verdeutlichung sind im folgenden die Parameter, deren Übergabe an entsprechende Methoden die Default- Einstellungen (wieder-) herstellt, unterstrichen.

ACHTUNG: Wenn eine Netzeinstellung besondere Parameter benötigt, welche vom Programm erfragt werden, heißt dies *nicht*, daß dies unbedingt im Moment der Änderung der Netzeinstellung erfolgen muß. Vielmehr werden einige Informationen erst beim Start des Vergleichs vom Programm anhand der aktuellen Parametereinstellung erfragt.

#### *8.1.5.1 Methode* *work.setAnfAktArt(enum anf_art Art)*

Änderung: Anfangsaktivierung der Netzknoten

*mögliche Parameter für Art (siehe enum anf_art in Programm 8-2- wta_work.h):*

interaktiv — für alle Knoten des Kompatibilitätsgraphs wird *jeweils* nach der Anfangsaktivierung gefragt

graphGroesse — Anfangsaktivierung wird anhand Anzahl der Knoten im Kompatibilitätsgraph berechnet

fest — alle Knoten werden mit dem gleichen Wert initialisiert, welcher durch 'work.setInit(Wert)' (8.1.5.2) festgelegt wird

aehnlichkeit Anfangsaktivierung wird wie bei 'graphGroesse' berechnet und zusätzlich für jeden Knoten mit dessen Ähnlichkeit gewichtet

listenwerte Den Neuronen wird eine Anfangsaktivierung zugeordnet, falls diese anhand der repräsentierten Knotenzuordnungen in einer Liste gefunden wird - ansonsten wird eine Anfangsaktivierung von 0 zugeordnet. (siehe 8.1.5.3 und 8.1.5.4)

#### *8.1.5.2 Methode work.setInit(float Wert)*

legt den Wert fest, welcher bei Anfangsaktivierungsart 'fest' für alle Knoten übernommen wird. (default 0.1).

#### *8.1.5.3 Methode work.addAnfAkt(char* A, char* B, float Wert)*

Wurde die Anfangsaktivierungsart auf 'listenwerte' eingestellt (work.setAnfAktArt(listenwerte) - siehe 8.1.5.1) werden alle Anfangsaktivierungen für die Neuronen des Netzes aus einer Liste entnommen - der Inhalt dieser Liste wird mit dieser Methode bestimmt. Dabei ist A ein Knotenbezeichner aus dem GraphA und B einer aus dem GraphB. Wert enthält die Anfangsaktivierung, welche dem Neuron, welches die Kombination aus A und B repräsentiert, zugeordnet wird. Die übergebenen Strings werden intern kopiert und können bei Bedarf sofort nach Aufruf dieser Methode gelöscht werden. (siehe auch 8.1.5.4)

Soll also dem Neuron, welches Knoten 'X' aus GraphA und Knoten 'Y' aus GraphB repräsentiert, eine Anfangsaktivierung von 0.75 zugeordnet werden, kann dies durch 'work.addAnfAkt("X","Y",0.75);' erfolgen.

#### *8.1.5.4 Methode work.resetAnfAktListe(void)*

Diese Methode löscht die Elemente der Liste von Neuronenaktivierungen (siehe 8.1.5.3)

#### *8.1.5.5 Methode work.setExtInpArt(enum exI_art Art)*

Änderung: externer Input der Netzknoten, ändert die Wichtigkeit der Knoten im Vergleich zu Kanten

*mögliche Parameter für Art (siehe enum exI_art in Programm 8-2- wta_work.h):*

ohne kein externer Input

prozentGraphGroesse ein Knoten ist genauso wichtig wie soviel Prozent der Kantenanzahl des Kompatibilitätsgraphes, wie mit 'work.setSkal(Wert)' eingestellt werden kann

anzahlKanten ein Knoten ist genauso wichtig wie die Anzahl Kanten, welche mit 'work.setSkal(Wert)' angegeben werden kann

#### *8.1.5.6 Methode work.setSkal(float Wert)*

legt den Wert fest, welcher für externen Input (bei Einstellung 'prozentGraphGroesse' bzw. 'anzahlKanten') als jeweiliger Parameter genutzt wird.

#### *8.1.5.7 Methode work.setKanSkalArt(enum kanSk_art Art)*

Änderung: Art der Kantenskalierung, also die Beeinflussung der an einem Knoten eingehenden Potentiale durch den Kanten zuzuordnende Werte.

*mögliche Parameter für Art (siehe enum kanSk_art in Programm 8-2- wta_work.h):*

keine alle eingehenden Potentiale werden nicht (d.h. mit 1) skaliert

kanten alle eingehenden Potentiale werden mit der Ähnlichkeit der jeweiligen Kante skaliert

kanten_u_knoten die eingehenden Potentiale werden mit der Ähnlichkeit der jeweiligen Kante und der Ähnlichkeit der Knoten, welche die Kante berühren, gewichtet

#### *8.1.5.8 Methode work.setSchrittArt(enum s_art Art)*

Änderung: Behandlung der Ergebnisknoten beim schrittweisen Matching. (Dies ist zum Beispiel durch Erhöhen der Berechnungszahl mit work.setAnzBerech(Wert) einstellbar). Dabei werden mehrere Berechnungen durchgeführt, wobei die Ähnlichkeitsschwelle, die übertroffen werden muß, um in den Kompatibilitätsgraph übernommen zu werden, jeweils gesenkt wird.

*mögliche Parameter für Art (siehe enum s_art in Programm 8-2- wta_work.h):*

lose die vorherige Lösung stellt die Anfangserregung für den neuen Berechnungsschritt dar, in jedem Schritt kommen neue Knoten und / oder Kanten hinzu

fixiert nach jedem Berechnungsschritt werden Knoten, welche nicht zur Lösung gehören, aus dem Graph entfernt, Knoten der Lösung werden auf ihrem aktuellen Wert (1) festgehalten. Dies dient dann als Anfangserregung für die nächsten Berechnungsschritte.

reduziert nach jedem Berechnungsschritt werden Knoten, welche nicht zur Lösung gehören, entfernt, danach werden mit neuer Ähnlichkeitsschwelle neue Knoten bzw. Kanten hinzugefügt und alle Knoten neu mit einer Anfangsaktivierung gleich initialisiert.

#### *8.1.5.9 Methode work.setSchrittWas(enum s_was Art)*

Änderung: Falls das schrittweise Matching durch die Angabe der maximalen Berechnungszahl eingestellt wurde, kann hiermit festgelegt werden, was schrittweise gematcht wird, also welche Ähnlichkeitsschwelle/ -n (Knoten u./o. Kanten) schrittweise gesenkt werden.

*mögliche Parameter für Art (siehe enum s_was in Programm 8-2- wta_work.h):*

knot Knotenschwelle wird schrittweise gesenkt

kant Kantenschwelle wird schrittweise gesenkt

beide beide Schwellen werden schrittweise gesenkt

#### *8.1.5.10 Methode work.setAnzBerech(int Wert)*

legt die Anzahl der Berechnungsschritte fest, die ausgeführt werden, um zur untersten Ähnlichkeitsschwelle zu gelangen. Dies ist eine Möglichkeit, schrittweises Matching einzustellen. Eine andere ist es, für eine Berechnung mehrere Schwellen anzugeben. (siehe 8.1.5.13 bzw. 8.1.5.15) Voreingestellt ist eine einzelne Berechnung (Berechnungsanzahl = 1).

#### *8.1.5.11 Methode work.setS(float Wert)*

stellt die Größe der Selbsthemmung $d$ der Neuronen des Netzes ein ($d = s * w$, als Wert wird $s$ übergeben). Die Voreinstellung ist 0, die Selbsthemmung ist damit abgeschaltet.

#### *8.1.5.12 Methode work.setKnotenSchwelle(float Wert)*

Diese Methode dient zur Festlegung der unteren Knotenschwelle, die nach Durchführung aller Berechnungen (siehe work.setAnzBerech(X)) erreicht wird. Bei der Bildung des Kompatibilitätsgraphes werden alle Knoten, welche als Vergleichsergebnis Werte über der aktuellen Knotenschwelle haben, in den Graphen aufgenommen.

Der voreingestellte Wert ist 1.

#### *8.1.5.13 Methode work.addKnotenSchwelle(float Wert)*

Durch diese Methode können zur bisher eingestellten unteren Knotenschwelle weitere Knotenschwellen hinzugefügt werden. Sind mehrere Knotenschwellen im System, so werden diese mit der größten beginnend zum schrittweisen Matching genutzt - die Berechnungsanzahl wird dabei ignoriert.

#### *8.1.5.14 Methode work.setKantenSchwelle(float Wert)*

Dies ist die adäquate Methode zu setKnotenSchwelle, jedoch für die Schwelle, über der Kanten zum Kompatibilitätsgraph hinzugefügt werden. Auch hier ist ein Wert von 1 voreingestellt.

#### *8.1.5.15 Methode work.addKantenSchwelle(float Wert)*

Dies ist die adäquate Methode zu addKnotenSchwelle - es können zusätzliche Kantenschwellen hinzugefügt werden.

## 8.2 Bedienung der Menüoberfläche

Für eine komfortable Bedienung des Programms wurde eine Bedienoberfläche implementiert, welche den Anwender bzw. die Anwenderin bei der Nutzung des Programms und der Verwaltung der Daten unterstützt. Die folgenden Abschnitte beschreiben kurz alle Menüpunkte - Anwendungsbeispiele sind in Abschnitt 9 zu finden. Dennoch erfolgt auch die Beschreibung der Menüpunkte sowie die Erläuterung bestimmter Programmausgaben in einer Reihenfolge, die einen Beispielablauf verdeutlicht, da nur durch den vorherigen Aufruf anderer Menüpunkte bestimmte Menüpunkte nutzbar werden.

### 8.2.1 Main Menu

```
Bitte waehlen Sie (Cursor up/down + Return oder Taste)

    ** Main Menu ************************
    *                                     *
    *       Markierungsbeschreibung       *
    *     s - einlesen                    *
    *     v - anzeigen                    *
    *     o - ausgeben                    *
    *       Graphbeschreibung             *
    *     g - einlesen                    *
    *     z - anzeigen                    *
    *     a - ausgeben                    *
    *     l - loeschen                    *
    *       KompatibilitaetsGraph         *
    *     k - bilden                      *
    *     u - anzeigen (lang)             *
    *     t - anzeigen (kurz)             *
    *                                     *
    *     n - Netz- Parameter aendern     *
    *     c - Graphen vergleichen         *
    *     x - Matrix berechnen            *
    *                                     *
    *     m - Makro einlesen              *
    *     e - ProgrammEnde                *
    *                                     *
    ***************************** V1.02 *

Bitte zuerst Markierungsbeschreibung einlesen

    ***********************************
```

*Ausgabeprotokoll 8-2 - main menu*

Das Hauptmenü des Programms bietet die grundlegenden Methoden für den Vergleich von Graphen an.

Zur Auswahl eines Menüpunktes kann dieser mit den Cursor- Tasten an- und mit Return ausgewählt werden. Außerdem besteht die Möglichkeit, durch Kurzbezeichner bestimmte Menüpunkte direkt auszuwählen.

Unterhalb des Menüs werden einige Systeminformationen gegeben, dies wird im folgenden als Statusinformation bezeichnet - diese Statusinformation gehört noch zum eigentlichen Menü. Wird ein bestimmter Menüpunkt ausgewählt, so erfolgen alle diesbezüglichen Ein- und Ausgaben unter dem Menü (und damit unterhalb der Statusinformationen).

Nach einem Programmstart wird das Hauptmenü mit den in Ausgabeprotokoll 8-2 gezeigten Systeminformationen ('Bitte zuerst Markierungsbeschreibung einlesen') angezeigt. Nach Abarbeitung jedes Menüpunktes wird wieder das aktuelle Menü angezeigt, lediglich bei bestimmten Menüpunkten wird zu einem anderen Menü gewechselt.

Zur Unterscheidung von Ein- und Ausgaben sind in den folgenden Ausgabeprotokollen alle Eingaben kursiv dargestellt - beim Programmablauf natürlich nicht. Beginnt in einem Ausgabeprotokoll die Zeile mit '#### Protokollunterbrechung', so steht dies für eine von mir vorgenommene Kürzung des Protokolls an dieser Stelle (zum Beispiel Ausgabeprotokoll 8-20).

#### *8.2.1.1 Markierungsbeschreibung*

##### 8.2.1.1.1 - einlesen

Mit diesem Menüpunkt kann eine 'Strukturbeschreibungsdatei' eingelesen werden.

Dafür muß der Name der Beschreibungsdatei angegeben werden, falls dieser mit '.struct' endet, kann dieser Teil weggelassen werden.

Ist eine Strukturbeschreibung eingelesen, ändert sich die Statusinformation und fordert die Anwenderin bzw. den Anwender auf, Graphen einzulesen.

##### 8.2.1.1.2 - anzeigen

Die Markierungsstruktur wird in der Syntax der Strukturbeschreibungsdatei (Abschnitt 7.1) auf dem Bildschirm ausgegeben (Ausgabeprotokoll 8-3)

```
GesamtStruktur aller Graphen:

Knoten:
symbol,
reell_2(0.5,4,4)
Kante:
reell_2(0.1,4,4)


                                                Taste druecken
```

*Ausgabeprotokoll 8-3 - Markierungsbeschreibung anzeigen*

##### 8.2.1.1.3 - ausgeben

Mit diesem Menüpunkt kann die Strukturbeschreibung in eine Datei ausgegeben werden - dafür muß natürlich der Name der Ausgabedatei angegeben werden.

### *8.2.1.2 Graphbeschreibung*

#### 8.2.1.2.1 - einlesen

Durch den Aufruf dieses Menüpunktes können beliebig viele Graphen, welche in der Syntax der 'Graphbeschreibungsdatei' (Abschnitt 7.2) vorliegen, in das System eingelesen werden.

Dafür muß natürlich der Name der Graphbeschreibungsdatei eingegeben werden - dieser wird bei Bedarf um '.graph' ergänzt. Der eingegebene Name dient im System als Bezeichner, durch welchen immer wieder auf genau diesen Graphen zugegriffen werden kann.

In der Statusinformation werden nun die im System vorhandenen Graphen aufgeführt - in Klammern wird die Anzahl der Knoten und Kanten des Graphen angegeben (Ausgabeprotokoll 8-4).

```
momentan im System verfuegbare Graphen      Name (KnotenZahl,KantenZahl):
bsp1 (3,2)
    VergleichsGraph A :  unbestimmt
    VergleichsGraph B :  unbestimmt

    ************************************
```

*Ausgabeprotokoll 8-4 - Statusinformation*

Zusätzlich werden die Vergleichsgraphen, aus denen der Kompatibilitätsgraph gebildet wurde, angezeigt. Wurde noch kein Kompatibilitätsgraph gebildet bzw. wurden noch keine Graphen verglichen, sind die Vergleichsgraphen mit 'unbestimmt' gekennzeichnet.

#### 8.2.1.2.2 - anzeigen

Ein Graph im System kann nach Eingabe seines Bezeichners auf dem Bildschirm angezeigt werden. Dafür wird die Syntax der 'Graphbeschreibungsdatei' (Abschnitt 7.2) genutzt. Für einen Beispielgraphen kann die Bildschirmanzeige wie in Ausgabeprotokoll 8-5 aussehen.

```
Anzeige eines Graphen...


 anzuzeigender Graph   : bsp1

1_1:Kreis,3
1_2:Block,0
1_3:Kreis,1
1_1,1_2:3
1_2,1_3:5


                                                  Taste druecken
```

*Ausgabeprotokoll 8-5 - Graphbeschreibung anzeigen*

#### 8.2.1.2.3 - ausgeben

Die Graphbeschreibung wird durch diesen Aufruf in eine Datei ausgegeben. Dafür muß zunächst der Graph ausgewählt werden, danach wird die Datei, in welche der Graph gespeichert werden soll, festgelegt.

Die Graphbeschreibung wird in dieser Datei durch die aktuelle Markierungsbeschreibung ergänzt - so kann diese Graphbeschreibungsdatei später gleichzeitig als Markierungsbeschreibung genutzt werden.

Ist der ausgegebene Graph das Ergebnis eines Vergleichs, wird die Graphbeschreibung zusätzlich um die Netzparameter in lesbarer Form als Kommentar ergänzt, da auf diese Weise

Ergebnisse von verschiedenen Vergleichen besser verwaltet werden können. (ähnlich Ausgabeprotokoll 8-12)

```
#*********************************************************
* WTA- Netz mit Berueckcksichtigung lokaler Aehnlichkeiten *
************************************************ V1.02 **
* autom. generierte Graphbeschreibung                    *
*********************************************************

*********************************************************
* Graph hat folgende Strukturbeschreibung:              #
Knoten:
symbol,
reell_2(0.5,4,4)
Kante:
reell_2(0.1,4,4)

#*******************************************************#
1_1:Kreis,3
1_2:Block,0
1_3:Kreis,1
1_1,1_2:3
1_2,1_3:5

#*******************************************************#
```

*Ausgabeprotokoll 8-6 - ausgegebene Graphdatei*

Im Ausgabeprotokoll 8-6 ist das Ergebnis der Ausgabe des Beispielgraphes von Ausgabeprotokoll 8-5 zu sehen.

#### 8.2.1.2.4 - löschen

Mit dieser Methode können Graphen anhand ihres Bezeichners wieder aus dem System entfernt werden.

### *8.2.1.3 Kompatibilitätsgraph*

#### 8.2.1.3.1 - bilden

Dieser Menüpunkt ist für die eigentliche Programmnutzung nicht unbedingt erforderlich. Hier wird ein Kompatibilitätsgraph gebildet, ohne daß ein anschließender Vergleich erfolgt. Sollen dagegen Graphen verglichen werden, ist dies nur mit dem Menüpunkt 'compare' möglich (siehe 8.2.1.5)

In bestimmten Situationen kann es jedoch wünschenswert sein, sich über den Kompatibilitätsgraph gesondert zu informieren - dies kann dann mit diesem Aufruf erfolgen.

Für die Bildung des Kompatibilitätsgraphen müssen die Bezeichner der zu vergleichenden Graphen (dies kann auch ein und derselbe Graph sein) und die Schwellen, ab denen Knoten bzw. Kanten in den Graphen aufgenommen werden, eingegeben werden (Ausgabeprotokoll 8-7).

```
neuer VergleichsGraph A   : bsp1
neuer VergleichsGraph B   : bsp2
KnotenSchwelle (0 <= Schwelle <= 1) : 0.5
KantenSchwelle (0 <= Schwelle <= 1) : 0.5
```

*Ausgabeprotokoll 8-7 - Kompatibilitätsgraph bilden*

Nach der Bildung des Kompatibilitätsgraphen (bzw. nach einem Vergleich mit 'compare') werden die Graphen, aus denen dieser gebildet wurde, in der Statusinformation angezeigt (Ausgabeprotokoll 8-8).

```
momentan im System verfuegbare Graphen      Name (KnotenZahl,KantenZahl):
bsp1 (3,2)    bsp2 (2,1)
    VergleichsGraph A : bsp1
    VergleichsGraph B : bsp2

    ************************************
```

*Ausgabeprotokoll 8-8 - Statusinformation*

#### 8.2.1.3.2 - anzeigen (lang)

Hierbei wird der Kompatibilitätsgraph komplett (mit Kanten) auf dem Bildschirm angezeigt.

```
**** {[1_1^2_1],0.75} = 0:
e{[1_2^2_2],1} i{[1_3^2_1]}
**** {[1_2^2_2],1} = 0:
e{[1_1^2_1],1} e{[1_3^2_1],1}
**** {[1_3^2_1],0.75} = 0:
e{[1_2^2_2],1} i{[1_1^2_1]}


                                                    Taste druecken
```

*Ausgabeprotokoll 8-9 - Kompatibilitätsgraph anzeigen*

Eine Ausgabe wie im Ausgabeprotokoll 8-9 gezeigt ist dabei wie folgt zu interpretieren. Die '****' kennzeichnen einen neuen Knoten im Kompatibilitätsgraph, der erste im Beispiel führt den Bezeichner '1_1^2_1', wurde also aus den Knoten '1_2' (Graph A) und '2_1' (Graph B) gebildet. Die Ähnlichkeit, welche bei dieser Zuordnung festgestellt wurden, wird nach dem Knotenbezeichner angegeben (im Beispiel 0.75).

Darauf folgend wird in derselben Zeile die Aktivität des Knotens angegeben - da kein kompletter Vergleich durchgeführt wurde, ist diese natürlich 0.

In der nun folgenden Zeile werden zuerst alle excitatorischen ('e{...}') und danach alle inhibitorischen ('i{...}') Kanten angezeigt, die diesen Knoten ('1_1^2_1') berühren. Bei excitatorischen Kanten wird das Ziel der Kante ('1_2^2_2') sowie deren Ähnlichkeit (1) angegeben, bei inhibitorischen Kanten natürlich nur das Ziel ('1_3^2_1'), da diesen keine Ähnlichkeit zugeordnet ist.

#### 8.2.1.3.3 - anzeigen (kurz)

Die Ausgabe für den oben beschriebenen Kompatibilitätsgraphen mit dieser Methode ist im Ausgabeprotokoll 8-10 zu sehen.

```
 1:
[1_2^2_2] [1_3^2_1]
 0:
[1_1^2_1]


                                                    Taste druecken
```

*Ausgabeprotokoll 8-10 - Kompatibilitätsgraph anzeigen*

Diesmal wurde - da auf diese Weise die Idee der Ausgabe deutlicher wird - der Kompatibilitätsgraph jedoch nicht mit 'Kompatibilitätsgraph bilden' erstellt, sondern die Graphen wurden mit 'compare' (8.2.1.5) verglichen. Der einzige Unterschied für den Kompatibilitätsgraphen ist an dieser Stelle, daß die Knoten nun verschiedene Aktivitäten besitzen können.

Angezeigt wird bei dieser Ausgabe zuerst ein Output- Potential und in der nächsten Zeile alle Knoten des Kompatibilitätsgraphen, welche diesen Output besitzen. Danach wird zum

nächstniedrigen Output- Potential übergegangen, und bis zum Output 0 werden alle Knoten ausgegeben.

#### *8.2.1.4 Netz- Parameter ändern*

Dieser Menüpunkt verzweigt zum Untermenü für die Veränderung der Netz- Parameter, siehe 8.2.2.

#### *8.2.1.5 Graphen vergleichen*

```
 neuer VergleichsGraph A   : bsp1
 neuer VergleichsGraph B   : bsp2
initialisiere Graph mit Schwellen 0.5, 0.5
Zeit fuer KompGraphBildung : 0 sec.
3 Knoten und 3 Kanten im Graph
Zeit fuer Iteration : 0.01 sec.
DurchlaufEnde erreicht
2 Knoten in der Loesung
Gueten  : 0.333333 0.583333 0.25



                                                Taste druecken
```

*Ausgabeprotokoll 8-11 - Graphen vergleichen*

Die eigentliche Aufgabe des Programms kann mit diesem Menüpunkt erfolgen - Graphen können verglichen werden (Ausgabeprotokoll 8-11).

```
momentan im System verfuegbare Graphen      Name (KnotenZahl,KantenZahl):
bsp1 (3,2)          bsp1^bsp2 (2,1)     bsp2 (2,1)
    VergleichsGraph A : bsp1
    VergleichsGraph B : bsp2

    ************************************

Anzeige eines Graphen...


 anzuzeigender Graph   : bsp1^bsp2

Graph wurde mit folgenden Parametern errechnet:
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0.5
   untere AehnlichkeitsSchwelle fuer Kanten = 0.5

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.333333 0.583333 0.25

1_2^2_2:Block,0
1_3^2_1:Kreis,{[1,2]}
1_3^2_1,1_2^2_2:{[4,5]}


                                                Taste druecken
```

*Ausgabeprotokoll 8-12 - Anzeigen des Ergebnisgraphen*

Dafür müssen nach dem Aufruf des Menüpunktes lediglich die Bezeichner der zu vergleichenden Graphen eingegeben werden - danach werden diese anhand der eingestellten Netzparameter verglichen.

Der neu gebildete Ergebnisgraph erhält als Bezeichner den Graphbezeichner des ersten Graphen, gefolgt von einem '^' und danach dem Bezeichner des zweiten Graphen und wird zum System hinzugefügt.

Alle Knotenbezeichner dieses Ergebnisgraphen werden natürlich auch auf diese Weise gebildet. Ausgabeprotokoll 8-12 zeigt die Statuszeile nach diesem Vergleich und die Änderung der Graphanzeige, da nun auch die Vergleichsparameter mit angezeigt werden.

#### *8.2.1.6 Matrix berechnen*

Für die Matrixberechnung (siehe Abschnitt 5.5) verzweigt dieser Menüpunkt zu einem besonderen Menü, siehe 8.2.3.

#### *8.2.1.7 Makro einlesen*

Dieser Punkt unterstützt die Abarbeitung von Makros - dies sind Dateien, welche genau die sonst am Terminal eingegebenen Zeichen enthalten und eine interaktive Bedienung des Programms simulieren können. Nach der Eingabe des Dateinamens wird sofort der Eingabestrom aus der Datei gelesen.

Eine einfache Makrodatei ist im Programm 8-3 zu sehen.

```
gbsp1
gbsp2
```

*Programm 8-3 - Beispielmakro*

Mit dieser Datei werden nacheinander zwei Graphen, zuerst 'bsp1' und danach 'bsp2', eingelesen. Dieses Makro zeigt auch schon den Nachteil dieser einfachen Makro- Methode. Falls ein Fehler auftritt (zum Beispiel einer der beiden Graphen bereits im System vorhanden ist - was eine besondere Abfrage zur Folge hat) wird dies im Gesamtablauf nicht berücksichtigt (Ausgabeprotokoll 8-13).

```
Einlesen eines neuen Graphen
    Name der GraphDatei (*.graph)  :  ACHTUNG: Graph 'bsp1' bereits vorhanden...

 A - Abbruch       L - vorherigen Graph loeschen
 ---> ungueltige Eingabe 'g'
 A - Abbruch       L - vorherigen Graph loeschen
 ---> ungueltige Eingabe 'b'
 A - Abbruch       L - vorherigen Graph loeschen
 ---> ungueltige Eingabe 's'
 A - Abbruch       L - vorherigen Graph loeschen
 ---> ungueltige Eingabe 'p'
 A - Abbruch       L - vorherigen Graph loeschen
 ---> ungueltige Eingabe '2'
 A - Abbruch       L - vorherigen Graph loeschen
 ---> ungueltige Eingabe '
'
 A - Abbruch       L - vorherigen Graph loeschen
```

*Ausgabeprotokoll 8-13 - Makrofehler*

#### *8.2.1.8 Programmende*

Das Programm endet. Falls Graphen im System sind, erfolgt vorher eine Sicherheitsabfrage.

### 8.2.2 Netz- Parameter

Dieses Menü (Ausgabeprotokoll 8-14) dient zur Einstellung der Parameter des WTA- Netzes und teilt sich grob in vier Teile. Zuerst können Parameterbeschreibungen eingelesen und ausgegeben werden. Danach folgen einige Punkte, die für die verschiedenen Arten der Ähnlichkeitsabbildung und deren Auswahl wiederum Menüs anbieten. Weiter folgen Einstellmöglichkeiten für häufig zu ändernde Netzparameter sowie eine Option, zum Hauptmenü zurückzukehren.

```
Bitte waehlen Sie (Cursor up/down + Return oder Taste)

    ** Netz- Parameter ****************************
    *                                              *
    *       Parameter                              *
    *     l - aus Datei lesen                      *
    *     d - in Datei schreiben                   *
    *                                              *
    *     s - schrittweises Matching / Schwellen   *
    *     a - AnfangsAktivierung                   *
    *     e - externer Input                       *
    *     k - KantenSkalierung                     *
    *                                              *
    *     u - set KantenSchwelle                   *
    *     w - set KnotenSchwelle                   *
    *     i - Selbsthemmung                        *
    *                                              *
    *     m - Main Menu                            *
    *                                              *
    ************************************** V1.02 *

keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 1
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

    ***********************************************
```

*Ausgabeprotokoll 8-14 - Netz- Parameter Menü*

Der Bereich der Statusinformation enthält alle Einstellungen des Netzes in lesbarer, verständlicher Form.

#### *8.2.2.1 Parameter*

##### 8.2.2.1.1 - aus Datei lesen

Hiermit können alle Parametereinstellungen aus einer mit dem folgenden Menüpunkt erzeugten Datei gelesen werden. Dafür ist natürlich die Eingabe des Dateinamens erforderlich.

##### 8.2.2.1.2 - in Datei schreiben

Wie bereits erwähnt, können mit diesem Aufruf sämtliche Parametereinstellungen in eine Datei geschrieben werden und stehen damit für eine spätere Rekonstruktion der Netzeinstellungen zur Verfügung.

### *8.2.2.2 schrittweises Matching*

Die verschiedenen im Ausgabeprotokoll 8-15 angegebenen Menüpunkte unterstützen die Einstellung des Netzes für schrittweises Matching. Bei der Festlegung der für diese Ähnlichkeitsabbildung genutzten Schwellen gibt es zwei verschiedene Wege.

Zum einen können die Schwellen explizit festgelegt werden. Die Menüpunkte 'set...' löschen dabei alle bisherigen Schwellen und ersetzen diese durch eine neue - die Punkte 'add...' fügen zu den bestehenden Schwellen eine neue hinzu. Bei der Abarbeitung wird dann zuerst die oberste Knotenschwelle genutzt und mit allen Kantenschwellen (mit der größten beginnend) kombiniert. Dies wird mit allen vorhandenen Knotenschwellen bis zur untersten fortgesetzt.

```
Bitte waehlen Sie (Cursor up/down + Return oder Taste)

   ** schrittweises Matching *********************
   *                                               *
   *      Festlegung der Schwellen                 *
   *    u - set KantenSchwelle                     *
   *    v - add KantenSchwelle                     *
   *    w - set KnotenSchwelle                     *
   *    x - add KnotenSchwelle                     *
   *      Festlegung der max. Berechnungszahl      *
   *    o - Knoten schrittweise matchen            *
   *    a - Kanten schrittweise matchen            *
   *    b - beide schrittweise matchen             *
   *    z - maximale Zahl der Berechnungen         *
   *      Ergebnisbehandlung                       *
   *    l - lose                                   *
   *    f - fixiert                                *
   *    r - reduziert                              *
   *                                               *
   *    n - Netz-Parameter Menu                    *
   *    m - Main Menu                              *
   *                                               *
   ************************************* V1.02 *
```

*Ausgabeprotokoll 8-15 - Menü für schrittweises Matching*

Eine andere Vorgehensweise ist die Bestimmung der maximalen Berechnungsanzahl. Hierbei muß festgelegt werden, ob nur die Knotenschwellen, nur die Kantenschwellen oder beide gleichzeitig gesenkt werden sollen.

Die Schwellenwerte erhalten jeweils nur einen Wert. Bei einem Vergleich wird nun für jede schrittweise zu ändernde Schwelle die Differenz zwischen 1 (maximale Ähnlichkeit) und der entsprechenden Schwelle durch die Berechnungsanzahl geteilt. Damit wird die Schrittweite für die Verringerung der Schwelle festgelegt. Schwellen, welche keine Änderung im Netz vornehmen, werden bei der Berechnung jedoch übergangen.

Durch beide oben erläuterten Verfahren kann also nun festgelegt werden, welche Schwellen zur Berechnung genutzt werden.

Die Ergebnisbehandlung entsprechend Abschnitt 3.3 kann durch weitere 3 Menüpunkte festgelegt werden.

Zusätzlich sind noch zwei Menüpunkte zum Rücksprung ins Netz- oder Hauptmenü vorhanden.

### *8.2.2.3 AnfangsAktivierung*

Mit diesem Menüpunkt können verschiedene Arten der Anfangsaktivierung menügestützt ausgewählt werden (Ausgabeprotokoll 8-16).

```
Bitte waehlen Sie (Cursor up/down + Return oder Taste)

   ** AnfangsAktivierung ***********
   *                               *
   *    a - Eingabe fuer alle      *
   *    e - Eingabe einzeln        *
   *    g - anhand Graphgroesse    *
   *    s - anhand Aehnlichkeit    *
   *                               *
   *    n - Netz-Parameter Menu    *
   *    m - Main Menu              *
   *                               *
   ************************* V1.02 *
```

*Ausgabeprotokoll 8-16 - Menü Anfangsaktivierung*

So kann durch den Menüpunkt 'Eingabe für alle' eine feste Anfangsaktivierung, welche alle Neuronen des Netzes erhalten, festgelegt werden. Eine weitere Möglichkeit der Anfangsaktivierung kann mit dem Punkt 'Eingabe einzeln' eingestellt werden. Dabei wird für alle Neuronen, welche in das Netz aufgenommen werden, einzeln nach deren Aktivierung gefragt.

Mit 'anhand Graphgroesse' kann zur Standardeinstellung, der Ermittlung der Anfangsaktivierung der Neuronen aus der Graph- bzw. Netzgröße, zurückgekehrt werden. Hier wird die Anfangserregung als Quotient aus der kleineren Anzahl von Knoten in beiden Vergleichsgraphen und der Anzahl der Neuronen im Netz gebildet (Formel 3-5).

Die Ähnlichkeitsabbildung auf die Anfangsaktivierung nach Abschnitt 3.2 kann mit dem Menüpunkt 'anhand Aehnlichkeit' eingestellt werden, wobei die Anfangsaktivierung anhand der Graphgröße errechnet und dann zusätzlich mit der Ähnlichkeit des entsprechenden Neurons gewichtet wird (Formel 3-6).

Außerdem sind natürlich auch hier Rücksprungmöglichkeiten zu den anderen Menüs vorgesehen.

#### *8.2.2.4 externer Input*

```
Bitte waehlen Sie (Cursor up/down + Return oder Taste)

   ** externer Input *********************
   *                                      *
   *    k - keiner                        *
   *      Aehnlichkeit gewichtet mit      *
   *    x - x Prozent der Graphgroesse    *
   *    a - absolutem Wert                *
   *                                      *
   *    n - Netz- Parameter Menu          *
   *    m - Main Menu                     *
   *                                      *
   ****************************** V1.02 *
```

*Ausgabeprotokoll 8-17 - Menü externer Input*

Dieser Menüpunkt verzweigt ebenfalls zu einem weiteren Menü (Ausgabeprotokoll 8-17), mit welchem nun die verschiedenen Arten eines externen Inputs eingestellt werden können.

Die Standardeinstellung ist 'keiner'- also die Neuronen besitzen dabei keinen externen Input.

Soll dagegen die Ähnlichkeit der Knoten auf den externen Input, wie in Abschnitt 3.4. beschrieben, abgebildet werden, so muß der Skalierungsfaktor *skal* nach Formel 3-9, Abschnitt 3.4. festgelegt werden.

Wird der Menüpunkt 'x Prozent der Graphgroesse' genutzt, kann ein Wert *data* eingegeben werden und die Skalierung der Ähnlichkeit für den externen Input erfolgt mit folgender Formel:

*skal=Kantenanzahl im Netz * data / 100*

Wird als Skalierung ein 'absoluter Wert' festgelegt, so kann *skal* (Formel 3-9) direkt eingegeben werden.

Wie in den beiden vorangegangenen Menüs gibt es auch hier abschließend die Möglichkeit, zu den übergeordneten Menüs zurückzukehren.

#### *8.2.2.5 KantenSkalierung*

```
Bitte waehlen Sie (Cursor up/down + Return oder Taste)

   ** KantenSkalierung **************
   *                               *
   *    k - keine                  *
   *    a - KantenAehnlichkeit     *
   *    o - Kanten+KnotenAehnl.    *
   *                               *
   *    n - Netz- Parameter Menu   *
   *    m - Main Menu              *
   *                               *
   ************************* V1.02 *
```

*Ausgabeprotokoll 8-18 - Menü Kantenskalierung*

Dieser Menüpunkt öffnet ein neues Menü (Ausgabeprotokoll 8-18) für die Festlegung der Art der Kantenskalierung, welche in der Voreinstellung nicht aktiviert ist. Die Kantenskalierung kann mit dem Punkt 'keine' jederzeit wieder abgeschaltet werden.

Sollen jedoch Ähnlichkeiten für die Skalierung der Kanten genutzt werden, gibt es die Möglichkeit nach Abschnitt 3.5.1.- die Skalierung anhand der 'Knotenaehnlichkeit' oder die im Abschnitt 3.5.2. beschriebene Skalierung anhand der Ähnlichkeit von Knoten und Kanten ('Kanten+KnotenAehnl.').

Zusätzlich hat auch dieses Menü wieder zwei Punkte, mit denen zu übergeordneten Menüs gesprungen werden kann.

#### *8.2.2.6 setKnotenSchwelle*

Dieser Aufruf löscht *alle* vorherigen Knotenschwellen und ersetzt diese durch eine neu einzugebende Schwelle.

#### *8.2.2.7 setKantenSchwelle*

Dieser Menüpunkt entspricht dem vorhergehenden, nur daß die Schwelle für Kanten geändert wird.

#### *8.2.2.8 Selbsthemmung*

Mit diesem Aufruf kann der neue Wert für die Selbsthemmung in Abhängigkeit von *w* eingestellt werden.

#### *8.2.2.9 main menu*

Rücksprung zum Hauptmenü.

### 8.2.3 Matrix Berechnung

```
Bitte waehlen Sie (Cursor up/down + Return oder Taste)

    ** Matrix Berechnung ***************
    *                                    *
    *       Matrix                       *
    *     b - bilden                     *
    *     a - anzeigen                   *
    *     s - speichern                  *
    *                                    *
    *       erweiterte Modi              *
    *     v - verteilt starten           *
    *     p - neuen Prozess              *
    *     d - Daten vervollstaendigen    *
    *     i - Status-Informationen       *
    *                                    *
    *       fortlaufende Berechnung      *
    *     1 - verteilt starten           *
    *     2 - neue Prozesse              *
    *                                    *
    *     m - Main Menu                  *
    *                                    *
    ************************** V1.02 *

Datei '.wta_matrix.matrix' nicht lesbar

    ************************************
```

*Ausgabeprotokoll 8-19 -Matrixberechnungsmenü*

Dieses Menü (Ausgabeprotokoll 8-19) dient zur Berechnung von Matrixdaten (siehe auch Abschnitt 5.5)

#### *8.2.3.1 Matrix*

Für eine einfache Berechnung einer Matrix können die ersten drei Menüpunkte genutzt werden.

##### 8.2.3.1.1 - bilden

Für die Bildung einer Matrix muß eine Datei genutzt werden, welche die Dateinamen aller zu vergleichenden Graphen, durch Return getrennt, enthält (dabei werden aus Kompatibilitätsgründen alle 'x' am Ende des Graphbezeichners ignoriert). Die Graphen werden eingelesen und die Berechnung wird gestartet - dabei werden die Resultate in einer Datei gespeichert, um nach einem Systemreset weiterhin alle bisher errechneten Ergebnisse nutzen zu können (Ausgabeprotokoll 8-20, zum Beispiel durch den Menüpunkt 'vervollstaendigen', Abschnitt 8.2.3.2.3).

```
Bilden einer Guete -Matrix :
  Name der Datei mit der Liste von Graphen  : all42

lese Liste mit Graphnamen aus Datei 'all42'
schreibe Datei '.wta_matrix.graphen'
lese Graphen: f3 f4 d190 e18

schreibe Datei '.wta_matrix.structur'
schreibe Datei '.wta_matrix.netparams'
schreibe Datei '.wta_matrix.matrix'
schreibe Datei '.wta_matrix.daten_0' und starte 1. Berechnung

vergleiche Graphen f3 und f4
initialisiere Graph mit Schwellen 1, 1
Zeit fuer KompGraphBildung : 0.09 sec.
66 Knoten und 204 Kanten im Graph
Zeit fuer Iteration : 0.45 sec.
stabile Loesung nach 84 Berechnungen (-17)
18 Knoten in der Loesung
Gueten  : 0.0566667 0.72 0.0566667

##### Protokollunterbrechnug, es folgen Berechnungen

6 Berechnungen erfolgt (6stabile, 0 unstabile Loesungen)
DurchschnittsAnzahl der Berechnungen = 58.1667
GesamtZeit fuer KompGraphBildung: 0.22 sec.
GesamtZeit fuer Iterationen     : 1.32 sec.


                                                Taste druecken
```

*Ausgabeprotokoll 8-20 - Matrix berechnen*

Bemerkbar ist nach einer Berechnung auch die Änderung der Statusinformation, welche nun Berechnungsdateien findet und diese prüft (Ausgabeprotokoll 8-21).

```
aktueller MatrixBerechnungsStatus:
prozess_anz = 1; berechn_anz = 6; graph_anz = 4
Graphanzahl in Datei '.wta_matrix.graphen' korrekt
pruefe Datei '.wta_matrix.netparams'  - ok.

    ************************************
```

*Ausgabeprotokoll 8-21 - Statusinformation*

#### 8.2.3.1.2 - anzeigen

```
lese MatrixSystemDaten aus Datei '.wta_matrix.matrix'
prozess_anz = 1; berechn_anz = 6; graph_anz = 4
lese Daten aus '.wta_matrix.daten_0' - lese - ok.
```

| | Graph 2 | Graph 3 | Graph 4 | |
|---|---|---|---|---|
| Güte 1 | 0.05667 | 0.02632 | 0 | |
| Güte 2 | 0.72 | 0.35 | 0.05 | Graph 1 |
| Güte 3 | 0.05667 | 0.02632 | 0 | |
| Güte 1 | | 0.01667 | 0.006667 | |
| Güte 2 | | 0.28 | 0.12 | Graph 2 |
| Güte 3 | | 0.01667 | 0.006667 | |
| Güte 1 | | | 0.05882 | |
| Güte 2 | | | 0.5556 | Graph 3 |
| Güte 3 | | | 0.05882 | |

```
Die Daten wurden ausgegeben.
Falls Sie die Daten nicht weiter nutzen wollen, sollten sie die
Dateien mit folgenden Muster im aktuellen Verzeichnis loeschen:
rm .wta_matrix*
Wenn Sie dies versaeumen, kann es Probleme mit weiteren Matrix-
berechnungen geben

                                                Taste druecken
```

*Abbildung 8-1 - modifiziertes Ausgabeprotokoll Matrixanzeige*

Die Matrix kann auf dem Bildschirm angezeigt werden, was natürlich nur sinnvoll ist, wenn nicht zu viele Graphen verglichen wurden. Im Ausgabeprotokoll in Abbildung 8-1 sind noch einige Informationen zur Ausgabe hinzugefügt, um die Struktur der Matrix zu verdeutlichen. Es ist ersichtlich, daß die rechte obere 'Hälfte' der Vergleichsmatrix aller Graphen dargestellt wird, und daß innerhalb jedes Blockes die drei Gütewerte untereinander aufgeführt sind.

#### 8.2.3.1.3 - speichern

Die Matrix wird in eine Datei gespeichert, der Name dieser muß natürlich von der Anwenderin bzw. dem Anwender eingegeben werden.

Die beiden eben beschriebenen Menüpunkte setzen voraus, daß alle Berechnungsdaten in den dafür vorgesehenen Dateien gefunden werden ('.wta_matrix.daten_*').

Zusätzlich zu den bereits beschriebenen Matrixbildungsverfahren wurden noch weitere Methoden implementiert, mit denen spezielle Vorgänge unterstützt werden.

### *8.2.3.2 erweiterte Modi*

#### 8.2.3.2.1 verteilt starten

Um den Rechenaufwand auf mehrere Rechner zu verteilen, wurde dieser Menüpunkt vorgesehen. Hierbei wird jedoch nur ein Prozeß auf einem Rechner gestartet, welcher für alle weiteren Prozesse die Referenzdateien (Tabelle 6-17) mit allen Informationen erzeugt- diese weiteren Prozesse können mit dem Punkt 'neuer Prozeß' gestartet werden und setzen ihre Berechnung an einem neuen Berechnungsteil mit genau denselben Einstellungen wie der 'Start'prozeß fort.

```
Bilden einer Guete -Matrix :
  Name der Datei mit der Liste von Graphen  : all42

lese alle in 'all42' gefundenen Graphen...
lese Graphenliste aus Datei 'all42'
schreibe Datei '.wta_matrix.graphen'



Fuer die Bildung der Matrix sind 6 Berechnungen noetig
In wieviel Prozessen sollen die Berechnungen laufen (1-6)   2
Dies bedeutet ca. 3 Berechnungen je Prozess, ist das in Ordnung ??
    J - ja,  N - nein,  A - Abbruch   j

schreibe Datei '.wta_matrix.structur'
schreibe Datei '.wta_matrix.netparams'
schreibe Datei '.wta_matrix.matrix'
schreibe Datei '.wta_matrix.daten_0' und starte 1. Berechnung

vergleiche Graphen f3 und f4
initialisiere Graph mit Schwellen 1, 1

##### Protokollunterbrechung, es folgen Berechnungen

stabile Loesung nach 3 Berechnungen (0)
1 Knoten in der Loesung
Gueten  : 0 0.05 0

3 Berechnungen erfolgt (3stabile, 0 unstabile Loesungen)
DurchschnittsAnzahl der Berechnungen = 52.3333
GesamtZeit fuer KompGraphBildung: 0.12 sec.
GesamtZeit fuer Iterationen     : 0.63 sec.


                                                Taste druecken
```

*Ausgabeprotokoll 8-22 - Matrixberechnung verteilt starten*

Der Menüpunkt 'verteilt starten' startet also eine verteilte Berechnung, was bedeutet, daß alle Graphen eingelesen werden und der Nutzer bzw. die Nutzerin danach bestimmen kann, in wieviel Prozesse die Gesamtaufgabe geteilt werden soll. Entsprechend diesen Informationen wird dann die Datei '.wta_matrix.matrix' geschrieben - spätere Prozesse stellen dies fest und erfahren so allgemeine Festlegungen.

Danach werden alle Berechnungen ausgeführt, welche dem ersten Prozeß zugeordnet sind (Ausgabeprotokoll 8-22).

#### 8.2.3.2.2 neuer Prozess

Sind durch den vorherigen Menüpunkt die Bedingungen für die verteilte Berechnung festgelegt, kann ein neuer Berechnungsteil durch einen erneuten Programmaufruf ('wta', eventuell auf einem anderen Rechner) und Auswahl dieses Menüpunktes gestartet werden. Dabei werden alle Matrixinformationen sowie die Liste der Graphnamen, die Markierungsstruktur sowie die Netzparameter aus den Dateien, welche durch 'verteilt starten' geschrieben wurden, eingelesen. Danach wird die erste noch nicht berechnete Datei für die Ergebnisse berechnet (Ausgabeprotokoll 8-23).

```
lese MatrixSystemDaten aus Datei '.wta_matrix.matrix'
prozess_anz = 2; berechn_anz = 3; graph_anz = 4
suche naechste noch fehlende Datei

pruefe '.wta_matrix.daten_0' - existiert
pruefe '.wta_matrix.daten_1' - nicht gefunden
lese NetzParameter aus Datei '.wta_matrix.netparams'
schreibe Datei '.wta_matrix.daten_1' und starte 4. Berechnung

vergleiche Graphen f3 und f3
initialisiere Graph mit Schwellen 1, 1

#### Protokollunterbrechung, es folgen Berechnungen

stabile Loesung nach 82 Berechnungen (-5)
7 Knoten in der Loesung
Gueten  : 0.0263158 0.35 0.0263158

3 Berechnungen erfolgt (3stabile, 0 unstabile Loesungen)
DurchschnittsAnzahl der Berechnungen = 59
GesamtZeit fuer KompGraphBildung: 0.15 sec.
GesamtZeit fuer Iterationen     : 0.68 sec.

                                              Taste druecken
```

*Ausgabeprotokoll 8-23 - neuen Prozeß starten*

#### 8.2.3.2.3 Daten vervollständigen

Wurde eine Berechnung unterbrochen, so kann es vorkommen, daß unvollständige Datensätze mit Berechnungsergebnissen vorhanden sind. Mit dem Menüpunkt 'vervollständigen' können diese Datensätze in den meisten Fällen vervollständigt werden (Ausgabeprotokoll 8-24).

```
lese MerkmalsStruktur aus Datei '.wta_matrix.structur'
lese Graphenliste aus Datei '.wta_matrix.graphen'
f3 f4 d190 e18
lese MatrixSystemDaten aus Datei '.wta_matrix.matrix'
prozess_anz = 2; berechn_anz = 3; graph_anz = 4
suche naechste noch nicht abgeschlossene Datei
pruefe '.wta_matrix.daten_0' - lese - komplett
pruefe '.wta_matrix.daten_1' - lese - nicht komplett, (33% gerechnet)
soll Datei '.wta_matrix.daten_1' fertig gerechnet werden ??
                    J - ja, N - nein, A - Abbruch
```

*Ausgabeprotokoll 8-24 - Datensätze vervollständigen*

*Vorsicht: Falls der Prozeß, welcher diesen Datensatz berechnet, noch nicht beendet ist, zerstört die Nutzung dieses Menüpunktes den Datensatz. Mit dem folgenden Menüpunkt kann der Rechner ermittelt werden, auf welchem der Prozeß lief - es ist wichtig, zu prüfen, ob dieser noch aktiv ist.*

#### 8.2.3.2.4 Status- Informationen

Dieser Menüpunkt gibt Informationen über den aktuellen Stand der Berechnungen an - dazu werden die momentan vorhandenen Datensätze geprüft (Ausgabeprotokoll 8-25).

```
aktueller MatrixBerechnungsStatus:
prozess_anz = 2; berechn_anz = 3; graph_anz = 4
'.wta_matrix.daten_0': Berechnung abgeschlossen; Rechner: bartok
'.wta_matrix.daten_1': nicht komplett, (33% gerechnet); Rechner: bartok
```

*Ausgabeprotokoll 8-25 - Menüpunkt Status- Informationen*

### *8.2.3.3 fortlaufende Berechnung*

Unter bestimmten Bedingungen kann es sinnvoll sein, die Gesamtberechnung in sehr viele Teilberechnungen zu teilen, und einen Rechner immer, wenn dieser den aktuellen Datensatz beendet hat, einen neuen berechnen zu lassen. Die beiden folgenden Menüpunkte realisieren genau dies - während Punkt '1' als erstes 'verteilt starten' aufruft und danach immer mit 'neuen Prozess' fortfährt, führt Punkt 2 fortlaufend 'neuen Prozess' aus. Sind keine weiteren Datensätze zu berechnen, so wird das Programm bei diesen beiden Menüpunkten beendet - alle vorher im System vorhandenen Graphen gehen unweigerlich verloren.

Zum Abschluß besitzt das Matrixmenü noch eine Rücksprungmöglichkeit zum Hauptmenü.

# 9 Anwendungsbeispiele

Zur Verdeutlichung verschiedener Einsatzmöglichkeiten und Funktionen des Programms sollen nun einige Vergleiche und Vergleichsergebnisse vorgestellt werden. Die kompletten Beschreibungsdateien sind dabei für alle folgenden Beispiele in Anhang B zu finden.

## 9.1 Block- Kreis- Beispiel Vergleich

Hier werden die beiden Graphen aus Abschnitt 2.1. und Abschnitt 3.1. verglichen. Die folgende Markierungsbeschreibung wurde dafür genutzt:

```
Knoten:
symbol,
reell_2(0.5,2)
Kante:
reell_2(0.1,2)
```

*Programm 9-1 - Markierungsbeschreibung*

Die Bedeutung dieser Beschreibung wird am besten im Vergleich mit den unteren Abbildungen deutlich. Knoten bestehen aus Symbolen, für welche Identität verlangt wird und einer reellen Zahl. Da die reelle Zahl das einzige gewichtete Knotenmerkmal ist, hat die Gewichtung mit 0.5 keine Bedeutung. Weiterhin ist festgelegt, daß bei einem Abstand der zu vergleichenden rellen Zahlen von 2 deren Ähnlichkeit auf 0 zurückgegangen ist.

Die Kanten werden ebenfalls mit einer reellen Zahl beschrieben, für die, wie bei den rellen Zahlen der Knotenmarkierungen, die Ähnlichkeit zweier Zahlen innerhalb eines Bereiches von 2 auf 0 zurückgeht. (siehe Abschnitt 2.4.1.1.)

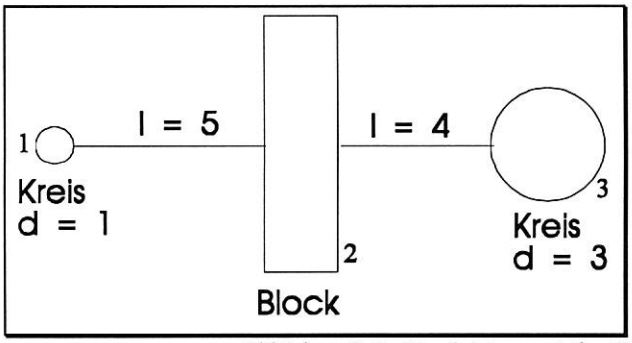



*Abbildung 9-1 - Vergleichsgraph 'bsp1'*

```
1_1:Kreis,1
1_2:Block,0
1_3:Kreis,3
1_1,1_2:5
1_2,1_3:4
```

*Programm 9-2 - Graphbeschreibung 'bsp1.graph'*

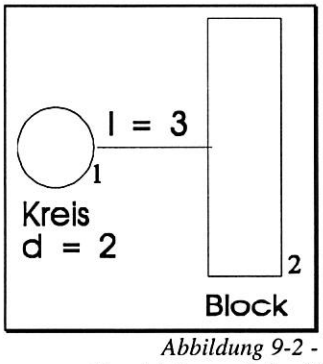



*Abbildung 9-2 - Vergleichsgraph 'bsp2'*

```
2_1:Kreis,2
2_2:Block,0
2_1,2_2:3
```

*Programm 9-3 - Graphbeschreibung 'bsp2.graph'*

Der folgende Vergleich beider Graphen ergab diese Ergebnisse:

bei ausschließlicher Berücksichtigung der Identität:
(Knoten- und Kantenschwelle = 1)
Güte1 = 0, Güte2 = 0.333, Güte3 = 0;
(siehe Abbildung 9-3)

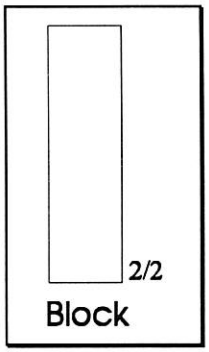



*Abbildung 9-3 - Vergleichsergebnis 1*

Das Ergebnis in Abbildung 9-3 zeigt deutlich den einzig möglichen identischen Subgraphen beider Ausgangsgraphen. Die Vergleichsgüten werden dabei nach den in Abschnitt 4 gezeigten Vorschriften berechnet. Da das WTA- Netz nur aus einem Neuron besteht, ergibt sich für alle Gütewerte, welche die Kanten berücksichtigen (Güte1 und Güte3) ein Wert von 0.

bei
schrittweisem Matching (lose), Schritte {0, 0.5, 1}
oder
Anfangsaktivierung aus Knotenähnlichkeit
Güte1 = 0.3333, Güte2 = 0.5, Güte3 = 0.1667;
(siehe Abbildung 9-4)

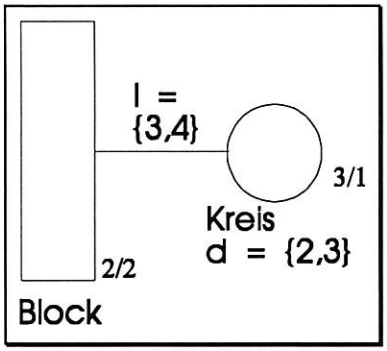



*Abbildung 9-4 - Vergleichsergebnis 2*

Bei diesem Ergebnis ist deutlich zu erkennen, daß der bereits im vorigen Beispiel vorhandene Ergebnisblock wiederum auftritt, da dieser natürlich mit einer Zuordnungsähnlichkeit von 1 sehr großes Gewicht im Netz hat. Zusätzlich wird hier die Zuordnung zweier Kreise hinzugefügt, da diese möglich ist- wenn auch mit einer geringeren Ähnlichkeit.

## 9.2 Vergleich von chemischen Verbindungen

Im folgenden sollen die diversen Vergleichsverfahren auf Moleküle aus dem von Dr. R. King veröffentlichten Datensatz mit stickstoffhaltigen aromatischen Verbindungen [King 1995] angewandt werden.

Als erstes gebe ich dazu die Markierungsbeschreibung der Molekül- Graphen an:

```
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)
```

*Programm 9-4 - Markierungsbeschreibung für Moleküle*

Wie in (Programm 9-4) deutlich zu sehen ist, werden für den Vergleich der Atome (Graphknoten) lediglich deren Bezeichner (begriff) genutzt, das darauf folgende Symbol sowie die darauf folgende reelle Zahl werden an dieser Stelle der Einfachheit halber ignoriert.

Beachten wir die Parameter, welche dem Merkmal 'begriff' übergeben werden, können wir feststellen, daß der Name der beim Vergleich zu berücksichtigenden Hierarchie 'atomhier' ist (Abbildung 9-5) und das zur Quantifizierung der Begriffsähnlichkeiten nur die Resthierarchie genutzt werden soll. (siehe Abschnitt 2.4.5.)

Die Kantenmarkierungen bestehen aus Symbolen, welche für eine Ähnlichkeit von 1 (Identität) übereinstimmen müssen, sonst kann keine Ähnlichkeit bestätigt werden.

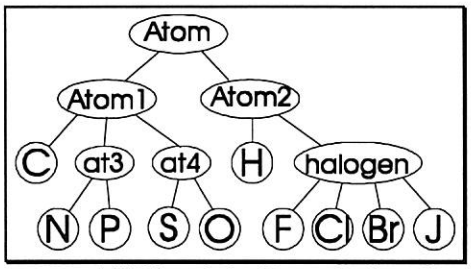



*Abbildung 9-5 - Hierarchie 'atomhier'*

Nun sollen die Graphen, welche verglichen wurden, gezeigt werden- sämtliche textuellen Graphbeschreibungen sind dabei (auch für die Ergebnisgraphen) im Anhang B zu finden. Die Zahlen an den Atomen der Moleküle geben die Nummer an, welche in der Graphbeschreibungsdatei den entsprechenden Graphknoten zugeordnet wurde.

Vergleichsgraphen aus dem 42- Graphen Datensatz:

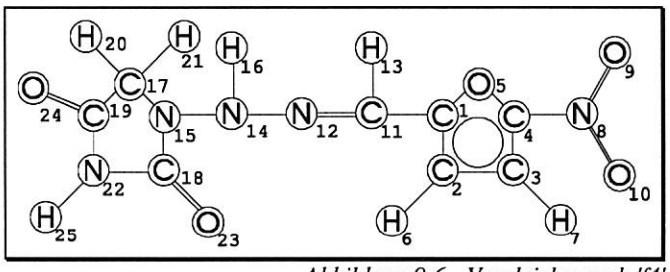


*Abbildung 9-6 - Vergleichsgraph 'f4'*

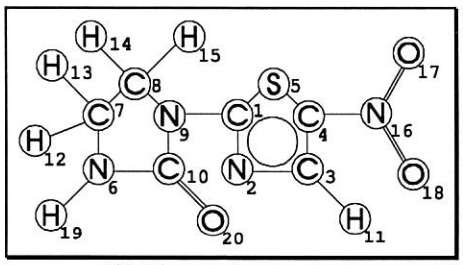


*Abbildung 9-7 - Vergleichsgraph 'f5'*

Die beiden Graphen 'f4' und 'f5' wurden nun mit verschiedenen Einstellungen verglichen. Alle Parametereinstellungen, welche dabei nicht den Standardeinstellungen (Programm 9-5) entsprechen, werden für jeden Vergleich angegeben und danach wird der Ergebnisgraph gezeigt.

```
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 1
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert
```

*Programm 9-5 - Standardeinstellungen der Parameter*

Die Atome in diesen Molekülen sind mit zwei Zahlen beschriftet, die erste entspricht dem Atom aus Graph 'f4', die zweite dem aus 'f5', welche einander zugeordnet wurden.

9.2.1 **Standardeinstellungen** (Programm 9-5), nur identische Zuordnungen

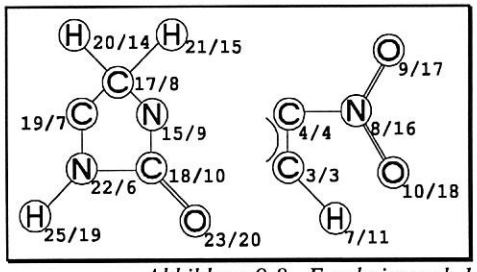



*Abbildung 9-8 - Ergebnisgraph 1*

Vergleichsgüten: Güte1 = 0.0466667, Güte2 = 0.600299, Güte3 = 0.0466667

### 9.2.2 untere AehnlichkeitsSchwelle fuer Knoten = 0.6

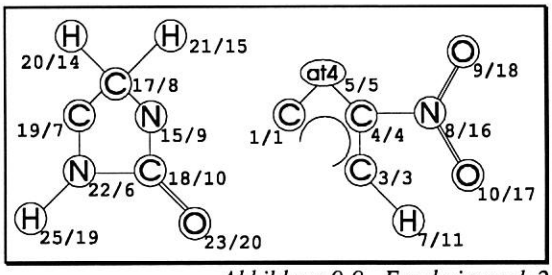



*Abbildung 9-9 - Ergebnisgraph 2*

Vergleichsgüten : Güte1 = 0.0533333, Güte2 = 0.66697, Güte3 = 0.0533333

Durch das Senken der Ähnlichkeitsschwelle wird das Netz um Neuronen einer geringeren Ähnlichkeit erweitert, der Ergebnisgraph wird um eine Schwefel/Sauerstoff- Zuordnung ('at4') erweitert.

### 9.2.3 untere AehnlichkeitsSchwelle fuer Knoten = 0.5

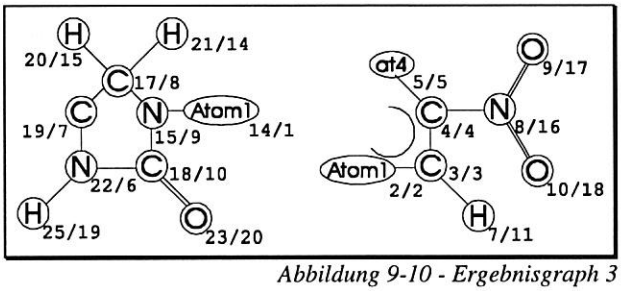



*Abbildung 9-10 - Ergebnisgraph 3*

Vergleichsgüten : Güte1 = 0.0566667, Güte2 = 0.666667, Güte3 = 0.0566667

Das weitere Senken der Ähnlichkeitsschwelle führt zur Aufnahme zusätzlicher Neuronen in das Netz, welche sich natürlich auch im Ergebnisgraph wiederspiegeln. So wird der Ergebnisgraph um neue Generalisierungen erweitert ('Atom1'), allerdings fällt auch auf, daß günstige Zuordnungen (wie $C_{1/1}$ in Abbildung 9-9) auch verändert werden können ($Atom1_{14/1}$)

### 9.2.4 untere AehnlichkeitsSchwelle fuer Knoten = 0.6
**AnfangsAktivierung der Knoten :**
**aus Graphgroesse errechnet und mit KnotenAehnlichkeit gewichtet**

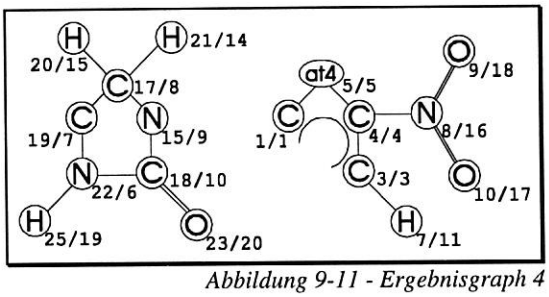



*Abbildung 9-11 - Ergebnisgraph 4*

Vergleichsgüten : Güte1 = 0.0533333, Güte2 = 0.666969, Güte3 = 0.0533333

### 9.2.5 untere AehnlichkeitsSchwelle fuer Knoten = 0.5
**AnfangsAktivierung der Knoten :**
**aus Graphgroesse errechnet und mit KnotenAehnlichkeit gewichtet**

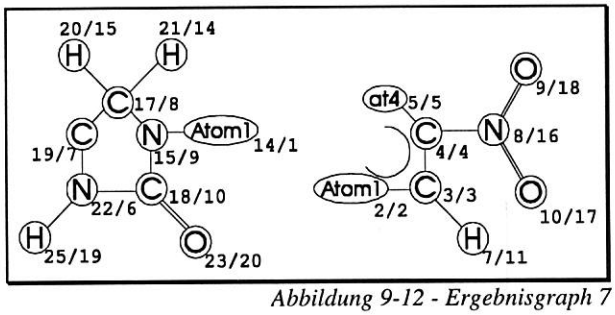



*Abbildung 9-12 - Ergebnisgraph 7*

Vergleichsgüten : Güte1 = 0.0566667, Güte2 = 0.666667, Güte3 = 0.0566667

### 9.2.6 untere AehnlichkeitsSchwelle fuer Knoten = 0.6
**externer Input :**
**Knoten mit Aehnlichkeit 1 kann x Prozent der Kanten des Graphen aufwiegen (x = 10)**

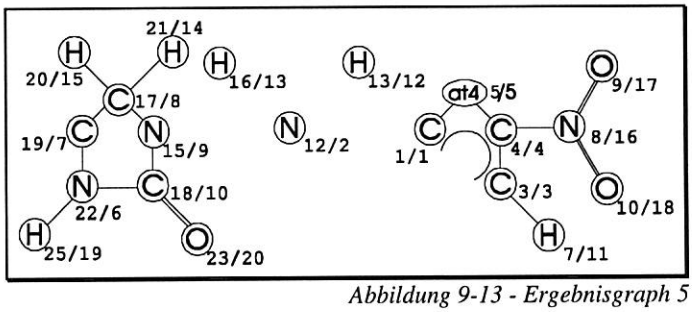



*Abbildung 9-13 - Ergebnisgraph 5*

Vergleichsgüten : Güte1 = 0.158095, Güte2 = 0.786667, Güte3 = 0.0533333

Durch das Hinzufügen eines externen Inputs treten nun im Ergebnisgraph auch einzelne Knoten ohne Kanten auf, da die fehlenden Kanten durch den externen Input kompensiert werden können.

### 9.2.7 untere AehnlichkeitsSchwelle fuer Knoten = 0.5

**externer Input :**

**Knoten mit Aehnlichkeit 1 kann x Prozent der Kanten des Graphen aufwiegen (x = 10)**

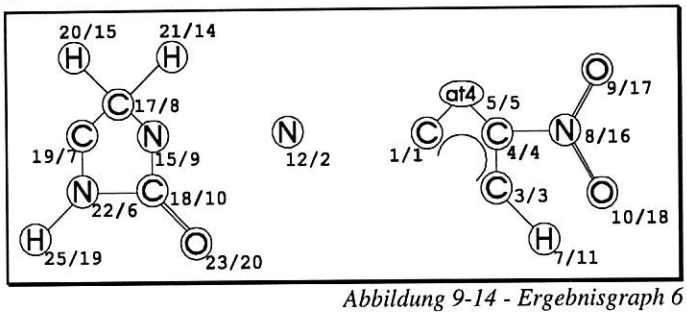



*Abbildung 9-14 - Ergebnisgraph 6*

Vergleichsgüten : Güte1 = 0.151393, Güte2 = 0.739752, Güte3 = 0.0533333

### 9.2.8 untere AehnlichkeitsSchwelle fuer Knoten = 0.6

**KantenSkalierung :**

**Kanten anhand der Kanten- und Knotenaehnlichkeiten skaliert**

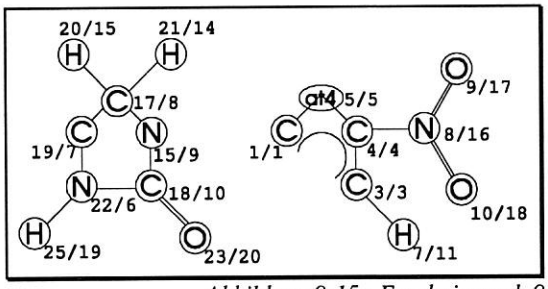



*Abbildung 9-15 - Ergebnisgraph 9*

Vergleichsgüten : Güte1 = 0.0511111, Güte2 = 0.66697, Güte3 = 0.0533333

### 9.2.9 untere AehnlichkeitsSchwelle fuer Knoten = 0.5

**KantenSkalierung :**

**Kanten anhand der Kanten- und Knotenaehnlichkeiten skaliert**

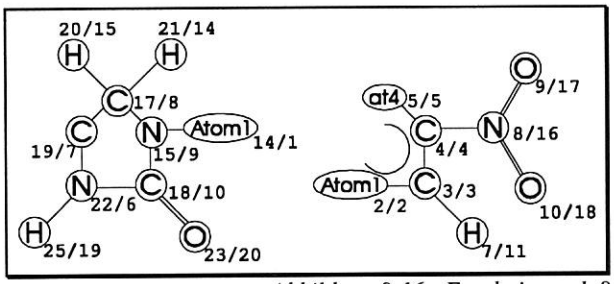



*Abbildung 9-16 - Ergebnisgraph 8*

Vergleichsgüten : Güte1 = 0.0522222, Güte2 = 0.666667, Güte3 = 0.0566667

### 9.2.10 schrittweises Matching (lose)
### AehnlichkeitsSchwelle(n) fuer Knoten : {0.5,0.6,1}

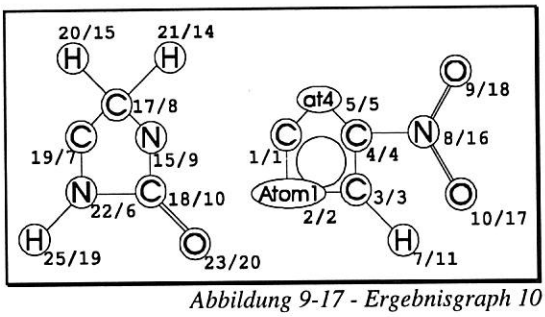



*Abbildung 9-17 - Ergebnisgraph 10*

Vergleichsgüten : Güte1 = 0.06, Güte2 = 0.686667, Güte3 = 0.06

Bei dieser Netzeinstellung ergibt sich offensichtlich das beste Ergebnis- es entsteht der erwartete Ergebnisgraph.

### 9.2.11 schrittweises Matching (fixiert)
### AehnlichkeitsSchwelle(n) fuer Knoten : {0.5,0.6,1}

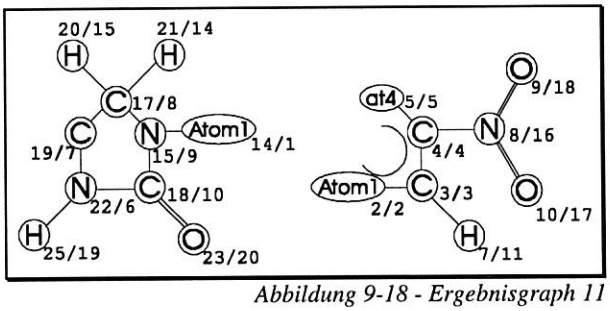



*Abbildung 9-18 - Ergebnisgraph 11*

Vergleichsgüten : Güte1 = 0.0566667, Güte2 = 0.666667, Güte3 = 0.0566667

Beim fixierten Matching wird hier ein besonderes Problem deutlich. Der Unterschied im Ergebnis zu Abbildung 9-17 besteht nur im fehlenden Atom $C_{1/1}$. Da dieses jedoch im ersten Schritt mit Schwelle 1 nicht zum Ergebnis gehörte (siehe auch Abbildung 9-8), wird das entsprechende Neuron beim Übergang zur nächsten Schwelle (0.6) entfernt und kann dann nicht mehr zur Vervollständigung des Ergebnisses genutzt werden.

### 9.2.12 schrittweises Matching (reduziert)
### AehnlichkeitsSchwelle(n) fuer Knoten : {0.5,0.6,1}

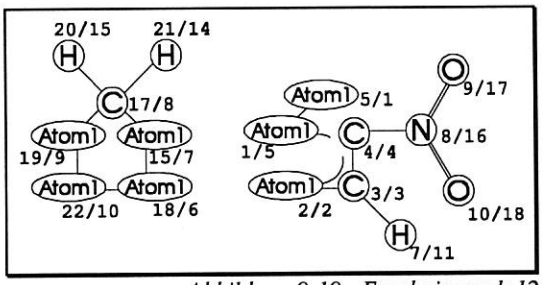



*Abbildung 9-19 - Ergebnisgraph 12*

Vergleichsgüten : Güte1 = 0.0466667, Güte2 = 0.5, Güte3 = 0.0466667

## 9.3 Generalisierung von chemischen Verbindungen

Eine weitere Nutzung der Vergleichsmöglichkeiten des Programms besteht in der Generalisierung verschiedener Graphen. Im folgenden Beispiel werden 8 chemische Verbindungen aus dem schon im Abschnitt 9.2 genutzten Datensatz stickstoffhaltiger aromatischer Verbindungen von Dr. R. King [King 1995] generalisiert. Dabei wurden alle Graphen mit folgender Markierungsbeschreibung eingelesen:

```
Knoten:
begriff(0.8,atomhier,Gesamt),
symbol(0.4),
reell_2(0.2,0.3,0.2)
Kante:
symbol(1)
```

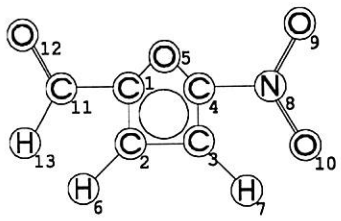
*Programm 9-6 - Markierungsbeschreibung*

Auffallend ist, daß trotz (teilweise) identischer Graphen an dieser Stelle eine andere Markierungsbeschreibung als in Programm 9-4 genutzt wird. Die Ursache hierfür ist die Nutzung der kompletten Beschreibungsinformationen zur Generalisierung, da die Generalisierung nur auf diese Weise zu einem zufriedenstellenden Ergebnis führt. Die Gewichtung der verschiedenen Knotenmerkmale zeigt, daß der Begriff, welcher die Art des Atoms beschreibt, immer noch sehr bedeutend ist, jedoch auch die in der Graphbeschreibung folgenden Daten, das Symbol und der reelle Wert, Einfluß auf den Vergleichsablauf gewinnen.

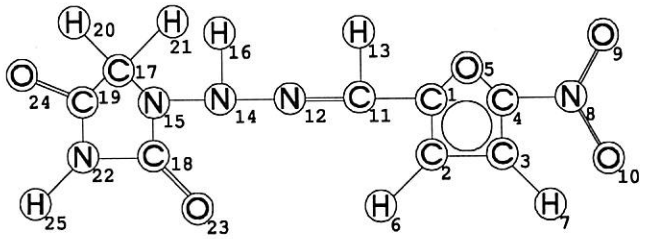
Für die Abbildung der Ähnlichkeit wurde die Beeinflussung der Anfangsaktivierung genutzt, weil im Vergleich der Generalisierungsergebnisse aus verschiedenen Verfahren hierbei die besten Resultate erzielt wurden. Die Knotenschwelle wurde auf 0 festgelegt, was bedeutet, daß alle möglichen Knotenzuordnungen berücksichtigt werden.

Graphen, welche generalisiert werden sollen:

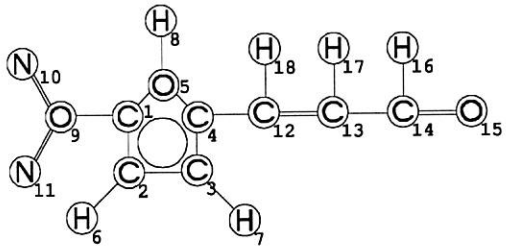
*Molekül f1:*



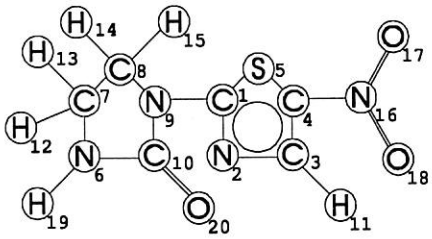
*Molekül f4:*



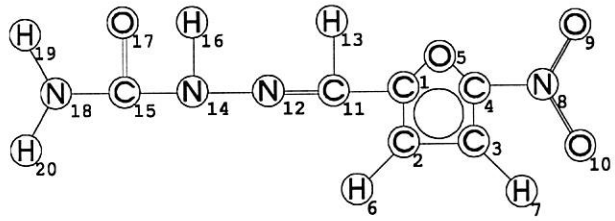
*Molekül f2:*



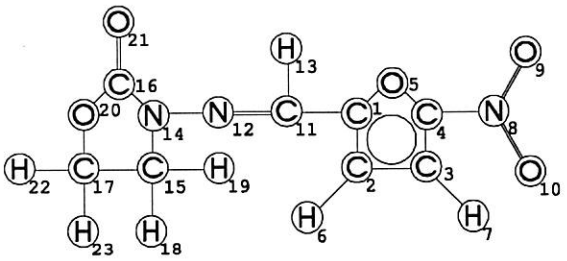
*Molekül f5:*



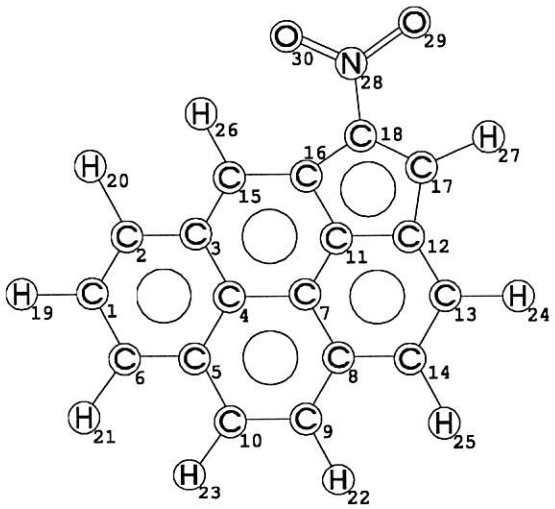
*Molekül f3:*



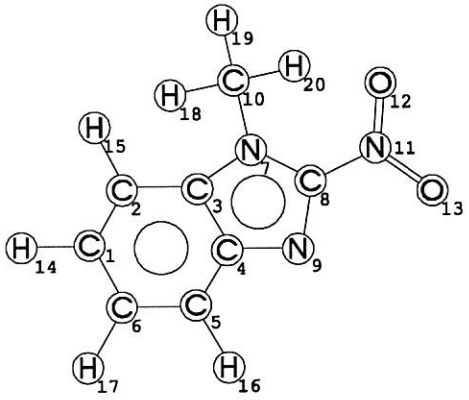
*Molekül f6:*



*Molekül d191:*



*Molekül d197:*



Außerdem wurde die schon im vorigen Abschnitt verwendete Begriffshierarchie genutzt (nochmals dargestellt in Abbildung 9-20).

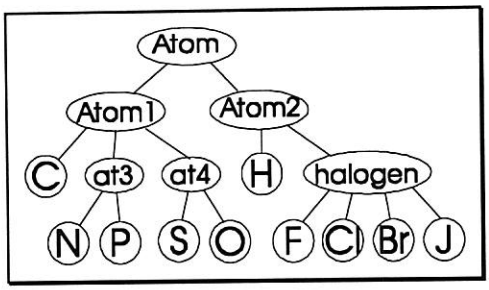



*Abbildung 9-20 - Hierarchie der Atome*

Alle Graphen wurden nun miteinander verglichen, was an dieser Stelle bedeuten soll, daß die Generalisierung zweier Graphen wiederum mit einem neuen Graphen verglichen wurde und die sich daraus ergebende Generalisierung wieder mit einem neuen Graphen und so weiter. Dafür wurden die Graphen, beginnend mit dem kleinsten, miteinander verglichen - dies entspricht folgender Reihenfolge: f1, f2, f3, d197, f5, f6, f4, d191.

Das Ergebnis der Generalisierung ist in Abbildung 9-21 zu sehen.

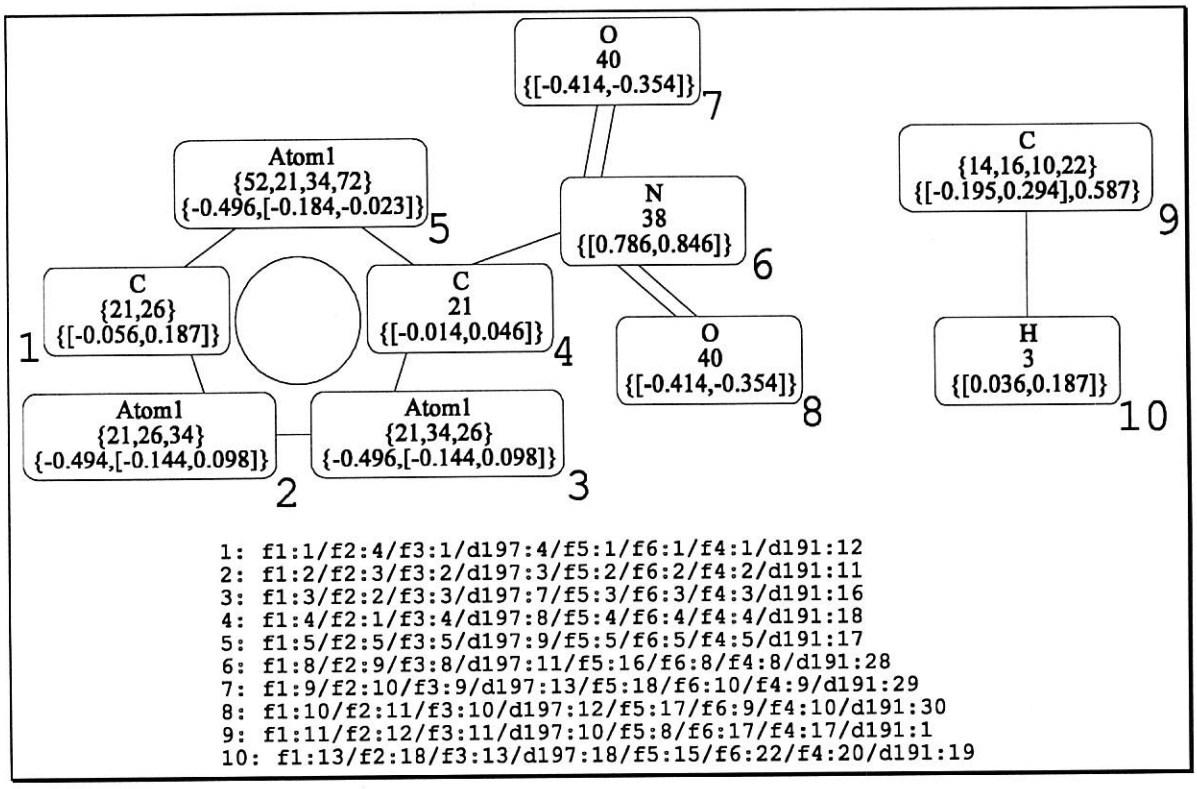



*Abbildung 9-21 - Generalisierung*

Im generalisierten Ergebnis zeigen sich deutlich die Ringstruktur und die $NO_2$ - Gruppe, welche in allen Molekülen auftreten. Diese Gemeinsamkeit kann jedoch erst durch die Generalisierung verschiedener Atome zu 'Atom1' nach der oben gezeigten Hierarchie beschrieben werden.

Unterhalb des Generalisierungsgraphes ist die Zuordnung der Atome der Ausgangsgraphen zu den Knoten des Ergebnisgraphen dargestellt.

Wie im vorigen Abschnitt sind auch hier die Graphbeschreibungsdateien aller Graphen im Anhang B zu finden.

# 10 Bewertung der verschiedenen Ähnlichkeitsabbildungen

Wie vielleicht schon im vorigen Abschnitt aufgefallen ist, kann eine Bewertung der verschiedenen Verfahren der Abbildung von Ähnlichkeiten auf das WTA- Netz nur sehr schwer aufgrund eines einzelnen Testvergleichs erfolgen. Das Vergleichsergebnis ist (natürlich) zu abhängig von der genauen Struktur und den Besonderheiten der verglichenen Graphen.

Eine Möglichkeit, die verschiedenen Abbildungsverfahren nach Abschnitt 3 miteinander zu vergleichen besteht jedoch in einem Klassifizierungstest.

Genutzt wurden hierbei wieder die von Dr. R. King veröffentlichten Datensätze [King 1995] mit chemischen Verbindungen - für beide (den mit 42 Molekülen und den mit 188) wurde die ausführlich in [Lowe 1993] beschriebene Klassifikation (Variable Kernel Similarity Learning, VSM) genutzt. Zusätzlich sollte die Signifikanz der ermittelten Unterschiede zwischen den verschiedenen Abbildungsverfahren mit dem in [Dietterich 1997] beschriebenen 'k-fold crossvalidated paired t- Test' geprüft werden. Hier soll nun kurz der Ablauf dieser Tests beschrieben werden.

Als erstes- und dies ist der zeitaufwendigste Teil- wurden alle zu berücksichtigenden Graphen untereinander verglichen. Dies sind bei 42 Graphen 861 Vergleiche, bei 188 Graphen allerdings schon 17578 - also ein enormer Rechenaufwand. Nach den Vergleichen, bei welchen nur die Vergleichsgüten wichtig sind (die Ergebnisgraphen werden ignoriert) wird eine Matrix mit allen Graphähnlichkeiten gebildet (siehe Abschnitte 5.5. und 8.2.3.).

Diese Graphähnlichkeiten bilden die durch verschiedene Strukturen der chemischen Verbindungen beschriebenen Ähnlichkeiten auf ein Abstandsmaß ab, welches nun für die Klassifizierung genutzt werden kann.

Für die Klassifizierung muß nun die Graphmenge in Trainings- und Testdaten zerteilt werden- dies erfolgt nach der in [Dietterich 1997, S.5] unter 'k-fold crossvalidated paired t- Test' beschriebenen Vorgehensweise. Dabei wird die Gesamtmenge der Graphen in $k$ gleich große Teile zerteilt, was bedeutet, daß $k$ Tests durchgeführt werden müssen. Für jeden dieser Tests wird einer dieser $k$ Teile als Test- Set genutzt, mit den anderen $k-1$ Teilen wird der Klassifikator trainiert. Bei dem Datensatz mit 42 Graphen wurde $k_{42}=42$ gesetzt, beim Datensatz mit 188 Graphen wurden 10 verschiedene Testsets gebildet ($k_{188}=10$).

Nach [Lowe 1993, S.5f] wird nun der Einfluß der $x$ nächsten Nachbarn des zu klassifizierenden Graphen durch deren (Ähnlichkeits-) Entfernung, gewichtet mit einer Gauß- Funktion, bestimmt. $M$ und $r$ sind dabei die Parameter der Gauß- Funktion. Die genaue Wahl dieser Parameter ($x$, $M$, $r$) ist in den Tabellen 10.1 und 10.5 jeweils angegeben.

Für die eben beschriebenen Klassifizierungsart wurden nun für die Bewertung der Vergleichsverfahren 3 verschiedene Wege zum Umgang mit dem so durch die Trainingsdaten erzeugten Gauß- Kernel genutzt. Natürlich ist eine Möglichkeit, alle Kerneldaten genau so zu nutzen, wie diese durch die Trainingsmenge erstellt wurden (in den Tabellen durch o gekennzeichnet). Da dies jedoch sehr viele Daten sind, werden in der zweiten verwendeten Methode alle Graphen, welche von Graphen gleicher Klasse umgeben sind, aus dem Kernel entfernt (in den Tabellen durch n gekennzeichnet).

Eine dritte genutzte Möglichkeit ist die Entfernung dieser Graphen aus dem Kernel nur dann, wenn sie selbst Element der durch die Umgebungsgraphen beschriebenen Klasse sind - dies ist sinnvoll, da aufgrund der (relativ) kleinen Testdatensätze, Graphen, welche eine andere Klasse

als alle Umgebungsgraphen haben, nicht unbedingt als Ausreißer betrachtet werden und darum entfernt werden können (t).

Nach der Klassifikation erhalten wir nun für alle $k$ Teiltests die Trefferquote (gemittelt über alle $k$ Teilergebnisse ergeben sich die in Tabelle 10.1 und Tabelle 10.5 angegebenen Werte) Die $k$ einzelnen Ergebnisse können nun für den t- Test nach [Dietterich 1997] genutzt werden, um signifikante Unterschiede zwischen den durchgeführten Tests nachzuweisen. Die Wahrscheinlichkeiten, mit der die in den Tabellen 10.1 und 10.5 angegebenen Werte signifikant verschieden sind, sind in den Tabelle 10.2 bis 10.4 und 10.6 bis 10.8 angegeben.

Die Klassifizierung wurde für alle drei in Abschnitt 4 vorgestellten Gütemaße vorgenommen, da jedoch jeweils bei Güte2 (Formel 4-3.) die besten Werte ermittelt wurden, werden hier nur diese Ergebnisse dargestellt.

Bei den schrittweisen Verfahren wurden folgende Knotenschwellen genutzt: {0.3, 0.6, 1}. Die Kantenskalierung wurde anhand der Kanten und Knotenähnlichkeit durchgeführt.

Alle nicht angegebenen Signifikanzen für Unterschiede zwischen den Abbildungsverfahren liegen unter 75 %.

**42er Datensatz:**

| Abbildungsart | kompletter Kernel (o), $x = 16, M = 16, r = 0.2$ | reduzierter Kernel (n), $x = 3, M = 3, r = 0.95$ | reduzierter Kernel (t) bei gleicher Klasse $x = 15, M = 15, r = 0.15$ |
|---|---|---|---|
| schrittweise, lose (1) | 85.7143 % | 83.3333 % | 80.9524 % |
| schrittweise, fixiert (2) | 85.7143 % | 83.3333 % | 80.9524 % |
| schrittweise, reduziert (3) | 76.1905 % | 88.0952 % | 80.9524 % |
| KantenSkalierung (4) | 80.9524 % | 83.3333 % | 83.3333 % |
| Kantenskalierung und Anfangsaktivierung (5) | 78.5714 % | 78.5714 % | 78.5714 % |

*Tabelle 10-1 - 42er Datensatz (k=42), Richtigklassifikation bei verschiedenen Kernels und Abbildungsverfahren*

Die in den folgenden Tabellen genutzten Zahlen zur Kennzeichnung der verschiedenen Abbildungsverfahren der Ähnlichkeiten auf das WTA- Netz entsprechen den in der Spalte 'Abbildungsart' in Tabelle 10-1 angegebenen Zahlen.

| Verfahren bei (o) | Signifikanz |
|---|---|
| (1), (3) | 90 % |
| (2), (5) | 90 % |
| (1), (4) | 75 % |
| (2), (4) | 75 % |
| (3), (4) | 75 % |
| (1), (5) | 90 % |
| (2), (5) | 90 % |

*Tabelle 10-2 - Signifikanz der Unterschiede bei kompletten Kernel*

| Verfahren bei (n) | Signifikanz |
|---|---|
| (1), (3) | 90 % |
| (2), (3) | 90 % |
| (3), (4) | 90 % |
| (1), (5) | 90 % |
| (2), (5) | 90 % |
| (3), (5) | 97.5 % |
| (4), (5) | 75 % |

*Tabelle 10-3 - Signifikanz der Unterschiede bei reduziertem Kernel*

| Verfahren bei (t) | Signifikanz |
|---|---|
| (4), (5) | 75 % |

*Tabelle 10-4 - Signifikanz der Unterschiede bei reduziertem Kernel (bei gleichen Klassen)*

**188er Datensatz:**

| AbbildungsArt | kompletter Kernel (o), $x = 16, M = 10, r = 0.25$ | reduzierter Kernel (n), $x = 10, M = 5, r = 0.35$ | reduzierter Kernel (t) bei gleicher Klasse $x = 14, M = 5, r = 0.3$ |
|---|---|---|---|
| schrittweise, lose (1) | 91.4849 % | 92.0213 % | 92.5532 % |
| schrittweise, fixiert (2) | 90.9574 % | 91.4894 % | 92.5532 % |
| schrittweise, reduziert (3) | 90.9574 % | 90.9574 % | 90.9574 % |
| KantenSkalierung (4) | 89.8936 % | 90.9574 % | 90.4255 % |
| Kantenskalierung und Anfangsaktivierung (5) | 90.4255 % | 89.8936 % | 90.4255 % |

*Tabelle 10-5 - 188er Datensatz (k=10), Richtigklassifikation bei verschiedenen Kernels und Abbildungsverfahren*

Die in den folgenden Tabellen genutzten Zahlen zur Kennzeichnung der verschiedenen Abbildungsverfahren der Ähnlichkeiten auf das WTA- Netz entsprechen den in der Spalte 'Abbildungsart' in Tabelle 10-5 angegebenen Zahlen.

| Verfahren bei (o) | Signi-fikanz |
|---|---|
| (1), (2) | 75 % |
| (1), (5) | 75 % |

*Tabelle 10-6 - Signifikanz der Unterschiede bei kompletten Kernel*

| Verfahren bei (n) | Signi-fikanz |
|---|---|
| (1), (2) | 75 % |
| (1), (3) | 75 % |
| (1), (5) | 90 % |
| (2), (5) | 75% |

*Tabelle 10-7 - Signifikanz der Unterschiede bei reduziertem Kernel*

| Verfahren bei (t) | Signi-fikanz |
|---|---|
| (1), (3) | 90 % |
| (2), (3) | 75 % |
| (1), (4) | 75 % |
| (2), (4) | 75 % |
| (1), (5) | 90 % |
| (2), (5) | 90 % |

*Tabelle 10-8 - Signifikanz der Unterschiede bei reduziertem Kernel (bei gleichen Klassen)*

Aus den Tabellen 10-1 und 10-5 geht deutlich hervor, daß die Verfahren des schrittweisen losen und schrittweisen fixierten Matchings überwiegend gute Ergebnisse liefern. Wie bereits erwähnt, zeichnen sich diese Verfahren zusätzlich durch eine geringe Vergleichszeit aus, da nicht sofort alle möglichen Knotenzuordnungen in das WTA- Netz aufgenommen werden. Die Verfahren mit Nutzung der Kantenskalierung bzw. Anfangsaktivierung schneiden hingegen, zumindest beim Datensatz mit 188 Graphen, signifikant schlechter ab.

# 11 Ausblick

Mit dem implementierten WTA- Netz wurde ein System entwickelt, welches einfach und übersichtlich den Vergleich beliebiger Graphen bei Berücksichtigung der lokalen Ähnlichkeiten unterstützt. Dieses System bietet eine Grundlage, um effektiv und schnell die Möglichkeiten dieses Ähnlichkeitsvergleichs zu ermitteln.

Durch die objektorientierte Implementierung ist eine einfache Anpassung an eigene Probleme möglich- während der Testphase des Programms haben sich die implementierten Ähnlichkeitsquantifizierungen und Merkmalsgeneralisierungen jedoch bereits gut bewährt.

Die Generalisierungsgraphen konnten durch die Berücksichtigung lokaler Ähnlichkeiten erweitert werden (siehe zum Beispiel 9.3), die gewählten Abbildungsverfahren der ermittelten Ähnlichkeiten auf das WTA- Netz können gut für die Ermittlung global ähnlicher Ergebnisgraphen genutzt werden.

Ein Problem besteht dennoch in der Rechenzeit des Programms. Durch die Berücksichtigung von Ähnlichkeiten bei den Knoten- bzw. Kantenzuordnungen für den Kompatibilitätsgraph entstehen Netze, welche hinsichtlich der Größe völlig andere Dimensionen haben als bei bloßer Abbildung identischer Zuordnungen (Tabelle 11-1).

| Abbildungsart | Schwelle für Knoten | Schwelle für Kanten | Anzahl der Knoten im Netz | Anzahl der Kanten im Netz | Anzahl der Propagationsschritte | Zeit für die Propagation |
|---|---|---|---|---|---|---|
| keine | 1 | 1 | 476 | 7823 | 711 | 120.25 sec. |
| Anfangs-aktivierung | 0.3 | 1 | 1152 | 39144 | 781 | 873.52 sec. |

*Tabelle 11-1 - Netzgrößen bei verschiedenen Ähnlichkeitsabbildungen*

So sind auch Matrixberechnungen (siehe Abschnitt 5.5), wie zum Beispiel mit 188 Graphen (17578 Vergleiche), nur noch effektiv auf mehreren Rechnern durchzuführen.

Die Unterstützung dieser verteilten Arbeit auf mehreren Rechnern könnte verbessert werden- im Moment ist lediglich eine Synchronistaion über die gemeinsamen Dateien möglich. Zusätzlich wäre zu überlegen, ob der objektorientierte Ansatz im Kern des Programms, der Netz-Propagation, aufgegeben werden sollte zugunsten einer schnelleren, jedoch damit schwerer zu ändernden Implementierung.

Als Ergebnis der Nutzung der verschiedenen Abbildungsarten von Ähnlichkeiten auf ein WTA-Netz kann abschließend festgestellt werden, daß die schrittweisen Verfahren (Abschnitt 3.3.) besonderes Interesse verdienen, da diese eine effektive Abbildungsart mit geringer Netzgröße (und damit geringer Rechenzeit) kombinieren.

# 12 Literatur:


Clauss 1970: Clauss, Günter; Ebner, Ebner: Grundlagen der Statistik. Verlag Harri Deutsch, Frankfurt a.M. und Zürich 1970

Dietterich 1997: Dietterich, Thomas G. : Statistical Tests for Comparing Supervised Classification Learning Algorithms. Department of Computer Science, Oregon State University, Corvallis, 27. Juni 1997

Falkenhainer 1989: Falkenhainer, Brian; Forbus, Kenneth D. : The Structure- Mapping Engine, Algorithm and Examples. in Artifical Intelligence 41 (1989/90), Elserier Science Publishers B.V. (North- Holland)

Harary 1974: Harary, Frank : Graphentheorie. R. Oldenbourg Verlag München Wien 1974

King 1995: King, R. D.; Sternberg, M. J. E.; Srinivasan, A.; Muggleton, S. H. : Knowledge Discovery in a Database of Mutagenic Chemicals. in: Proceedings of the Workshop "Statistics, Machine Learning and Discovery in Databases" at the ECML-95. 1995

Klemm 1994: Klemm, Elmar: Einführung in die Statistik, für Sozialwissenschaftler/innen, Script zur Vorlesung 'Methoden der Soziologie' WS 1994/95, TU Berlin

Lowe 1993: Lowe, David G. : Similarity Metric Learning for a Variable- Kernel Classifier. Computer Science Department, University of British Columbia, Vancouver, Canada, 25. November 1993 (veröffentlicht in 'Neural Computation')

Lui 1992: Lui, Young- Jon; Huang, Ching- Lai : Fuzzy Mathoematical Programming, Methods and Applications. Springer Verlag Berlin Heidelberg 1992

Schädler: Schädler, Kristina; Wysotzki, Fritz : A Connectionist Approach to the Distance- Based Analysis of Relational Data. Technical University of Berlin

Schädler 1996: Schädler, Kristina; Wysotzki, Fritz : Theoretical Foundations of a Special Neural Net Approach for Graphmatching. Technische Universität Berlin, Fachbereich 13, Informatik, Report 96-26, 11. Juli 1996

Schädler 1997: Schädler, Kristina; Wysotzki, Fritz : Untersuchungen zum Graphmatching mit speziellen Neuronalen Netzen. Technische Universität 14. Februar 1997

Schuchard 1980: Schuchard, C. et al. : Multivariate Analysemethoden, Eine anwendungsorientierte Einführung. Springer Verlag Berlin 1980

Stroustrup 1993: Stroustrup, Bjarne : The C++ programming language. Second Edition, ADDISON- WESLEY PUBLISHING COMPANY 1993

Valtchev 1997: Valtchev, Petko; Euzenat, Jérôme : Dissimilarity Measure for Collections of Objects and Values. in X. liu, P. Cohen, M. Berthold: Advances in Intelligent Data Analysis. Springer- Verlag Berlin Heidelberg 1997

Wysotzki 1994: Wysotzki, F.; Wiebrock, S. : Grundlagen der Künstlichen Intelligenz, Sommersemester 1994, TU Berlin, Fachbereich 13 (Informatik), FG Methoden der KI

# Anhang A Headerdateien

## Anhang A I - basis_liste.h

```
////////////////////////////////////////////////////////////////////////
//Die folgenden Klassen enthalten alle Methoden, welche unabhaengig//
// von der Listenart allen Listen gemeinsam sind.                    //
////////////////////////////////////////////////////////////////////////
#ifndef BASIS_LISTE_H
#define BASIS_LISTE_H

#include <stdio.h>
#include <iostream.h>

////////////////////////////////////////////////////////////////////////
// listElemArt dient zur Kennzeichnung der Intervalle. Dabei ist     //
// der Status neueIntGrenze nur zur Internen Verwaltung neu hinzu- //
// gefuegter IntervallGrenzen noetig, tritt also immer paarweise     //
// auf und wird bei der Reperatur der IntervallStrukturen beseitigt//
////////////////////////////////////////////////////////////////////////
enum listElemArt{Element, IntervallAnfang, IntervallEnde,
     neueIntGrenze, DEnde};


////////////////////////////////////////////////////////////////////////
// Durch die Realisierung dieser Klasse ueber templates ist es       //
// moeglich, die wichtigsten Methoden fuer verschiedene ObjektTypen//
// zu nutzen                                                         //
////////////////////////////////////////////////////////////////////////
template<class AnyType>
class basis_liste
{
 protected:
   struct ListElem                // Struktur eines Listenelements
     {
       AnyType data;
       enum listElemArt status;
       ListElem *prev, *next;
     };
   ListElem *first,               // Zeiger auf ListenKopf
            *last,                // Zeiger auf ListenEnde
            *pos,                 // Zeiger auf aktuelle Position
            *pos2,                // Zeiger auf zweite Position
            *pos3;                // Zeiger auf dritte Position

   void addDoppel(AnyType);       // Element wird hinzugefuegt
 public:
   basis_liste(void); // Konstruktor

   // Methoden bezueglich Zeiger pos
   AnyType get(void);        // gibt Element an aktueller Position zurueck
   AnyType* getFirst(void); // gibt Zeiger auf erstes Element zurueck
   AnyType* getNext(void);  // gibt Zeiger auf aktuelles Element zurueck
                       // und setzt internen Zeiger auf naechstes
   AnyType* getLast(void);  // gibt Zeiger auf letztes Element zurueck
   AnyType* getLastPos(void); // gibt Zeiger auf leztes Element zurueck
                         // und setzt internen Zeiger auf vorheriges
   AnyType* getPrev(void);  // gibt Zeiger auf aktuelles Element zurueck
                         // und setzt internen Zeiger auf vorheriges

   // Methoden bezueglich Zeiger pos2
   AnyType* getFirst2(void);// gibt Zeiger auf erstes Element zurueck
   AnyType* getNext2(void); // gibt aktuelles Element zurueck und setzt
                         // Zeiger auf naechstes, wenn ungleich pos

   // Methoden bezueglich Zeiger pos3
   AnyType* getFirst3(void);// gibt Zeiger auf erstes Element zurueck
   AnyType* getNext3(void); // gibt aktuelles Element zurueck und setzt
                         // Zeiger auf naechstes, wenn ungleich pos

   void reset(void);   // setzt aktuelle Position auf Start

   listElemArt getArt(void); // gibt Art des Elements zurueck
   int getStatus(void);      // gibt Information ueber die Anzahl der
                       // Elemente zurueck (0-leer, 1-eins, 2-mehr)
```



```
   void free(void);          // loescht alle PointerZiele und Elemente

   virtual void del(void) = 0;

   ~basis_liste(void);       // loescht alle ListenElemente
};


template <class AnyType>
class univ_liste : public basis_liste<AnyType>
{
 public:
   void del(void);           // loescht ListenElement
   void gibAusListe(ostream);// gibt Liste aus
   void gibAusListe(void)    // gibt ListenElemente aus
     (gibAusListe(cout);};
   virtual void add(AnyType) = 0;
   virtual void add(AnyType, AnyType)
     { cout << "ERROR: basis_liste set(AnyType,AnyType) nicht ueberlagert\n";
       exit(1); };
   virtual void setIntervall(float)
     { cout << "ERROR: basis_liste setIntervall(float) nicht ueberlagert\n";
       exit(1); };
   virtual bool listElem(AnyType)
     { cout << "ERROR: basis_liste listElem(AnyType) nicht ueberlagert\n";
       exit(1); };
};


template <class AnyType>
class pointer_liste : public basis_liste<AnyType>
{
 public:
   void del(void);           // loescht pointerZiel und ListenElement
   void gibAusListe(ostream);// gibt Liste aus
   void gibAusListe(void)    // gibt ListenElemente dereferenziert aus
     (gibAusListe(cout);};
   virtual void add(AnyType) = 0;
   virtual void add(AnyType, AnyType)
     { cout << "ERROR: basis_liste set(AnyType,AnyType) nicht ueberlagert\n";
       exit(1); };
   virtual void setIntervall(float)
     { cout << "ERROR: basis_liste setIntervall(float) nicht ueberlagert\n";
       exit(1); };
   virtual bool listElem(AnyType)
     { cout << "ERROR: basis_liste listElem(AnyType) nicht ueberlagert\n";
       exit(1); };
};

#endif
```

## Anhang A II - menge.h

```
////////////////////////////////////////////////////////////////////////
// Beschreibung spezieller Zugriffe fuer (nichtsortierte) Mengen    //
////////////////////////////////////////////////////////////////////////
#ifndef MENGE_H
#define MENGE_H

#include "basis_liste.h"


////////////////////////////////////////////////////////////////////////
// allgemeine Implementation fuer das Hinzufuegen von Elementen     //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class menge_basis
{
 public:
   void add(AnyType);       // fuegt ein Element zur Liste hinzu
   virtual void addDoppel(AnyType) = 0;
   virtual bool listElem(AnyType) = 0;
};

////////////////////////////////////////////////////////////////////////
// spezielle Methoden fuer 'univ_liste'-n                           //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class menge : public menge_basis<AnyType>,
              public univ_liste<AnyType>
{
 public:
   bool listElem(AnyType);
   void addDoppel(AnyType Datum)
     { basis_liste<AnyType>::addDoppel(Datum); };
   void add(AnyType Datum)
     { menge_basis<AnyType>::add(Datum); };
};

////////////////////////////////////////////////////////////////////////
// spezielle Methoden fuer 'pointer_liste'-n                        //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class menge_p : public menge_basis<AnyType>,
                public pointer_liste<AnyType>
{
 public:
   bool listElem(AnyType);
   void addDoppel(AnyType Datum)
     { basis_liste<AnyType>::addDoppel(Datum); };
   void add(AnyType Datum)
     { menge_basis<AnyType>::add(Datum); };
};

#endif
```



## Anhang A III - menge_sort.h

```
////////////////////////////////////////////////////////////////////////
// Klasse fuer eine Liste von Objekten, welche tatsaechlich (nicht  //
// bloss Zeiger auf diese) in der Liste gespeichert werden          //
////////////////////////////////////////////////////////////////////////
#ifndef MENGE_SORT_H
#define MENGE_SORT_H

#include "basis_liste.h"

////////////////////////////////////////////////////////////////////////
// allgemeine Methoden fuer sortierte Mengen                        //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class menge_sort_basis
{
 protected:
   float IntervallGroesse;
 public:
   menge_sort(void)                    // Konstruktor
     { IntervallGroesse = 0; };
   void setIntervall(const float Int)  // stellt die IntervallGroesse ein
     { IntervallGroesse = Int; };
};

////////////////////////////////////////////////////////////////////////
// Methoden fuer Listen, welche 'echte' Elemente verwalten          //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class menge_sort : public univ_liste<AnyType>,
                   public menge_sort_basis<AnyType>
{
   virtual void bildeIntervalle(void);     // bildet neue Intervalle, wenn
                                           // die Elemente zu dicht liegen
   void repairIntervalle(void);            // repariert die Intervallstruktur
                                           // aus {[x,[y,z]} wird {[x,z]}
 public:
   void add(AnyType);       // fuegt ein Element zur Liste hinzu
   void add(AnyType, AnyType); // fuegt Intervall in Liste ein
   bool listElem(AnyType); // prueft, ob Element in Liste ist
   void setIntervall(const float Int)
     { menge_sort_basis<AnyType>::setIntervall(Int); };
};

////////////////////////////////////////////////////////////////////////
// Methoden fuer Listen, welche Pointer verwalten                   //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class menge_sort_p : public pointer_liste<AnyType>,
                     public menge_sort_basis<AnyType>
{
   virtual void bildeIntervalle(void);     // bildet neue Intervalle, wenn
                                           // die Elemente zu dicht liegen
   void repairIntervalle(void);            // repariert die Intervallstruktur
                                           // aus {[x,[y,z]} wird {[x,z]}
 public:
   void add(AnyType);       // fuegt Element zur Liste hinzu
   void add(AnyType, AnyType); // fuegt Intervall in Liste ein
   bool listElem(AnyType); // prueft, ob dereferenziertes Element
                           // bereits in der Liste ist
   void setIntervall(const float Int)
     { menge_sort_basis<AnyType>::setIntervall(Int); };
};

#endif
```

## Anhang A IV - multiset.h

```
////////////////////////////////////////////////////////////////////////
// Multiset-Implementierung, einfachste Art einer Liste              //
////////////////////////////////////////////////////////////////////////
#ifndef MULTISET_H
#define MULTISET_H

#include "basis_liste.h"


////////////////////////////////////////////////////////////////////////
// Fuer alle Multisets wichtige Methoden                             //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class multiset_basis
{
 public:
   void add(AnyType);       // fuegt ein Element zur Liste hinzu
   virtual void addDoppel(AnyType) = 0;
};


////////////////////////////////////////////////////////////////////////
// Verweis auf die Standard- Methoden                                //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class multiset : public univ_liste<AnyType>,
                 public multiset_basis<AnyType>
{
 public:
   void addDoppel(AnyType Datum)
     { basis_liste<AnyType>::addDoppel(Datum); };
   void add(AnyType Datum)
     { multiset_basis<AnyType>::add(Datum); };
};

////////////////////////////////////////////////////////////////////////
// Verweis auf die Standard- Methoden                                //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class multiset_p : public pointer_liste<AnyType>,
                   public multiset_basis<AnyType>
{
 public:
   void addDoppel(AnyType Datum)
     { basis_liste<AnyType>::addDoppel(Datum); };
   void add(AnyType Datum)
     { multiset_basis<AnyType>::add(Datum); };
};

#endif
```



## Anhang A V - multiset_sort.h

```
////////////////////////////////////////////////////////////////////////
// sortiertes Multiset, alle Methoden zur Erweiterung der Basiskl. //
////////////////////////////////////////////////////////////////////////
#ifndef MULTISET_SORT_H
#define MULTISET_SORT_H

#include "basis_liste.h"

////////////////////////////////////////////////////////////////////////
// Methoden fuer sortierte Multisets mit Elementen                   //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class multiset_sort : public univ_liste<AnyType>
{
 public:
   void add(AnyType);       // fuegt ein Element zur Liste hinzu
   bool listElem(AnyType); // prueft, ob Element in Liste ist
};

////////////////////////////////////////////////////////////////////////
// Methoden fuer sortierte Multisets mit Pointern                    //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class multiset_sort_p : public pointer_liste<AnyType>
{
 public:
   void add(AnyType);       // fuegt Element zur Liste hinzu
   bool listElem(AnyType); // prueft, ob dereferenziertes Element
                           // bereits in der Liste ist
};

#endif
```

## Anhang A VI - file.h

```
////////////////////////////////////////////////////////////////////////
// Diese Struktur dient dazu, saemtliche Zugriffe auf die Dateien,  //
// welche zur Ein- und Ausgabe genutzt werden, zu verwalten bzw.    //
// durchzufuehren                                                   //
////////////////////////////////////////////////////////////////////////
#ifndef FILE_H
#define FILE_H

#include <stdio.h>
#include "multiset.h"


////////////////////////////////////////////////////////////////////////
// Der folgende Enumerator dient zur Beschreibunbg des aktuellen    //
// Lesestatus, und wird beim Lesen jedes neuen Wortes aktualisiert  //
////////////////////////////////////////////////////////////////////////
enum StatusVar{Objekt,Knoten,Kante,Konzept,KonzeptEnde,Intervall,
               Parameter,EndeMerkmal,DateiEnde};

class WordBuffer
{
 public:
   char* word;
   StatusVar status;
   int line;
};

////////////////////////////////////////////////////////////////////////
// Die grundlegende Klasse readfile verwaltet saemtliche Datei-     //
// zugriffe                                                         //
////////////////////////////////////////////////////////////////////////
class readfile
{
   FILE *InFile;       // Zeiger auf die geoeffnete Datei

   // zur Fehlersuche werden zwei ZeilenNummern verwaltet
   int line,           // Pos des letzten gueltigen Wortes (nextWord())
       nextLine;       // Pos des letzten gelesenen Zeichens

   bool ZeilenAnfang; // wird am ZeilenAnfang ein gueltiges Wort mit ','
                      // abgeschlossen, kann auf Kante getestet werden

   StatusVar status;  // enthaelt den aktuellen Status

   // in Einzelfaellen muessen mehr Daten als gewuenscht von der Datei
   // gelesen werden, um den aktuellen Status zu bestimmen etc.
   multiset_p<WordBuffer*> word_buffer;
   char     zeichen_buffer;     // Puffer fuer vorzeitig gelesene Zeichen

   // Hilfsmethoden fuer nextWord(), zum Pruefen des gelesenen Zeichens
   void kommentar(char *Zeichen, const int index);
   int wortEnde(const char Zeichen, const int index);
   int statusAenderung(const char Zeichen);

 protected:
   char *Name;         // DateiName

   // gibt Fehlermeldung aus und bricht Programm ab
   void readerror(const int);

 public:
   readfile(void);

   // setzt Dateizeiger auf Dateianfang zurueck
   void reset(void);

   // oeffnet Datei
   void open(char Datei[]);

   // gibt den aktuellen Status zurueck
   StatusVar getStatus(void) {return status;};
```



```
   // gibt den aktuellen Status aus
   void gibStatus(void);

   // gibt Zeiger auf naechstes gelesenes Wort zurueck
   char* nextWord(void);

   // prueft die ReadStatus- Variable und bricht den Programmablauf
   // eventuell ab
   bool error(void);

   // schliesst die Datei und gibt Speicher frei
   ~readfile(void);
};

void operator <<(class ostream ausgabestrom, class WordBuffer a);
bool operator ==(class WordBuffer a, class WordBuffer b);

#endif
```

## Anhang A VII - graph_file.h

```
//////////////////////////////////////////////////////////////////////
// Diese Struktur dient dazu, spezielle Zugriffe zum initialisieren//
// von Graph- bzw. Mekrmalstruktur als Erweiterung der Mutterklasse//
// readfile anzubieten                                              //
//////////////////////////////////////////////////////////////////////
#ifndef GRAPH_FILE_H
#define GRAPH_FILE_H

#include "file.h"
#include "merkstrukt.h"

//////////////////////////////////////////////////////////////////////
// Die grundlegende Klasse readfile verwaltet saemtliche Datei-     //
// zugriffe, die folgenden Klassen verwalten uebergeordnete Datei- //
// nutzungen.                                                       //
//////////////////////////////////////////////////////////////////////
class g_file : public readfile
{
 public:
   g_file(char[]);

   // sucht die naechste Knotenbeschreibung und gibt Zeiger auf den
   // gelesenen String zurueck
   char* nextKnoten(void);

   // sucht naechste Kantenbeschreibung, gibt zeiger auf String zurueck
   char* nextKante(void);
};


class s_file : public g_file
{
   // liest die Strukturbeschreibung bis zum Abbruch ein umd nutzt
   // die uebergebenen Felder zur Speicherung
   int readStruct(multiset_p<merkmal_buffer*>*,const StatusVar Abbruch);

 public:
   s_file(char[]);

   // liest die Strukturbeschreibung am Beginn einer Datei
   bool initStruct(void);
};

#endif
```



## Anhang A VIII - terminal.h

```
//////////////////////////////////////////////////////////////////////
// Klasse fuer saemtliche Eingaben (incl. Makroverwaltung)          //
//////////////////////////////////////////////////////////////////////
#ifndef TERMINAL_H
#define TERMINAL_H

#include <sys/termios.h>
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>

//
// ANSI- SteuerSequencen fuer tty- terminals
// Command              Description
// -------------------  ------------------------------------------------
// ESC[rowsA            Cursor up
// ESC[rowsB            Cursor down
// ESC[colsC            Cursor right
// ESC[colsD            Cursor left
// ESC[row;colH         Set cursor position (top left is row 1, column 1)
// ESC[2J               Clear screen
// ESC[K                Clear from cursor to end of line
// ESC[row;colf         Set cursor position, same as "H" command
// ESC[s                Save cursor position (may not be nested)
// ESC[u                Restore cursor position after a save
//

//////////////////////////////////////////////////////////////////////
// Die folgende Klasse enthaelt Methoden fuer die Eingabe, welche   //
// in aehnlicher Form, jedoch nicht mit diesem Komfort in den       //
// Standardbibliotheken vorhanden sind - alle Eingaben im Programm  //
// werden ueber diese Klasse abgewickelt.                           //
//////////////////////////////////////////////////////////////////////
class tty
{
   struct termios *termios_ptr;      // IO- Struktur des Terminals
   istream *instream;                // Eingabestrom
   ifstream *infile;                 // EingabeFileStrom
   void eofTest(void);               // testet auf Ende des Files
 public:
   tty(void);                        // Konstruktor
   // TerminalStatusMethoden (on/off - Schalter)
   void canonOn(void);               // Canonische Eingabe, Zeichen werden
erst
   void canonOff(void);              // nach Return zum Programm gesendet
   void echoOn(void);                // Tastatureingaben werden sofort auf
   void echoOff(void);               // dem Bildschirm dargestellt
   // Bildschirmausgabe
   void cls(void);                   // Bildschirm loeschen
   void setPos(int,int);             // Cursor auf Pos setzen
   void invers(void);                // inverse Darstellung
   void normal(void);                // normale Darstellung
   void savePos(void);               // merke Cursorposition
   void restorePos(void);            // stelle Cursorposition wieder her
   void up(int);                     // bewege Cursor nach oben
   void down(int);                   // bewege Cursor nach unten
   void killLine(void);              // loesche Zeile, in welcher Cursor
steht
   // Eingabemethoden
   bool killReturn(void);            // entferne Returns aus Eingabestrom
   bool getString(char*, int);       // lese String ein
   char getChar(void);               // lese Char ein
   float getFloat(void);             // lese Float ein
   int getInt(void);                 // lese Int ein
   void wait(void);                  // warte auf Tastendruck
   // Makromethoden
   void setInput(char*);             // setzt Eingabequelle auf Datei
   void resetInput(void);            // schliesst Datei und
                                     // setzt Quelle auf 'cin'
   ~tty(void);
};
```

```
//////////////////////////////////////////////////////////////////////
// Instanz der Klasse terminal - ueber diese Instanz werden alle    //
// Eingaben verwaltet                                               //
//////////////////////////////////////////////////////////////////////
extern tty term;

#endif
```



## Anhang A IX - merkstrukt.h

```
//////////////////////////////////////////////////////////////////////
// Diese Klassse enthaelt die Struktur der einzelnen Merkmale,      //
// das heisst deren Zuordnung zu den Objekt/Konzeptarten wie reell  //
// symbol etc. Ausserdem werden Zugriffsmoeglichkeiten auf die zu   //
// jeder Mermalsbeschreibung moeglichen Parameter gegeben.          //
// Zusaetzlich wird zur kontrolle des Einlesevorganges noch die     //
// jeweilige Anzahl der Einzelmerkmale registriert.                 //
//////////////////////////////////////////////////////////////////////

#ifndef MERKSTRUKT_H
#define MERKSTRUKT_H

#include <stdio.h>
#include <iostream.h>
#include "file.h"
#include "m_symbol.h"


//////////////////////////////////////////////////////////////////////
// Art der Merkmale, sollte erweitert werden, falls neue Merkmale   //
// neue Untermerkmale (reell-normalverteilt) zu beruecksichtigen    //
// sind.                                                            //
//////////////////////////////////////////////////////////////////////
enum merkmal_art{reell_1, reell_2, ganz_1, ganz_2, gmenge_1, gmenge_2,
     symbol, begriff, fehler};


//////////////////////////////////////////////////////////////////////
// Diese Funktion ordnet einem String (z.B. aus einer Datei) ein    //
// Element des Typs merkmal_art so zu, dass dies an Stelle des      //
// Strings gespeichert werden kann. Falls also Aenderungen an der   //
// Art der (Beschreibung) der Einzelmerkmale vorgenommen werden,    //
// muessen diese auch hier berueckgesichtigt werden.                //
//////////////////////////////////////////////////////////////////////
merkmal_art art(char[]);


//////////////////////////////////////////////////////////////////////
// Der Merkmal Puffer enthaelt den Merkmalsbezeichner aus der ein-  //
// gelesenen Datei und eine Liste von Symbolen, also unbewerteten   //
// Strings, welche als Parameter uebergeben wurden.                 //
//////////////////////////////////////////////////////////////////////
class merkmal_buffer
{
 public:
   merkmal_buffer(void);
   void free(void);
   char* merkmal;
   multiset_p<symbl*> *parameter;
};


//////////////////////////////////////////////////////////////////////
// Diese Klasse dient zur Speicherung der Parameter eines Merkmals, //
// sie ist nur als Strulturelement in der Klasse merkmal_struktur   //
// enthalten.                                                       //
//////////////////////////////////////////////////////////////////////
class para
{
 public:
   float Gewicht;
   int Anzahl;
   char **param;
   void free(void);
};


//////////////////////////////////////////////////////////////////////
// Strukturbeschreibung der Merkmale                                //
//////////////////////////////////////////////////////////////////////
```

```
class merkmal_struktur
{
   merkmal_art *Knoten,          // Art der Merkmale der Knoten
               *Kante,           // Art der ungerichteten Kantenmerkmale
               *KanteGerichtet; // Art der gerichteten Kantenmerkmale
   para        *ParaKnoten,      // ParameterBeschreibung der Merkmale
               *ParaKante,
               *ParaKanteGerichtet;
   int AnzKnoten, AnzKante,   // Anzahl der entsprechenden Einzelmerkmale
     AnzKanteGerichtet;

   void zeigeStruktur(merkmal_art*, // BasisMethode zur Strukturausgabe
                  int, para*, ostream);
   int setIt(multiset_p<merkmal_buffer*>* buffer, // BasisMethode zum Setzen
der Einzel-
             int, merkmal_art*, para*);  //          strukturen

 public:
   // Konstruktor, initalisiert alle Zeiger mit NULL
   merkmal_struktur(void);

   // Strukturelemente setzen, merkmal enthaelt den Name des Merkmals,
   // index dessen laufende Nummer, parameter enthaelt eine Liste mit
   // Zeigern auf ParameterStrings und para_index deren Anzahl
   int setKnoten(multiset_p<merkmal_buffer*>*, int);
   int setKante(multiset_p<merkmal_buffer*>*, int);
   int setKanteGerichtet(multiset_p<merkmal_buffer*>*, int);

   // liefert als Resultat die Merkmalsart des Einzelmerkmals, welches
   // an der durch Stelle festgelegten Stelle im Gesamtmerkmal steht
   merkmal_art MerkmalKnoten(int Stelle);
   merkmal_art MerkmalKante(int);
   merkmal_art MerkmalKanteGerichtet(int);
   merkmal_art Merkmal(int Stelle, int Typ)
     {switch(Typ)
     {case 0: return MerkmalKnoten(Stelle);
      case 1: return MerkmalKante(Stelle);
      default: return MerkmalKanteGerichtet(Stelle);};};

   // Methode zur StrukturAusgabe, uebergibt der BasisMethode die
   // entsprechenden Parameter
   void gibAusStruktur(ostream);
   void gibAusStruktur(void) {gibAusStruktur(cout);};

   // Die folgenden Methoden geben einen Zeiger auf die para - Struktur
   // des entsprechenden Merkmals zurueck, welche die genaueren Informationen
   // ueber die Parameter enthaelt
   para ParamKnoten(int Stelle) {return ParaKnoten[Stelle];};
   para ParamKante(int Stelle) {return ParaKante[Stelle];};
   para ParamKanteGerichtet(int Stelle) {return ParaKanteGerichtet[Stelle];};
   para Param(int Stelle, int Typ)
     {switch(Typ)
     {case 0: return ParaKnoten[Stelle];
      case 1: return ParaKante[Stelle];
      default: return ParaKanteGerichtet[Stelle];};};

   // liefert die Anzahl der Einzelmerkmale als Resultat
   int anzKnotenMerkmal(void) {return AnzKnoten;};
   int anzKantenMerkmal(void) {return AnzKante;};
   int anzKantenGerichtet(void) {return AnzKanteGerichtet;};

   void free(void);

   // Destruktor, gibt reservierten Speicher frei
   ~merkmal_struktur(void);
};


////////////////////////////////////////////////////////////////////////
// ueberladener Operator zur Ausgabe                                  //
////////////////////////////////////////////////////////////////////////
void operator <<(class ostream ausgabestrom,
                 class merkmal_buffer);
#endif
```

## Anhang A X - graph.h

```
////////////////////////////////////////////////////////////////////////
// Diese Klasse repraesentiert einen kompletten Graphen, und          //
// besteht im wesentlichen aus einem Zeiger auf ein Array mit         //
// Zeigern auf Knoten, und aus einem Zeiger auf ein Array mit         //
// Zeigern auf Kanten                                                 //
////////////////////////////////////////////////////////////////////////
#ifndef GRAPH_H
#define GRAPH_H

#include <stdio.h>
#include <iostream.h>
#include "graph_file.h"
#include "knoten.h"
#include "kanten.h"

class graph
{
   knoten_struktur *KnotenFeld;      // Zeiger auf Feld mit Knoten
   int KnotenAnz,                    // Anzahl der Knoten in dem Feld
       pos;                          // aktuelle Position im Feld

   kanten_struktur *KantenFeld;      // Zeiger auf Feld mit Kanten
   int KantenAnz,                    // Anzahl der Kanten in dem Feld
       kan_pos;                      // aktuelle Position im Feld

   bool grapherror(int, g_file*);    // Fehlermeldung und Abbruch

   enum K_art{_Kante, abgeh_Kante, ankom_Kante, _Knoten};
   void setMerkmal(int, int,         // setzt Merkmal bei K_art
                char*, char*, K_art);
   bool setIntervall(int, int,       // setzt Intervall bei K_art
                  char*, char*, char*, K_art);
   int setMerkmale(g_file*, int,     // uebertraegt gelesene Merkmale
                enum K_art, char*);

   void initKnoten(g_file*);         // Zaehlt die Knoten und initialisiert
                                     // das KnotenFeld
   int  readKnoten(g_file*, int);    // liest die Daten eines Knotens ein

   void initKanten(g_file*);         // Zaehlt die Kanten und initialisiert
                                     // das KantenFeld
   int  readKante(g_file*, int);     // liest die Daten einer Kante ein

 public:
   graph(void)                       // Konstruktor
     {KnotenFeld = NULL; KnotenAnz = 0; pos = 0;
      KantenFeld = NULL; KantenAnz = 0; kan_pos = 0;};

  // Methoden fuer Generalisierung
   void initKnoten(int a)            // initialisiere KnotenStruktur
     {  KnotenAnz = a;
        KnotenFeld = new knoten_struktur[KnotenAnz];};

   void initKanten(int a)            // intialisiere KantenStruktur
     {  KantenAnz = a;
        KantenFeld = new kanten_struktur[KantenAnz];};

  // Methoden fuer Lesen des Graphs aus einer Datei
   bool readGraph(g_file*);         // liest die GraphBeschreibung aus Datei

  // KnotenMethoden:
   knoten_struktur* gibKnoten(char*); // sucht Knoten nach Name
   knoten_struktur* nextKnoten(void)  // gibt Zeiger auf aktuellen Knoten
     {                                // und erhoeht Positionszeiger
       if (pos == KnotenAnz) {pos = 0; return(NULL);};
       return(KnotenFeld+pos++);
     };
   knoten_struktur* firstKnoten(void) // setzt Pos zurueck, gibt Zeiger
     {pos = 1; return(KnotenFeld);};  // auf ersten Knoten zurueck
```



```
   int gibKnotenAnzahl(void)          // gibt die Anzahl der Knoten zurueck
     {return KnotenAnz;};

  // KantenMethoden:
   kanten_struktur* gibKante(char*, char*); // sucht Kante anhand der Namen
   kanten_struktur* nextKante(void)  // gibt Zeiger auf aktuelle Kante
     {                               // und erhoeht Positionszeiger
       if (kan_pos == KantenAnz) {kan_pos = 0; return(NULL);};
       return(KantenFeld+kan_pos++);
     };
   kanten_struktur* firstKante(void)  // setzt Pos zurueck, gibt Zeiger
     {kan_pos = 1; return(KantenFeld);};  // auf erste Kante zurueck

   int gibKantenAnzahl(void)          // gibt die Anzahl der Kanten zurueck
     {return KantenAnz;};
   // GraphMethode:
   void gibAusGraph(ostream);       // gibt den Graph aus
   void gibAusGraph(void)           // gibt den Graph komplett aus
     {gibAusGraph(cout);};
   void free(void);
   ~graph(void);                     // Destruktor
};

#endif
```

## Anhang A XI - knoten.h

```
//////////////////////////////////////////////////////////////////////
// Die Klasse knoten_struktur enthaelt die komplette Beschreibung  //
// eines Knoten eines Ausgangs- bzw Ergebnisgraphen.               //
// Ein Knoten besteht aus dem KnotenName, Zeigern auf dessen       //
// Merkmale sowie auf Kanten, welche an diesem Knoten beginnen bzw.//
// enden.                                                          //
//////////////////////////////////////////////////////////////////////

#ifndef KNOTEN_H
#define KNOTEN_H

#include "basismerkmal.h"
#include "merkstrukt.h"
#include "kanten.h"

extern merkmal_struktur *merkstruct;

class knoten_struktur
{
   // Zeiger auf Feld mit Zeigern auf BasisMerkmale
   Basismerkmal **Merkmale;
   // Zeiger auf Feld mit Zeigern auf Kanten
   kanten_struktur **Kanten;
   char *Name;                  // Name des Knotens, einmalig im Graph !!
   int AnzKanten;               // Anzahl der Kanten an diesem Knoten
   void initMerkmal(int);       // ordnet Zeiger im Feld *Merkmal Element
                                // der Hierarchie Basismerkmal zu
 public:
   knoten_struktur(void);       // initialisiert Variablen, reserviert Speicher
                                // fuer *Merkmale und ruft initMerkmal() auf
 // allg. Methoden
   void setName(char* word)     // Name festlegen, nur einmal moeglich
     {if (Name == NULL) Name = word;};
   char* getName(void) {return(Name);}; // Name abfragen
   void gibAusKnoten(ostream);// gibt den Knoten aus
   void gibAusKnoten(void)      // gibt Knoten formatiert aus
     {gibAusKnoten(cout);};
 // Methoden fuer Kanten
   void addKante(void)          // Anzahl der Kanten am Knoten erhoehen
     {AnzKanten++;};
   void addKante(kanten_struktur*); // Kante hinzufuegen, beim ersten Aufruf
                                    // wird das Feld Kanten anhand AnzKanten
                                    // initialisiert
   kanten_struktur* gibKante(char*);// sucht Kante nach char* von diesem
Knoten
 // Methoden fuer Merkmale
   // fuege Intervall [word1,word2] zu Merkmal an MerkmalPos hinzu
   int setIntervall(int MerkmalPos,char* word1,char* word2)
     {return (*(Merkmale+MerkmalPos))->addIntervall(word1, word2);};
   // fuege Merkmal 'word' zu Merkmal an MerkmalPos hinzu
   void setMerkmal(int MerkmalPos,char* word)
     {(*(Merkmale+MerkmalPos))->addMerkmal(word);};

   // gibt Zeiger auf MerkmalFeld zurueck
   Basismerkmal** getMerkmal(void)
     { return Merkmale;};

   void free(void);  // gibt Speicher wieder frei
);

#endif
```



## Anhang A XII - kanten.h

```
//////////////////////////////////////////////////////////////////////
// Die folgende Klasse dient zur Speicherung aller Informationen   //
// ueber eine Kante. Daher besteht die Klasse vor allem aus zwei   //
// Strings, welche die Namen der angrenzenden Knoten anthalten und //
// drei Feldern fuer ungerichtete, abgehende bzw. ankommende Merk- //
// male. Bei Anfragen auf diese Klasse ist es oft wichtig, aus wel-//
// cher Sicht die Kante betrachtet wird. Stimmen initialisierungs- //
// Startort (von) und aktuelle Betrachtungsposition ueberein, sind //
// die Bezeichnungen der Merkmale als abgehend bzw. ankommend      //
// treffend. Ist die aktuelle Position das initialisierungsziel der//
// Kante, muessen die Felder fuer die gerichteten Kantenmerkmale   //
// getauscht werden.                                               //
//////////////////////////////////////////////////////////////////////
#ifndef KANTEN_H
#define KANTEN_H

#include "basismerkmal.h"

class kanten_struktur
{
   Basismerkmal **ungerichtet_M,   // Feld fuer Zeiger auf Merkmale
                **abgehend_M,      // abgehend von initStartOrt (von)
                **ankommend_M;     // abgehend von initZielOrt (nach)
   char *von,                      // initStartOrt
        *nach;                     // initZielOrt
   void initUngerMerkmal(int);     // initialisier ungerichtete MerkmalFelder
   void initGerMerkmal(int);       // initlaisiert gerichtetes MerkmalFeld
 public:
   kanten_struktur(void);          // Konstruktor, ruft init...Merkmal() auf
                                   // und richtet MerkmalFelder ein
   void setNames(char* a, char* b) // legt die initialisierungsNamen fest
     {von = a; nach = b;};
   char* wohin(char* a);           // gibt Ziel von a aus gesehen zurueck

 // ungerichtete Merkmale
   int setIntervall(int MerkmalPos, char* word1, char* word2)
     {return (*(ungerichtet_M+MerkmalPos))->addIntervall(word1, word2);};

   void setMerkmal(int MerkmalPos, char* word)
     {(*(ungerichtet_M+MerkmalPos))->addMerkmal(word);};

 // abgehendes Merkmal
   int setAbIntervall(char*, int, char*, char*);
   void setAbMerkmal(char*, int, char*);

 // ankommendes Merkmal
   int setAnIntervall(char*, int, char*, char*);
   void setAnMerkmal(char*, int, char*);

 // Zugriffsmethoden auf Merkmale
   Basismerkmal** getMerkmal(void)      // gibt ungerichtetes Merkmal zurueck
     { return ungerichtet_M ;};
   Basismerkmal** getAbMerkmal(char*); // gibt Abgehend von uebergebenen
Namen
   Basismerkmal** getAnMerkmal(char*); // gibt Ankommend an uebergebenen
Namen
                                       // zurueck
   void gibAusKante(ostream);
   void gibAusKante(void)
     {gibAusKante(cout);};

   void free(void);  // gibt Speicher frei

   friend void general(kanten_struktur*, char*,
                       kanten_struktur*, char*,
                       kanten_struktur*);
);

#endif
```

## Anhang A XIII - basismerkmal.h

```
//////////////////////////////////////////////////////////////////////
// Stellt die Merkmalsklasse zur Verfuegung, von welcher alle       //
// anderen Merkmale abgeleitet sind. Da diese Klasse abstrakte vir- //
// tuelle Methoden enthaelt, darf ihr kein Merkmal direkt zugeord-  //
// net sein. Demzufolge existiert lediglich fuer den Konstruktor    //
// eine Implementation.                                             //
//////////////////////////////////////////////////////////////////////
#ifndef BASISMERKMAL_H
#define BASISMERKMAL_H

#include <iostream.h>

class Basismerkmal
{
 public:
   virtual bool addIntervall(char* a, char* b)
     {
       // Intervall in Beschreibung fuer Merkmal ohne Intervalle
       cout << "\nWARNING: Intervall [" << a << "," << b
         << "] kann dem Merkmal nicht zugeordnet\n"
         << "          werden und wird ignoriert\n";
       return 1;
     };
   virtual void addMerkmal(char*) = 0;
   void gibAusMerkmal(void)
     {gibAusMerkmal(cout);};
   virtual void gibAusMerkmal(ostream) = 0;
   virtual ~Basismerkmal(void) = 0;
};

#endif
```



## Anhang A XIV - m_symbol.h

```
//////////////////////////////////////////////////////////////////////
// Merkmalsklasse fuer symbolische Merkmale                         //
//////////////////////////////////////////////////////////////////////
#ifndef M_SYMBOL_H
#define M_SYMBOL_H

#include "basismerkmal.h"
#include "menge.h"


//////////////////////////////////////////////////////////////////////
// Klasse zum Speichern von Symbolbezeichnern, wichtig fuer die     //
// Nutzung von ueberlagerten Operatoren fuer Vergleiche etc.        //
//////////////////////////////////////////////////////////////////////
class symbl
{
 public:
   char* Name;
};

//////////////////////////////////////////////////////////////////////
// Merkmalsklasse, enthaelt Menge mit Symbolen                      //
//////////////////////////////////////////////////////////////////////
class symbol_merkmal : public Basismerkmal
{
   menge_p<symbl*> daten;
 public:
   void addMerkmal(char*);          // Konstruktor
   void gibAusMerkmal(ostream);
   friend float cmp(symbol_merkmal*, symbol_merkmal*);
   friend float cmpObjKon(symbl*, symbol_merkmal*);
   friend float cmpKonKon(symbol_merkmal*, symbol_merkmal*);
   friend void general(symbol_merkmal*, symbol_merkmal*, symbol_merkmal*);
   ~symbol_merkmal(void);           // Destruktor
};

//////////////////////////////////////////////////////////////////////
// Vergleichs und Generalisierungsfunktionen                        //
//////////////////////////////////////////////////////////////////////
float cmp(symbol_merkmal*, symbol_merkmal*);
void general(symbol_merkmal*, symbol_merkmal*, symbol_merkmal*);

//////////////////////////////////////////////////////////////////////
// ueberladenen Operatoren                                          //
//////////////////////////////////////////////////////////////////////
int operator ==(class symbl a, class symbl b);
void operator <<(class ostream ausgabestrom, class symbl a);

#endif
```

## Anhang A XV - m_zahl.h

```
////////////////////////////////////////////////////////////////////////
// Klasse fuer Merkmale mit ganzen oder reellen Zahlen,               //
////////////////////////////////////////////////////////////////////////
#ifndef M_ZAHL_H
#define M_ZAHL_H

#include <iostream.h>
#include "merkstrukt.h"
#include "basismerkmal.h"
#include "basis_liste.h"


////////////////////////////////////////////////////////////////////////
// Enumerator fuer Vergleich und Generalisierung                      //
////////////////////////////////////////////////////////////////////////
enum mArt(statistisch, fuzzy);


////////////////////////////////////////////////////////////////////////
// Template Klasse, wird fuer reelle und ganze Zahlen uebersetzt.     //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
class merkmal : public Basismerkmal
{
   mArt art;
   univ_liste<AnyType> *daten;
 public:
   merkmal(int , para);        // Konstruktor
   bool addIntervall(char*, char*);
   bool addIntervall(AnyType, AnyType);
   void addMerkmal(char*);
   void addMerkmal(AnyType a) {daten->add(a);};
   void gibAusMerkmal(ostream);
   univ_liste<AnyType>* getListe(void) {return daten;};
   friend float cmpObjKon(AnyType, merkmal*, float);
   friend float cmpKonKon(merkmal<AnyType>*, merkmal<AnyType>*, float, int);
   ~merkmal(void);             // Destruktor
};


////////////////////////////////////////////////////////////////////////
// Vergleichs und Generalisierungsfunktionen                          //
////////////////////////////////////////////////////////////////////////
template <class AnyType>
float cmp(merkmal<AnyType>*, merkmal<AnyType>*, para);

template <class AnyType>
void general(merkmal<AnyType>*, merkmal<AnyType>*,
             merkmal<AnyType>*, para);

#endif
```



## Anhang A XVI - m_gmenge.h

```
////////////////////////////////////////////////////////////////////////
// Merkmalsklasse fur geordnete Mengen, alle bei basis_merkmal be-    //
// schriebenen Methoden sind hier implementiert.                     //
////////////////////////////////////////////////////////////////////////
#ifndef M_GMENGE_H
#define M_GMENGE_H

#include "gmenge.h"
#include "merkstrukt.h"
#include "basismerkmal.h"
#include "menge.h"
#include "m_zahl.h"


////////////////////////////////////////////////////////////////////////
// Sammlung geordneter Mengen, Hintergrundinformation fuer alle       //
// Merkmale                                                           //
////////////////////////////////////////////////////////////////////////
class geord_menge
{
   menge_p<gmenge*> GMengeListe;
 public:
   void addGMenge(para);
   gmenge* getGMenge(char*);
   void free(void);
   ~geord_menge(void);
};


////////////////////////////////////////////////////////////////////////
// Instanz der Sammlung geordneter Mengen wird hier zur Verfuegung    //
// gestellt                                                           //
////////////////////////////////////////////////////////////////////////
extern geord_menge GMengeSammlung;

////////////////////////////////////////////////////////////////////////
// eigentliches Merkmal, bestht hauptsaechlich aus einem Merkmal      //
// 'm_zahl' (*daten), welches den/die Rang/Raenge registriert.        //
////////////////////////////////////////////////////////////////////////
class gmenge_merkmal : public Basismerkmal
{
   merkmal<int> *daten;      // ganzzahliges Merkmal
   char* Name;               // Name der geordneten Menge
   para* mod(para);          // verkuerzt Parameter
   char* getBegriff(int);    // sucht Begriff fuer Rang
   int getRang(char*);       // sucht Rang von Begriff
 public:
   gmenge_merkmal(int, para);           // Konstruktor
   bool addIntervall(char*, char*);
   void addMerkmal(char*);
   void gibAusMerkmal(ostream);
   friend float cmp(gmenge_merkmal*, gmenge_merkmal*, para);
   friend void general(gmenge_merkmal*, gmenge_merkmal*,
                       gmenge_merkmal*, para);
   ~gmenge_merkmal(void);               // Destruktor
};


////////////////////////////////////////////////////////////////////////
// Verglecish und Generalisierungsfunktionen.                         //
////////////////////////////////////////////////////////////////////////
float cmp(gmenge_merkmal*, gmenge_merkmal*, para);
void general(gmenge_merkmal*, gmenge_merkmal*,
             gmenge_merkmal*, para);

#endif
```

## Anhang A XVII - gmenge.h

```
//////////////////////////////////////////////////////////////////////
// Diese beiden Klassen dienen zur Speicherung einer geordneten     //
// Liste. (Prototyp, gelesen aus einer Datei, nicht zum Speichern   //
// einzelner MerkmalsAuspraegungen)                                 //
//////////////////////////////////////////////////////////////////////
#ifndef GMENGE_H
#define GMENGE_H

#include "multiset_sort.h"


//////////////////////////////////////////////////////////////////////
// Klasse, welche immer einen Begriff mit dem zugeordneten Rang     //
// in der geordneten Menge (1 <= Rang <= Anzahl der Elemente)       //
//////////////////////////////////////////////////////////////////////
class gmenge_elem
{
 public:
   char* Begriff;  // aus Datei gelesener Name, muss am Ende durch
                   // free() geloescht werden
   int Rang;
   void free(void);
};


//////////////////////////////////////////////////////////////////////
// Die folgende Klasse enthaelt eine Liste von Elementen, und die   //
// zum effektiven Umgang mit der Klasse noetigen Methoden.          //
//////////////////////////////////////////////////////////////////////
class gmenge
{
   char* Name;                          // Name der geordneten Menge
   multiset_sort_p<gmenge_elem*> *GMengeListe;// Liste mit Elementen
 public:
   gmenge(char*);                       // Konstruktor (Parameter Name)
   int initGMenge(void);                // liest die gMenge aus der Datei

   char* getName(void) {return Name;};  // gebe gMengenName zurueck
   char* getBegriff(const int);         // suche Name zu Rang
   int getRang(const char*);            // suche Rang zu Name

   void gibAusGMenge(void);             // gebe geordnete Menge aus

   void free(void);                     // gib belegten Speicher frei
};


//////////////////////////////////////////////////////////////////////
// ueberlagerte Operatoren fuer Vergleich und Ausgabe               //
//////////////////////////////////////////////////////////////////////
int operator <=(class gmenge_elem a, class gmenge_elem b);
int operator <(class gmenge_elem a, class gmenge_elem b);
int operator ==(class gmenge_elem a, class gmenge_elem b);
int operator -(class gmenge_elem a, class gmenge_elem b);
void operator <<(class ostream ausgabestrom,
                 class gmenge_elem a);

int operator ==(class gmenge a, class gmenge b);
void operator <<(class ostream ausgabestrom, class gmenge a);

#endif
```



## Anhang A XVIII - m_hierarchie.h

```
//////////////////////////////////////////////////////////////////////
// Merkmalsklasse fuer Begriffe einer Hierarchie                    //
//////////////////////////////////////////////////////////////////////
#ifndef M_HIERARCHIE_H
#define M_HIERARCHIE_H

#include "basismerkmal.h"
#include "hierarchie.h"
#include "merkstrukt.h"
#include "menge_sort.h"

//////////////////////////////////////////////////////////////////////
// HierarchieSammlung, alle systemweiten Informationen ueber die    //
// verschiedenen Hierarchien koennen hier verwaltet werden.         //
//////////////////////////////////////////////////////////////////////
class begr_hier
{
   menge_sort_p<hierarchie*> HierListe;
 public:
   void addHier(para);
   hierarchie* getHier(char*);
   void free(void);
   ~begr_hier(void);
};

//////////////////////////////////////////////////////////////////////
// systemweite Instanz der Hierarchiesammlung                       //
//////////////////////////////////////////////////////////////////////
extern begr_hier HierSammlung;

//////////////////////////////////////////////////////////////////////
// Merkmalsklasse fuer Begriffe einer Hierarchie                    //
//////////////////////////////////////////////////////////////////////
class hier_merkmal : public Basismerkmal
{
   char* Hierarchie;        // Zeiger auf den Name der Hierarchie
   char* Begriff;           // Name des Begriffs (Auspraegung)
 public:
   hier_merkmal(para Par); // Konstruktor
   void addMerkmal(char*);
   void gibAusMerkmal(ostream out) {out << Begriff;};
   friend float cmp(hier_merkmal*, hier_merkmal*, para);
   friend void general(hier_merkmal*, hier_merkmal*,
                       hier_merkmal*, para);
   ~hier_merkmal(void);    // Destruktor
};

//////////////////////////////////////////////////////////////////////
// Vergleichs und Generalisierungsfunktionen                        //
//////////////////////////////////////////////////////////////////////
float cmp(hier_merkmal*, hier_merkmal*, para);
void general(hier_merkmal*, hier_merkmal*,
             hier_merkmal*, para);

#endif
```

## Anhang A XIX - hierarchie.h

```
////////////////////////////////////////////////////////////////////////
// Die folgenden Klassen dienen zur Speicherung einer aus einer     //
// Datei zu lesenden Begriffshierarchie. Jedes Element dieser       //
// Hierarchie wird (durch Zeiger) in zwei Listen verwaltet, einmal  //
// innerhalb der Gesamtstruktur (hierarchie) und zum zweiten in der //
// Hierarchiestufe, in welcher sich das Element befindet.           //
////////////////////////////////////////////////////////////////////////
#ifndef HIERARCHIE_H
#define HIERARCHIE_H

#include <string.h>
#include <iostream.h>
#include "file.h"
#include "multiset_sort.h"

////////////////////////////////////////////////////////////////////////
// Versuch der Verdeutlichung der Gesamtstruktur
//
// hier_stufe:[hier_elem][hier_elem][hier_elem]
//      ^           ^          ^          ^
//      |           |          |          |
//      |           |          |          v
//      +---------------------------hier_stufe:[hier_elem][hier_elem]
//      |           |          |                      ^          ^
//      |           |          v                      |          |
//      +-----------------hier_stufe:...              v          v
//
// jedes hier_elem besitzt Zeiger auf die untergeordneten
Begriffe(hier_stufe)
// und auf die eigene HierarchieStufe(hier_stufe)
// jede Hierarchiestufe besitzt Zeiger auf das uebergeordnete Element und
// auf die uebergeordnete Hierarchiestufe
// zusaetzlich werden Zeiger auf alle Elemente in der Klasse hierarchie
// gesammelt, da dies guenstiger zur Ausgabe und zur Suche ist

class hier_Stufe;
class hier_elem;


////////////////////////////////////////////////////////////////////////
// Klasse zur Speicherung einer aus einer Datei gelesenen Begriffs-//
// hierarchie                                                       //
////////////////////////////////////////////////////////////////////////
class hierarchie
{
   char* Name;          // Zeiger auf Name der Begriffshierarchie, Original
                          // wird weiterhin in merkmal_struktur gehalten
   int Groesse;         // Anzahl der Stufen der Hierarchie
   multiset_sort_p<hier_elem*> *BegriffsListe;  // Liste mit allen Elementen

   void h_readerror(const int,       // Methode zur Fehlerausgabe
                    readfile);
   void gibAusStufe(hier_elem*);     // gibt eine HierarchieStufe aus
   hier_elem* getElem(char*);        // sucht HierarchieElement anhand
                                     // des Namens
 public:
   hierarchie(char*);                // Konstruktor
   int initHierarchie(void);         // liest alle Informationen aus Datei
   char* getName(void)               // gebe HierarchieName
     {return Name;};
   int getGroesse(void)              // gebe HierarchieGroesse
     {return Groesse;};
   void gibAusHierarchie(void);      // gibt gesamte Hierarchie aus
   char* general(char*, char*);      // sucht Generalisierung
   int stufe(char*);                 // sucht Stufe eines Begriffs
   bool elem(char*);                 // testet, ob Begriff in Hierarchie
   void free(void);                  // gibt Speicherstruktur frei
};
```



```
////////////////////////////////////////////////////////////////////////
// Klasse, welche ein HierarchieElement mit Zeigern auf die eigene //
// und die untergeordnete HierarchieStufe enthaelt                  //
////////////////////////////////////////////////////////////////////////
class hier_elem
{
   hier_Stufe *down,    // alle Untergeordneten Begriffe, wichtig fuer
                        //  die Ausgabe der Begriffshierarchie
              *own;     // Zeiger auf eigene HierarchieStufe
   char* Name;          // Begriff Original aus Datei gelesen
 public:
   hier_elem(void)      // Konstruktor
     {down = NULL; Name = NULL;};

   char* getName(void) // gebe Begriff zurueck
     {return Name;};
   hier_Stufe* getDown(void) // gebe Zeiger auf untergeordnete Stufe zurueck
     {return down;};
   hier_Stufe* getOwn(void)  // gebe Zieger auf eigene HierarchieStufe
zurueck
     {return own;};

   void free(void);    // gibt belegten Speicher wieder frei

   friend hier_elem* hierarchie::elem(char*);
   friend int hierarchie::initHierarchie(void);
   friend void hierarchie::free(void);
};


////////////////////////////////////////////////////////////////////////
// HierarchieStufe, enthaelt eine Liste mit allen Elementen, welche//
// direkt einem Begriff der Hierarchie untergeordnet sind.          //
////////////////////////////////////////////////////////////////////////
class hier_Stufe
{
   hier_elem *up_elem;   // uebergeordneter Begriff
   hier_Stufe *up_Stufe; // uebergeordnete HierarchieStufe
   int Stufe;            // eigene HierarchieHoehe
   multiset_sort_p<hier_elem*>
     *BegriffsListe;     // Liste mit Elementen dieser Stufe
 public:
   hier_Stufe(hier_elem*, hier_Stufe*, int);  // Konstruktor

   hier_elem* getUpElem(void)    // gebe Zeiger auf Uebergeordnetes Element
     {return up_elem;};          // zurueck
   hier_Stufe* getUpStufe(void) // gebe Zieger auf die uebergeordnete
     {return up_Stufe;};         // HierarchieStufe zurueck
   void add(hier_elem* elem)     // fuege Element zur Stufe hinzu
     {BegriffsListe->add(elem);};
   int getStufe(void)            // gebe StufenHoehe zurueck
     {return Stufe;};
   void gibAusStufe(void);       // gebe HierarchieStufe aus

   void free(void);              // gibt Speicher wieder frei
};


////////////////////////////////////////////////////////////////////////
// ueberlagerte Operatoren fuer Ausgaben und Vergleiche             //
////////////////////////////////////////////////////////////////////////
int operator <=(class hier_elem a, class hier_elem b);
int operator <(class hier_elem a, class hier_elem b);
int operator ==(class hier_elem a, class hier_elem b);
int operator -(class hier_elem a, class hier_elem b);
void operator <<(class ostream ausgabestrom, class hier_elem a);

int operator <=(class hierarchie a, class hierarchie b);
int operator <(class hierarchie a, class hierarchie b);
int operator ==(class hierarchie a, class hierarchie b);
int operator -(class hierarchie a, class hierarchie b);
void operator <<(class ostream ausgabestrom, class hierarchie a);

#endif
```

## Anhang A XX - cmp.h

```
//////////////////////////////////////////////////////////////////////
// Diese beiden Funktionen dienen zum Vergleich von Knoten bzw.     //
// Kanten und rufen ihrerseits Funktionen auf, welche direkt        //
// den bestimmten Merkmalsauspraegungen zugeordnet sind. Daher      //
// muss die Funktion float cmp(Basis..., Basis...), welche den      //
// Aufruf der den Merkmalen zugeordneten Vergleichsfunktionen       //
// uebernimmt, bei Ergaenzung neuer Merkmale modifiziert werden.    //
//////////////////////////////////////////////////////////////////////


#ifndef CMP_H
#define CMP_H

#include "knoten.h"
#include "kanten.h"

float cmp(knoten_struktur*, knoten_struktur*);

float cmp(kanten_struktur*, char*,
        kanten_struktur*, char*);

#endif
```

## Anhang A XXI - general.h

```
//////////////////////////////////////////////////////////////////////
// Die folgenden Funktionen sind GeneralisierungsAufrufe und bilden//
// auf die gleiche Art wie bei cmp.cpp virtuelle friend functions   //
// fuer die Klasse Basismerkmal                                     //
//////////////////////////////////////////////////////////////////////
#ifndef GENERAL_H
#define GENERAL_H

#include "knoten.h"
#include "kanten.h"

void general(knoten_struktur*, knoten_struktur*, knoten_struktur*);

void general(kanten_struktur*, char*,
           kanten_struktur*, char*, kanten_struktur*);

#endif
```



## Anhang A XXII - komp_graph.h

```
//////////////////////////////////////////////////////////////////////
// Die folgenden Klassen dienen zur kompletten Beschreibung eines   //
// KompatibilitaetsGraphen. Sie enthalten alle zur Erstellung,      //
// Iteration und Ausgabe noetigen Methoden                          //
//////////////////////////////////////////////////////////////////////
#ifndef KOMP_GRAPH_H
#define KOMP_GRAPH_H

#include "knoten.h"
#include "graph.h"
#include "menge.h"
#include "multiset.h"


class komp_kante;
class excitat_kante;


//////////////////////////////////////////////////////////////////////
// Die Klasse komp_obj stellt einen Knoten des Kompatibilitaets-    //
// Graphen dar.                                                     //
//////////////////////////////////////////////////////////////////////
class komp_obj
{
 public:
   komp_obj(void);            // Konstruktor
   knoten_struktur *KnotenA, // Zeiger auf Knoten im Graph A
                   *KnotenB; // Zeiger auf Knoten im Graph B
   float Aktivitaet[2];       // Feld fuer aktuelle und letzte Aktivitaet
   float Aehnlichkeit;        // enthealt VergleichsWert beider Knoten
   bool Aenderbar;            // wenn Aenderbar = 0, bleibt Wert konstant
   float I_;                  // externer Input
   multiset_p<komp_kante*>    *inhibit;  // Liste mit inhibitorischen Kanten
   multiset_p<excitat_kante*> *excitat;  // Liste mit excitatorischen Kanten
   void free();               // gibt den durch die Listen belegten Speicher
                              // frei
};


//////////////////////////////////////////////////////////////////////
// Die Klasse stellt KompatibilitaetsKanten                         //
//////////////////////////////////////////////////////////////////////
class komp_kante
{
   komp_obj *ObjektA,    // Zeiger auf KompatibObjekt A
            *ObjektB;    // Zeiger auf KompatibObjekt B
 public:
   komp_kante(komp_obj *a, komp_obj *b)  // Konstruktor
     { ObjektA = a; ObjektB = b; };
   komp_obj *Objekt(komp_obj *MatchObj)  // gibt anderes Objekt zurueck
     {
       if (MatchObj == ObjektA) return ObjektB;
       if (MatchObj == ObjektB) return ObjektA;
       else { cout << "ERROR: komp_kante Objekt() fail\n";
              exit(1); };
     };
   friend void operator <<(class ostream ausgabestrom,
                           class komp_kante a);
};


//////////////////////////////////////////////////////////////////////
// Excitatorische Kanten werden durch die folgende Klasse von       //
// komp_kanten abgeleitet und um einige Parameter erweitert.        //
//////////////////////////////////////////////////////////////////////
class excitat_kante : public komp_kante
{
 public:
   excitat_kante(komp_obj *a, komp_obj *b)  // Konstruktor
     : komp_kante(a,b) { };
   float Aehnlichkeit;   // enthaelt VergleichsWert beider Kanten oder 0
```

```
   float w_ij_;          // Gewicht der Kante
};


//////////////////////////////////////////////////////////////////////
// Die Klasse kompatib_graph besteht im wesentlichen aus einer Lis-//
// te von komp_obj(-ekten, also Knoten), einer Liste mit komp_kan- //
// ten, und den fuer jede Iteration noetigen BerechnungsGewichten. //
// Desweiteren enthaelt diese Klasse alle Methoden zur Eingabe,    //
// Iteration und Ausgabe des KompatibilitaetsGraphen               //
//////////////////////////////////////////////////////////////////////
class kompatib_graph
{
 protected:
   menge_p<komp_obj*> komp_liste;             // Liste mit Knoten
   multiset_p<komp_kante*> komp_kanten_liste;  // Liste mit Kanten
   float aktKnotenSchwelle, aktKantenSchwelle; // aktuelle Schwellen
   int  KnotenAnzahl;                 // Anzahl der Knoten
   int  KantenAnzahl;                 // Anzahl der Kanten
   int  minGraphKnotenAnzahl,         // min. Anzahl der Knoten in den Graphen
        maxGraphKnotenAnzahl,         // max. Anzahl der Knoten in den Graphen
        GraphAKnotenAnzahl,           // Anzahl der Knoten in den Graphen
        GraphBKnotenAnzahl;
   void addKompObj(knoten_struktur*,  // fuegt neues komp_obj zum Graph hinzu
                knoten_struktur*, float);
   // Methoden, welche zwischen zwei Objekten eine Kante einfuegen
   void addInhibit(komp_obj*, komp_obj*);
   void addExcitat(komp_obj*, komp_obj*, float);
   char* newName(komp_obj *);         // bildet aus zwei GraphKnoten neuen Name
   void initKompatGraph(graph *a, // bilde aus zwei Graphen Kompatib-Graph
                        graph *b, float*, float*);
 public:
   void free(void);
   kompatib_graph(void);              // Konstruktor
   char* newName(char*, char*);       // bildet aus zwei Namen einen neuen
   void initKompatGraph(graph *a, // bilde aus zwei Graphen Kompatib-Graph
                        graph *b);
   void gibAusGraph(void);            // gibt Komp-Graph auf Bildschirm aus
   graph* getErgGraph(void);          // erzeugt aus Ergebnis neuen Graphen
   ~kompatib_graph(void);             // Destruktor
};



// ueberladene Operatoren fuer die Ein- und Ausgabe
void operator <<(class ostream ausgabestrom, class komp_obj a);
bool operator ==(class komp_obj a, class komp_obj b);

void operator <<(class ostream ausgabestrom,
                 class komp_kante a);

#endif
```



## Anhang A XXIII - wta_params.h

```
//////////////////////////////////////////////////////////////////////
// Klasse fuer die Registrierung der Netzparameter                  //
//////////////////////////////////////////////////////////////////////
#ifndef WTA_PARAMS_H
#define WTA_PARAMS_H

#include <iostream.h>
#include "menge.h"
#include "menge_sort.h"


//////////////////////////////////////////////////////////////////////
// Definition der Enumeratoren, ebenfalls in 'wta_work.h' vorhanden//
//////////////////////////////////////////////////////////////////////
#ifndef WTA_PARAMS_ENUM
#define WTA_PARAMS_ENUM
enum s_art{lose,fixiert,reduziert};
enum s_was{knot,kant,beide};
enum anf_art{interaktiv,graphGroesse,fest,aehnlichkeit,listenwerte};
enum exI_art{ohne,prozentGraphGroesse,anzahlKanten};
enum kanSk_art{keine,kanten,kanten_u_knoten};
#endif


//////////////////////////////////////////////////////////////////////
// Listenelement fuer die Speicherung einer speziellen Anfangs-     //
// Aktivierung                                                      //
//////////////////////////////////////////////////////////////////////
class AnfAktListElem
{
 public:
   char *Name1,       // Name des Knotens aus dem ersten Graphen
        *Name2;       // Name des Knotens aus dem zweiten Graphen
   float Wert;        // Wert der Anfangsaktivierung
};


//////////////////////////////////////////////////////////////////////
// Klasse fuer die Netzparameter und die Vergleichsguete            //
//////////////////////////////////////////////////////////////////////
class wta_params
{
 protected:
   // Vergleichsergebnis -- Zeiger (dann auf Feld mit 3 Werten)
   float *Guete;
   // Parameter
   enum anf_art anfAktArt;          // Anfangsaktivierungsart
   menge_p<AnfAktListElem*> AnfAktListe; // Liste mit Anfangsaktivierungen
   float init;                      // Wert der Standard- AnfAkt
   enum exI_art extInpArt;          // Art des externen Inputs
   float skal;                      // skal- Wert fuer ext. Input
   enum kanSk_art kanSkalArt;       // Art der Kantenskalierung
   enum s_art schrittArt;           // Art des schrittweisen Matchings
   enum s_was schrittWas;           // Was wird schrittweise gematcht
   int AnzBerech;                   // bei wieviel Berechnungen
   float s;                         // Selbsthemmung
   float Genauigkeit;
   menge_sort<float> KnotenSchwelle, // Liste mit Knotenschwellen
                     KantenSchwelle; // Liste mit Kantenschwellen
 public:
   wta_params(void);                // Konstruktor
   wta_params* getParams(void);     // bildet eigene Belegung auf neue Klasse ab
   float getGuete(int);             // gibt Guete zurueck (0 bis 2)
   // Ein- / Ausgabemehoden
   void gibAusParams(void);         // zeigt Parameter auf Bildschirm an
   void gibAusParams(ostream);      // gibt Parameter nach ostream aus
   void saveParams(char*);          // speichert Parameter
   bool readParams(char*);          // liest Parameter ein
```

```
 // Einstellung der Parameter
   void setAnfAktArt(enum anf_art Art) // setzt Anfangsaktivierungsart
     {anfAktArt = Art;
       if (Art != listenwerte) resetAnfAktListe();};
   void resetAnfAktListe(void);        // leert Liste der
Anfangsaktivierungen
   bool addAnfAkt(char*,char*,float);  // fuegt AnfAkt zu Liste hinzu
   void gibAusAnfAktListe(void)        // gibt Liste mit Anfangsakt. aus
     {AnfAktListe.gibAusListe();};
   void setInit(float Wert);           // setzt init fuer Anfangsaktivierung
   void setExtInpArt(enum exI_art Art);// setzt Art des externen Inputs
   void setSkal(float Wert);           // setzt skal- Faktor fuer ext. Inp.
   void setKanSkalArt(enum kanSk_art Art) // setzt Art der KantenSkalierung
     {kanSkalArt = Art;};
   void setSchrittArt(enum s_art Art)  // setzt Art des schrittweisen Matchs
     {schrittArt = Art;};
   void setSchrittWas(enum s_was WAS)  // was wird schrittweise gematcht
     {schrittWas = WAS;};
   void setAnzBerech(int Wert)         // legt Anzahl der Berechnungen fest
     { AnzBerech = Wert;};
   void setS(float Wert) { s = Wert ;}; // stellt Selbsthemmung ein
   void setKnotenSchwelle(float Wert); // setzt Knotenschwelle
   void setKantenSchwelle(float Wert); // setzt Kantenschwelle
   void addKnotenSchwelle(float Wert); // fuegt Knotenschwelle hinzu
   void addKantenSchwelle(float Wert); // fuegt Kantenschwelle hinzu
};


////////////////////////////////////////////////////////////////////
// ueberladene Operatoren fuer Azsgaben und Vergleiche            //
////////////////////////////////////////////////////////////////////
void operator <<(class ostream ausgabestrom, class AnfAktListElem a);
bool operator ==(class AnfAktListElem a, class AnfAktListElem b);

#endif
```



## Anhang A XXIV - wta_init.h

```
////////////////////////////////////////////////////////////////////
// Initialisierungsmethoden fuer die Neuronen des WTA- Netzes     //
////////////////////////////////////////////////////////////////////
#ifndef WTA_INIT_H
#define WTA_INIT_H

#include <iostream.h>
#include "wta_params.h"
#include "komp_graph.h"

class wta_init : public wta_params,
                 public kompatib_graph
{
   void initAnfAkt(void);          // initialisiert Anfangsaktivierung
 protected:
   void init_I_w(void);            // initialisiert ext.Input und
                                   // Kantenskalierung
   void initAll(void);             // beide Initialisierungen
};

#endif
```

## Anhang A XXV - wta_iterate.h

```
////////////////////////////////////////////////////////////////////
// Alle Methoden zur Propagation sind in dieser Klasse verfuegbar //
////////////////////////////////////////////////////////////////////
#ifndef WTA_ITERATE_H
#define WTA_ITERATE_H

#include "wta_init.h"

class wta_iterate : public wta_init
{
   void updateNet(float*,int*);    // ermittle neue Potentiale
   void changeEnvironment(int*,int*,float*,float*);  // aendere Umgebung
   void newEnergyZmax(komp_obj*,float*,float,float); // berechnet Zmax
   float Anzahl_excitat_Kanten(void); // bestimmt Gewicht der excitat. Kanten
   void restoreErg(void);          // stellt ErgebnisPotentiale wieder her
 protected:
   float w,                        // Gewicht der excitatorischen Kanten
         r,                        // Gewicht der eigenen Aktivierung
         EPS,                      // Epsilon - Umgebung
         energy[2];                // Energie (aktuelle und alte)
   float f(float);                 // nichtlineare Funktion
   int iterate(int, float);        // iteriere Berechnungen
 public:
   wta_iterate(void);                 // Konstruktor
};


#endif
```

## Anhang A XXVI - wta_netz.h

```
//////////////////////////////////////////////////////////////////////
// Klasse mit Zugriffsmethoden und Verwaltungsmethoden zum WTA-Netz//
//////////////////////////////////////////////////////////////////////
#ifndef WTA_NETZ_H
#define WTA_NETZ_H

#include "graph.h"
#include "wta_iterate.h"

//////////////////////////////////////////////////////////////////////
// extern verfuegbare Versionsnummer des Programms                  //
//////////////////////////////////////////////////////////////////////
extern char* version;

class wta_netz : public wta_iterate
{
   void calcGuete(void);      // berechnet VergleichsGuete
   int fixiereErgebnis(void); // bearbeite ergebnis bei schrittweisem Match
 public:
   int InitIterate(graph *a,  // ruft IntialisierungsMethode
                   graph *b);//  und Iteration auf
   void gibAusErgebnis(void); // zeigt Neuronenaktivitaet an
};

#endif
```



## Anhang A XXVII - wta_work.h

```
//////////////////////////////////////////////////////////////////////
// Schnittstelle fuer die Nutzung des WTA- Netzes                   //
//////////////////////////////////////////////////////////////////////
#ifndef WTA_WORK_H
#define WTA_WORK_H


#ifndef WTA_PARAMS_ENUM
#define WTA_PARAMS_ENUM
enum s_art{lose,fixiert,reduziert};
enum s_was{knot,kant,beide};
enum anf_art{interaktiv,graphGroesse,fest,aehnlichkeit,listenwerte};
enum exI_art{ohne,prozentGraphGroesse,anzahlKanten};
enum kanSk_art{keine,kanten,kanten_u_knoten};
#endif

extern class wta_work work;

class wta_work
{
   class wta_netz *Netz;
   class graph *graphA, *graphB, *ErgGraph;
 public:
   wta_work(void);

// Ein-/ Ausgabemethoden
   bool leseStruktur(char*);
   bool leseGraphA(char*);
   bool leseGraphB(char*);
   int compare(void);
   void speicherErgebnisGraph(char*);

// Kontrollmethoden
   void gibAusStruktur(void);
   void gibAusGraphA(void);
   void gibAusGraphB(void);
   void gibAusErgGraph(void);
   void gibAusParams(void);
   void gibAusAnfAktListe(void);

// Methoden zur Parameteraenderung
   void setAnfAktArt(enum anf_art);
   void setInit(float);
   void resetAnfAktListe(void);
   bool addAnfAkt(char*,char*,float);
   void setExtInpArt(enum exI_art);
   void setSkal(float);
   void setKanSkalArt(enum kanSk_art);
   void setSchrittArt(enum s_art);
   void setSchrittWas(enum s_was);
   void setAnzBerech(int);
   void setS(float);
   void setKnotenSchwelle(float);
   void addKnotenSchwelle(float);
   void setKantenSchwelle(float);
   void addKantenSchwelle(float);

   ~wta_work(void);
);

#endif
```

## Anhang A XXVIII - menu.h

```
////////////////////////////////////////////////////////////////////////
// Diese Klasse verwaltete Aufbau und bafrage eines Menu's          //
////////////////////////////////////////////////////////////////////////
#ifndef MENU_H
#define MENU_H

class menu
{
   char **daten;              // Feld mit Namen der Menupunkte
   char *kurz;                // Kurzbezeichner aller Punkte
   char *Titel;               // Menutitel
   void (*TitelAusgabe) (void); // Zieger auf Menutitelfunktion
   int aktPos;                // aktuelle Position im Menu
   int maxLaenge;             // laengster Menupunkt
   int PunktAnzahl;           // Anzahl der Punkte
   bool testCursor(char);
   void gibAusZeile(int);
   void moveTo(int);
 public:
   menu(int);                 // Konstruktor
   void gibAusMenu(void);     // zeigt Menu an
   void addTitel(void (*Function) (void))  // fuegt Titelfunktion hinzu
     { TitelAusgabe = Function;};
   void addTitel(char* Name);  // fuegt Titelname hinzu
   void addPoint(char*,char);  // fuegt Menupunkt hinzu
   char auswahl(int);         // Auswahlmethode
   ~menu(void);               // Destruktor
};

#endif
```



## Anhang A XXIX - metaclass.h

```
////////////////////////////////////////////////////////////////////////
// Hier wird eine Metaklasse fuer die Bedienoberflaeche in den      //
// verschiedenen Klassen implementiert                              //
////////////////////////////////////////////////////////////////////////
#ifndef METACLASS_H
#define METACLASS_H

#include "graph.h"
#include "merkstrukt.h"
#include "menge_sort.h"
#include "wta_netz.h"


////////////////////////////////////////////////////////////////////////
// Die Merkmalsstruktur ist extern definiert (in main.cpp)          //
////////////////////////////////////////////////////////////////////////
extern merkmal_struktur *merkstruct;


////////////////////////////////////////////////////////////////////////
// generelle Datei Eingabe                                          //
////////////////////////////////////////////////////////////////////////
class sys_file_io
{
 protected:
   char* getOutFileName(char*);      // prueft AusgabeDateiName
   char* newFileName(void);          // erfragt neuen Dateinamen
};


////////////////////////////////////////////////////////////////////////
// Struktur, in welcher Graphen im System verwaltet werden, also    //
// allen Graphen sind zusaetzlich noch ihr Name, ihr Dateiname und  //
// falls moeglich die Netzparameter des WTA- Netzes zugeordnet      //
////////////////////////////////////////////////////////////////////////
class graph_struct
{
 public:
   graph_struct(void);
   graph  *Graph;                    // eigentlicher Graph
   char   *Name;                     // Graphbezeichner im System
   char   *DateiName;                // Name der Graphdatei
   wta_params *wtaParam;             // Zeiger auf Parameter
   void getFileName(char*, char*);   // erfragt Name und Dateiname
   void free(void);
};


////////////////////////////////////////////////////////////////////////
// Metaklasse, enthaelt einige komplexere Bedienablaeufe            //
////////////////////////////////////////////////////////////////////////
class metaclass : public sys_file_io
{
   graph_struct* getGraph(char*);    // sucht Graph in graph_liste
   bool newCompareGraph(void);       // stellt Vergleichsgraphen ein
 protected:
   wta_netz *Netz;                   // WTA- Netz
   graph_struct *graphA, *graphB;    // Zeiger auf die Vergleichsgraphen
   menge_sort_p<graph_struct*> graph_liste; // Liste mit Graphstrukturen
   int maxGNameL;                    // Groesse des laengsten Graphnamen
 public:
   bool merkstrukt_init;             // MerkmalsStruktur eingelesen ?
   metaclass(void);                  // Konstruktor
   void zeigeGraphen(void);          // zeigt alle Graphnamen an
   int getGraphStatus(void)          // Info ueber Anzahl der Graphen
     { return graph_liste.getStatus();};

   void getStruct(void);             // Markierungsstruktur einlesen
   void gibAusStruct(void);          // Markierungsstruktur anzeigen

   void getGraph(void);              // Graph einlesen
```

```
   void zeigeGraph(void);              // Graph anzeigen
   void gibAusGraph(void);             // Graph ausgeben
   void killGraph(void);               // Graph entfernen

   void bildeKGraph(void);             // Kompatibilitaetsgraph bilden
   void zeigeKGraph(void)
     { if (Netz != NULL) Netz->gibAusGraph();
       else cout << "noch kein KompGraph erstellt...\n";};
   void zeigeKGraphKurz(void)
     { if (Netz != NULL) Netz->gibAusErgebnis();
       else cout << "noch kein KompGraph erstellt...\n";};

   void compare(void);                 // Graphen vergleichen
   void MakroEinlesen(void);           // Makro einlesen

   ~metaclass(void);                   // Destruktor
};

//////////////////////////////////////////////////////////////////////
// ueberladene Operatoren fuer AUsgabe und Vergleich                //
//////////////////////////////////////////////////////////////////////
int operator <=(class graph_struct a, class graph_struct b);
int operator <(class graph_struct a, class graph_struct b);
int operator ==(class graph_struct a, class graph_struct b);
int operator -(class graph_struct a, class graph_struct b);
void operator <<(class ostream ausgabestrom, class graph_struct a);

#endif
```



## Anhang A XXX - metawta.h

```
//////////////////////////////////////////////////////////////////////
// Metaklasse, welche Menus zur Aenderung der Netzparameter         //
// verwaltet                                                        //
//////////////////////////////////////////////////////////////////////
#ifndef METAWTA_H
#define METAWTA_H

#include "metaclass.h"

class meta_wta : public metaclass   // Erweiterung von 'metaclass'
{
   char changeSchritt(void);        // schrittweises Matching
   char changeAnfAkt(void);         // Anfangsaktivierung
   char changeExtInp(void);         // externer Input
   char changeKanSkal(void);        // KantenSkalierung
   void leseParams(void);           // liest Netzparameter
   void schreibeParams(void);       // schreibt Netzparameter
 public:
   void changeDef(void);            // aendert Netzparameter
   friend void gibAusParameter(void);
};


#endif
```

## Anhang A XXXI - matrix.h

```
////////////////////////////////////////////////////////////////////
// Zur Bildung von Deskriptoren kann es sinnvoll sein, bestimmte  //
// Graphen alle mit allen zu vergleichen und die Ergebnisse in    //
// Matrix abzulegen. Dier folgende Klasse unterstuetzt dieses     //
// Vorgehen und verwaltet die aufeinanderfolgenden Berechnungsauf- //
// rufe. Da derartige Berechnungen sehr lange dauern, werden die  //
// Daten bei der Berechnung in einer Datei gespeichert und erst   //
// spaeter zur Ausgabe als Matrix zusammengesetzt.                //
////////////////////////////////////////////////////////////////////
#ifndef MATRIX_H
#define MATRIX_H

#include <fstream.h>
#include <iostream.h>
#include "metawta.h"
#include "graph.h"
#include "multiset.h"

extern char* Matrix;  // Name der SystemDatei muss exportiert werden,
                      // damit andere Klassen die Existenz pruefen koennen

void status_infos(void);

////////////////////////////////////////////////////////////////////
// Die folgende Struktur dient als Element, welches in der Liste  //
// aller zu vergleichenden Graphen gehalten wird                  //
////////////////////////////////////////////////////////////////////
class matr_struct
{
 public:
   char *Name;         // GraphName (nach dem wird im System gesucht)
   int Pos;            // Position, damit Matrix rekonstruiert werden kann
};


////////////////////////////////////////////////////////////////////
// Die eigentliche Klasse enthaelt eine Liste der zu Vergleichenden /
// Graphen, einige Informationen ueber die Berechnungsbedingungen //
// und ein Feld fuer die Bildung der Matrix                       //
////////////////////////////////////////////////////////////////////
class matrix : public meta_wta    // ist als Erweiterung der Ein-/
                                  // Ausgabe Metaklassen gedacht
{
   float ***Guete;        // Feld zur Pufferung der Ergebnisdaten
                          // beim Einlesen vor der Ausgabe
   char *Daten;           // DateiName, in der die Berechnungsergebnisse
                          // gepuffert werden
   int prozess_anz,       // Anzahl der Prozesse, welche die Matrix berechnen
       graph_anz,         // Anzahl der Graphen in der Matrix
       berechn_anz;       // Anzahl der Berechnungen fuer jeden Prozess
   multiset_p<matr_struct*> matr_liste;  // Liste mit GraphNamen und Pos
   bool leseGraphen(char*,bool);  // liest alle Grpahen ein
   void berechneMatrix(int);      // berechnet MatrixDaten
   void gibAusMatrix(ostream);    // gibt Matrix aus
   bool verteilen();              // fragt, ob Berechnungen auf mehrere
                                  // Rechner verteilt werden sollen
   bool sucheNextNewDatei(int*);  // sucht die naechste noch nicht vorhandene
                                  // Datei
   bool sucheNextDatei_noEnd(int*);  // sucht die naechste vorhandene, aber
noch
                                     // noch nicht beendete Datei
   void newGuete(int);            // loescht altes GueteFeld und erstellt
neues
   bool readMainData(void);       // liest MatrixSystemDaten
   bool warning(void);            // gibt Warnung aus
   bool readStruct(void);         // liest StrukturBeschreibung ein
   bool leseMatrix(void);         // liest MatrixDateien ein
 public:
   matrix(void);
```



```
// StandardMethoden
  bool bildeMatrix(bool);      // koordiniert alle Methoden zur Bildung
                               // einer Matrix
  void gibAusMatrix(void);     // gibt Matrix auf Bildschirm aus
  void speichereMatrix(void); // gibt Matrix in datei aus
// ExpertInnenMethoden
  bool new_prozess(void);      // prueft und startet neuen Prozess
  void vervollstaendige(void);// prueft und vervollstaendigt Dateien
  void status_infos(void);
// Methoden fuer HintergrundBetrieb
  void new_exit(void)          // wiederholt new_prozess,
    {                          // bis letzter Prozess gerechnet
      for(;new_prozess(););
      exit(1);
    };
// Methoden zur Freigabe des Speichers
  void free(void);
  ~matrix(void);
};

// Ueberlagerter AusgabeOperatior fuer multiset_p<matr_struct*>
void operator <<(class ostream ausgabestrom, class matr_struct a);

#endif
```

## Anhang A XXXII - main.cpp

```
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
#include "matrix.h"
#include "menu.h"
#include "terminal.h"
#include <unistd.h>

////////////////////////////////////////////////////////////////////////
// systemweite Markierungsstruktur, wird ausserhalb einer Klasse      //
// verwaltet, da jeweils nur EINE Markierunsgstruktur nutzbar ist     //
////////////////////////////////////////////////////////////////////////
merkmal_struktur *merkstruct;


////////////////////////////////////////////////////////////////////////
// Schalter, welche Programmausgaben, jedoch (im wesentlichen)        //
// nicht den Programmablauf beeinflussen (fuer Entwicklung etc.)      //
////////////////////////////////////////////////////////////////////////
bool v = 0; // dokumentierte Ausgabe
bool e = 0; // FehlerSuche
bool t = 0; // Trace
bool tt = 0; // Trace privat-Methoden
bool d = 0; // Destruktor- Ausgabe


////////////////////////////////////////////////////////////////////////
// Instanz der Metaklasse                                             //
////////////////////////////////////////////////////////////////////////
matrix *basis = new matrix;


////////////////////////////////////////////////////////////////////////
// Matrix- Bildungs Menu.                                             //
////////////////////////////////////////////////////////////////////////
void matrix(void)
{
   char data;
   menu matrix(16);
   matrix.addTitel("Matrix Berechnung");
   matrix.addTitel(status_infos);
   matrix.addPoint("Matrix",0);
   matrix.addPoint("bilden",'b');
   matrix.addPoint("anzeigen",'a');
   matrix.addPoint("speichern",'s');
   matrix.addPoint("",0);
   matrix.addPoint("erweiterte Modi",0);
   matrix.addPoint("verteilt starten",'v');
   matrix.addPoint("neuen Prozess",'p');
   matrix.addPoint("Daten vervollstaendigen",'d');
   matrix.addPoint("Status-Informationen",'i');
   matrix.addPoint("",0);
   matrix.addPoint("fortlaufende Berechnung",0);
   matrix.addPoint("verteilt starten",'1');
   matrix.addPoint("neue Prozesse",'2');
   matrix.addPoint("",0);
   matrix.addPoint("Main Menu",'m');
   do
     {
      term.cls();
      matrix.gibAusMenu();
      data = matrix.auswahl(17);
      if (data != 'm') matrix.gibAusMenu();
      switch(data)
        {
         case 'b':
           basis->bildeMatrix(0);
           break;
         case 'a':
           basis->gibAusMatrix();
           break;
```



```
         case 's':
           basis->speichereMatrix();
           break;
         case 'v':
           basis->bildeMatrix(1);
           break;
         case 'p':
           basis->new_prozess();
           break;
         case 'd':
           basis->vervollstaendige();
           break;
         case 'i':
           basis->status_infos();
           break;
         case '1':
           if (basis->bildeMatrix(1)) basis->new_exit();
           break;
         case '2':
           basis->new_exit();
           break;
        };
      if (data != 'm') term.wait();
     }
   while (data != 'm');
}


////////////////////////////////////////////////////////////////////////
// Funktion, welche neues Hauptmenu darstellt                         //
////////////////////////////////////////////////////////////////////////
void neumenu(menu *main)
{
   term.cls();
   main->gibAusMenu();
}


////////////////////////////////////////////////////////////////////////
// Funktion fuer Statusinformationen im Hauptmenu                     //
////////////////////////////////////////////////////////////////////////
void main_titel(void)
{
   basis->zeigeGraphen();
}


////////////////////////////////////////////////////////////////////////
// Hauptprogramm, stellt Hauptmenu dar.                               //
////////////////////////////////////////////////////////////////////////
main()
{
   char data;
   menu *main = new menu(20);
   main->addTitel("Main Menu");
   main->addTitel(main_titel);
   main->addPoint("Markierungsbeschreibung",0);
   main->addPoint("einlesen",'s');
   main->addPoint("anzeigen",'v');
   main->addPoint("ausgeben",'o');
   main->addPoint("Graphbeschreibung",0);
   main->addPoint("einlesen",'g');
   main->addPoint("anzeigen",'z');
   main->addPoint("ausgeben",'a');
   main->addPoint("loeschen",'l');
   main->addPoint("KompatibilitaetsGraph",0);
   main->addPoint("bilden",'k');
   main->addPoint("anzeigen (lang)",'u');
   main->addPoint("anzeigen (kurz)",'t');
   main->addPoint("",0);
   main->addPoint("Netz- Parameter aendern",'n');
   main->addPoint("Graphen vergleichen", 'c');
   main->addPoint("Matrix berechnen",'x');
   main->addPoint("",0);
```

```
main->addPoint("Makro einlesen",'m');
main->addPoint("ProgrammEnde", 'e');
neumenu(main);
do
  {
   data = main->auswahl(21);
   neumenu(main);
   switch(data)
     {
      case 's':
        basis->getStruct();
        break;
      case 'v':
        if (!basis->merkstrukt_init) break;
        cout << "\nGesamtStruktur aller Graphen:\n\n";
        merkstruct->gibAusStruktur();
        term.wait();
        break;
      case 'o':
        basis->gibAusStruct();
        break;
      case 'g':
        basis->getGraph();
        break;
      case 'z':
        basis->zeigeGraph();
        term.wait();
        break;
      case 'a':
        basis->gibAusGraph();
        term.wait();
        break;
      case 'l':
        basis->killGraph();
        break;
      case 'k':
        basis->bildeKGraph();
        break;
      case 'u':
        basis->zeigeKGraph();
        term.wait();
        break;
      case 't':
        basis->zeigeKGraphKurz();
        term.wait();
        break;
      case 'n':
        basis->changeDef();
        break;
      case 'c':
        basis->compare();
        term.wait();
        break;
      case 'x':
        if ((!access(Matrix,F_OK)) || (basis->merkstrukt_init))
          matrix();
        else
          {
            cout << "Bitte erst MerkmalsStruktur einlesen" << endl;
            term.wait();
          }
        break;
      case 'm':
        basis->MakroEinlesen();
        break;
      case 'e':
        if (basis->getGraphStatus())
          {
            cout << "### Programmende, alle Graphen gehen verloren...\n"
              << "###    E - ProgrammEnde    A - Abbruch, gehe zurueck    ";
            data = term.getChar();
          }
     };
   neumenu(main);
```



```
   }
  while((data != 'e') && (data != 'E'));
  delete basis;
}
```

# Anhang B Graphbeschreibungsdateien

## Anhang B I - Graphbeschreibung zu Abbildung 9-3

```
#*********************************************************#
# WTA- Netz mit Beruecksichtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                     #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:          *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 1
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0 0.333333 0

*********************************************************#
# Graph hat folgende Strukturbeschreibung:                #
Knoten:
symbol,
reell_2(0.5,2)
Kante:
reell_2(0.1,2)

#*********************************************************#
1_2^2_2:Block,0

#*********************************************************#
```

## Anhang B II - Graphbeschreibung zu Abbildung 9-4

```
#*********************************************************#
# WTA- Netz mit Beruecksichtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                     #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:          *
keine Selbsthemmung, ( d = 0 )

schrittweises Matching (lose)
AehnlichkeitsSchwelle(n) fuer Knoten : {0,0.5,1)
AehnlichkeitsSchwelle(n) fuer Kanten : {0,0.5,1)

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.333333 0.5 0.166667

*********************************************************#
# Graph hat folgende Strukturbeschreibung:                #
Knoten:
symbol,
reell_2(0.5,2)
```



```
Kante:
reell_2(0.1,2)

#*********************************************************#
1_2^2_2:Block,0
1_3^2_1:Kreis,(2,3)
1_3^2_1,1_2^2_2:(3,4)

#*********************************************************#
```

## Anhang B III - Hierarchiebeschreibung zu Abbildung 9-5

```
Atom:Atom1,Atom2
Atom1:c,at3,at4
at3:n,p
at4:s,o
Atom2:h,halogen
halogen:f,cl,br,i
```

## Anhang B IV - Graphbeschreibung zu Abbildung 9-6 (f4)

siehe Anhang B XXI

## Anhang B V - Graphbeschreibung zu Abbildung 9-7 (f5)

siehe Anhang B XXII

## Anhang B VI - Graphbeschreibung zu Abbildung 9-8

```
#*********************************************************#
# WTA- Netz mit Beruecksichtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                     #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:          *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 1
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.0466667 0.600299 0.0466667

*********************************************************#
# Graph hat folgende Strukturbeschreibung:                #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_17:o,40,{[-0.393,-0.354]}
f4_10^f5_18:o,40,{[-0.393,-0.354]}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
f4_18^f5_10:c,14,{[0.608,0.646]}
```

```
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_14:h,3,{[0.057,0.096]}
f4_21^f5_15:h,3,{[0.057,0.096]}
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_4^f5_4,f4_3^f5_3:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_17,f4_8^f5_16:2
f4_10^f5_18,f4_8^f5_16:2
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_14,f4_17^f5_8:1
f4_21^f5_15,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1

#*********************************************************#
```

## Anhang B VII - Graphbeschreibung zu Abbildung 9-9

```
#*********************************************************#
# WTA- Netz mit Berueckschtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                   #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:        *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0.6
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.0533333 0.66697 0.0533333

*********************************************************#
# Graph hat folgende Strukturbeschreibung:              #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_1^f5_1:c,21,{[0.046,0.107]}
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_5^f5_5:at4,{52,72},{[-0.184,-0.023]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_18:o,40,{[-0.393,-0.354]}
f4_10^f5_17:o,40,{[-0.393,-0.354]}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
f4_18^f5_10:c,14,{[0.608,0.646]}
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_14:h,3,{[0.057,0.096]}
f4_21^f5_15:h,3,{[0.057,0.096]}
```



```
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_4^f5_4,f4_3^f5_3:7
f4_5^f5_5,f4_1^f5_1:7
f4_5^f5_5,f4_4^f5_4:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_18,f4_8^f5_16:2
f4_10^f5_17,f4_8^f5_16:2
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_14,f4_17^f5_8:1
f4_21^f5_15,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1

#*********************************************************#
```

## Anhang B VIII - Graphbeschreibung zu Abbildung 9-10

```
#*********************************************************#
# WTA- Netz mit Berueckschtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                   #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:        *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0.5
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.0566667 0.666667 0.0566667

*********************************************************#
# Graph hat folgende Strukturbeschreibung:              #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_2^f5_2:Atom1,{21,34},{-0.494,-0.123}
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_5^f5_5:at4,{52,72},{[-0.184,-0.023]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_17:o,40,{[-0.393,-0.354]}
f4_10^f5_18:o,40,{[-0.393,-0.354]}
f4_14^f5_1:Atom1,{36,21},{-0.293,0.046}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
f4_18^f5_10:c,14,{[0.608,0.646]}
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_15:h,3,{[0.057,0.096]}
f4_21^f5_14:h,3,{[0.057,0.096]}
```

```
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_3^f5_3,f4_2^f5_2:7
f4_4^f5_4,f4_3^f5_3:7
f4_5^f5_5,f4_4^f5_4:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_17,f4_8^f5_16:2
f4_10^f5_18,f4_8^f5_16:2
f4_15^f5_9,f4_14^f5_1:1
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_15,f4_17^f5_8:1
f4_21^f5_14,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1

#*********************************************************#
```

## Anhang B IX - Graphbeschreibung zu Abbildung 9-11

```
#*********************************************************#
# WTA- Netz mit Berueckschtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                   #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:        *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0.6
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet und mit KnotenAehnlichkeit gewichtet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.0533333 0.666969 0.0533333

*********************************************************#
# Graph hat folgende Strukturbeschreibung:              #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_1^f5_1:c,21,{[0.046,0.107]}
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_5^f5_5:at4,{52,72},{[-0.184,-0.023]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_18:o,40,{[-0.393,-0.354]}
f4_10^f5_17:o,40,{[-0.393,-0.354]}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
f4_18^f5_10:c,14,{[0.608,0.646]}
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_15:h,3,{[0.057,0.096]}
f4_21^f5_14:h,3,{[0.057,0.096]}
```



```
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_4^f5_4,f4_3^f5_3:7
f4_5^f5_5,f4_1^f5_1:7
f4_5^f5_5,f4_4^f5_4:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_18,f4_8^f5_16:2
f4_10^f5_17,f4_8^f5_16:2
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_15,f4_17^f5_8:1
f4_21^f5_14,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1

#*********************************************************#
```

## Anhang B X - Graphbeschreibung zu Abbildung 9-12

```
#*********************************************************#
# WTA- Netz mit Berueckschtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                   #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:        *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0.5
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet und mit KnotenAehnlichkeit gewichtet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.0566667 0.666667 0.0566667

*********************************************************#
# Graph hat folgende Strukturbeschreibung:              #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_2^f5_2:Atom1,{21,34},{-0.494,-0.123}
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_5^f5_5:at4,{52,72},{[-0.184,-0.023]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_18:o,40,{[-0.393,-0.354]}
f4_10^f5_17:o,40,{[-0.393,-0.354]}
f4_14^f5_1:Atom1,{36,21},{-0.293,0.046}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
f4_18^f5_10:c,14,{[0.608,0.646]}
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_15:h,3,{[0.057,0.096]}
f4_21^f5_14:h,3,{[0.057,0.096]}
```

```
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_3^f5_3,f4_2^f5_2:7
f4_4^f5_4,f4_3^f5_3:7
f4_5^f5_5,f4_4^f5_4:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_18,f4_8^f5_16:2
f4_10^f5_17,f4_8^f5_16:2
f4_15^f5_9,f4_14^f5_1:1
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_15,f4_17^f5_8:1
f4_21^f5_14,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1

#*********************************************************#
```

## Anhang B XI - Graphbeschreibung zu Abbildung 9-13

```
#*********************************************************#
# WTA- Netz mit Beruecksichtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                     #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:          *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0.6
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Knoten mit Aehnlichkeit 1 kann x Prozent der Kanten
        des Graphen aufwiegen (x = 10)

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.158095 0.786667 0.0533333

*********************************************************#
# Graph hat folgende Strukturbeschreibung:                #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_1^f5_1:c,21,{[0.046,0.107]}
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_5^f5_5:at4,{52,72},{[-0.184,-0.023]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_17:o,40,{[-0.393,-0.354]}
f4_10^f5_18:o,40,{[-0.393,-0.354]}
f4_12^f5_2:n,{32,34},{[-0.494,-0.393]}
f4_13^f5_12:h,3,{[0.057,0.096]}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_16^f5_13:h,{1,3},{[0.096,0.157]}
f4_17^f5_8:c,10,{[0.007,0.046]}
```



```
f4_18^f5_10:c,14,{[0.608,0.646]}
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_15:h,3,{[0.057,0.096]}
f4_21^f5_14:h,3,{[0.057,0.096]}
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_4^f5_4,f4_3^f5_3:7
f4_5^f5_5,f4_1^f5_1:7
f4_5^f5_5,f4_4^f5_4:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_17,f4_8^f5_16:2
f4_10^f5_18,f4_8^f5_16:2
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_15,f4_17^f5_8:1
f4_21^f5_14,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1

#*********************************************************#
```

## Anhang B XII - Graphbeschreibung zu Abbildung 9-14

```
#*********************************************************#
# WTA- Netz mit Beruecksichtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                     #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:          *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0.5
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Knoten mit Aehnlichkeit 1 kann x Prozent der Kanten
        des Graphen aufwiegen (x = 10)

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.151393 0.739752 0.0533333

*********************************************************#
# Graph hat folgende Strukturbeschreibung:                #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_1^f5_1:c,21,{[0.046,0.107]}
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_5^f5_5:at4,(52,72),{[-0.184,-0.023]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_17:o,40,{[-0.393,-0.354]}
f4_10^f5_18:o,40,{[-0.393,-0.354]}
f4_12^f5_2:n,{32,34},{[-0.494,-0.393]}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
```

```
f4_17^f5_8:c,10,{[0.007,0.046]}
f4_18^f5_10:c,14,{[0.608,0.646]}
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_15:h,3,{[0.057,0.096]}
f4_21^f5_14:h,3,{[0.057,0.096]}
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_4^f5_4,f4_3^f5_3:7
f4_5^f5_5,f4_1^f5_1:7
f4_5^f5_5,f4_4^f5_4:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_17,f4_8^f5_16:2
f4_10^f5_18,f4_8^f5_16:2
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_15,f4_17^f5_8:1
f4_21^f5_14,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1

#*********************************************************#
```

## Anhang B XIII - Graphbeschreibung zu Abbildung 9-15

```
#*********************************************************#
# WTA- Netz mit Berueckschtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                   #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:        *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0.6
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten anhand der Kanten- und Knotenaehnlichkeiten skaliert

VergleichsGuete : 0.0511111 0.66697 0.0533333

*********************************************************#
# Graph hat folgende Strukturbeschreibung:              #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_1^f5_1:c,21,{[0.046,0.107]}
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_5^f5_5:at4,{52,72},{[-0.184,-0.023]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_17:o,40,{[-0.393,-0.354]}
f4_10^f5_18:o,40,{[-0.393,-0.354]}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
```



```
f4_18^f5_10:c,14,{[0.608,0.646]}
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_15:h,3,{[0.057,0.096]}
f4_21^f5_14:h,3,{[0.057,0.096]}
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_4^f5_4,f4_3^f5_3:7
f4_5^f5_5,f4_1^f5_1:7
f4_5^f5_5,f4_4^f5_4:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_17,f4_8^f5_16:2
f4_10^f5_18,f4_8^f5_16:2
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_15,f4_17^f5_8:1
f4_21^f5_14,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1

#*********************************************************#
```

## Anhang B XIV - Graphbeschreibung zu Abbildung 9-16

```
#*********************************************************#
# WTA- Netz mit Berueckschtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                   #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:        *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0.5
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten anhand der Kanten- und Knotenaehnlichkeiten skaliert

VergleichsGuete : 0.0522222 0.666667 0.0566667

*********************************************************#
# Graph hat folgende Strukturbeschreibung:              #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_2^f5_2:Atom1,{21,34},(-0.494,-0.123)
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_5^f5_5:at4,{52,72},{[-0.184,-0.023]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_17:o,40,{[-0.393,-0.354]}
f4_10^f5_18:o,40,{[-0.393,-0.354]}
f4_14^f5_1:Atom1,{36,21},(-0.293,0.046)
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
```

```
f4_18^f5_10:c,14,{[0.608,0.646]}
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_15:h,3,{[0.057,0.096]}
f4_21^f5_14:h,3,{[0.057,0.096]}
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_3^f5_3,f4_2^f5_2:7
f4_4^f5_4,f4_3^f5_3:7
f4_5^f5_5,f4_4^f5_4:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_17,f4_8^f5_16:2
f4_10^f5_18,f4_8^f5_16:2
f4_15^f5_9,f4_14^f5_1:1
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_15,f4_17^f5_8:1
f4_21^f5_14,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1

#*********************************************************#
```

## Anhang B XV - Graphbeschreibung zu Abbildung 9-17

```
#*********************************************************#
# WTA- Netz mit Beruecksichtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                    #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:         *
keine Selbsthemmung, ( d = 0 )

schrittweises Matching (lose)
AehnlichkeitsSchwelle(n) fuer Knoten : {0.5,0.6,1}
AehnlichkeitsSchwelle(n) fuer Kanten : 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.06 0.686667 0.06

*********************************************************#
# Graph hat folgende Strukturbeschreibung:               #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_1^f5_1:c,21,{[0.046,0.107]}
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_18:o,40,{[-0.393,-0.354]}
f4_10^f5_17:o,40,{[-0.393,-0.354]}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
f4_18^f5_10:c,14,{[0.608,0.646]}
```



```
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_14:h,3,{[0.057,0.096]}
f4_21^f5_15:h,3,{[0.057,0.096]}
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_5^f5_5:at4,{52,72},{[-0.184,-0.023]}
f4_2^f5_2:Atom1,{21,34},{-0.494,-0.123}
f4_4^f5_4,f4_3^f5_3:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_18,f4_8^f5_16:2
f4_10^f5_17,f4_8^f5_16:2
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_14,f4_17^f5_8:1
f4_21^f5_15,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1
f4_5^f5_5,f4_1^f5_1:7
f4_5^f5_5,f4_4^f5_4:7
f4_2^f5_2,f4_1^f5_1:7
f4_2^f5_2,f4_3^f5_3:7

#*********************************************************#
```

## Anhang B XVI - Graphbeschreibung zu Abbildung 9-18

```
#*********************************************************#
# WTA- Netz mit Beruecksichtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                    #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:         *
keine Selbsthemmung, ( d = 0 )

schrittweises Matching (fixiert)
AehnlichkeitsSchwelle(n) fuer Knoten : {0.5,0.6,1}
AehnlichkeitsSchwelle(n) fuer Kanten : 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.0566667 0.666667 0.0566667

*********************************************************#
# Graph hat folgende Strukturbeschreibung:               #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_18:o,40,{[-0.393,-0.354]}
f4_10^f5_17:o,40,{[-0.393,-0.354]}
f4_15^f5_9:n,32,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
```

```
f4_18^f5_10:c,14,{[0.608,0.646]}
f4_19^f5_7:c,{14,10},{0.046,0.608}
f4_20^f5_15:h,3,{[0.057,0.096]}
f4_21^f5_14:h,3,{[0.057,0.096]}
f4_22^f5_6:n,32,{[-0.393,-0.354]}
f4_23^f5_20:o,40,{[-0.542,-0.504]}
f4_25^f5_19:h,1,{[0.257,0.346]}
f4_5^f5_5:at4,{52,72},{[-0.184,-0.023]}
f4_2^f5_2:Atom1,{21,34},{-0.494,-0.123}
f4_14^f5_1:Atom1,{36,21},{-0.293,0.046}
f4_4^f5_4,f4_3^f5_3:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_18,f4_8^f5_16:2
f4_10^f5_17,f4_8^f5_16:2
f4_17^f5_8,f4_15^f5_9:1
f4_18^f5_10,f4_15^f5_9:1
f4_19^f5_7,f4_17^f5_8:1
f4_20^f5_15,f4_17^f5_8:1
f4_21^f5_14,f4_17^f5_8:1
f4_22^f5_6,f4_18^f5_10:1
f4_22^f5_6,f4_19^f5_7:1
f4_23^f5_20,f4_18^f5_10:2
f4_25^f5_19,f4_22^f5_6:1
f4_5^f5_5,f4_4^f5_4:7
f4_2^f5_2,f4_3^f5_3:7
f4_14^f5_1,f4_15^f5_9:1

#*********************************************************#
```

## Anhang B XVII - Graphbeschreibung zu Abbildung 9-19

```
#*********************************************************#
# WTA- Netz mit Beruecksichtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                     #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:          *
keine Selbsthemmung, ( d = 0 )

schrittweises Matching (reduziert)
AehnlichkeitsSchwelle(n) fuer Knoten : {0.5,0.6,1}
AehnlichkeitsSchwelle(n) fuer Kanten : 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert

VergleichsGuete : 0.0466667 0.5 0.0466667

*********************************************************#
# Graph hat folgende Strukturbeschreibung:                #
Knoten:
begriff(1,atomhier,Teil),
symbol(0),
reell_2(0,0.1,0.2)
Kante:
symbol(1)

#*********************************************************#
f4_3^f5_3:c,21,{[-0.123,0.046]}
f4_4^f5_4:c,21,{[0.007,0.046]}
f4_7^f5_11:h,3,{[0.137,0.146]}
f4_8^f5_16:n,38,{[0.808,0.846]}
f4_9^f5_17:o,40,{[-0.393,-0.354]}
f4_10^f5_18:o,40,{[-0.393,-0.354]}
f4_17^f5_8:c,10,{[0.007,0.046]}
```



```
f4_20^f5_15:h,3,{[0.057,0.096]}
f4_21^f5_14:h,3,{[0.057,0.096]}
f4_1^f5_5:Atom1,{21,72},{-0.184,0.107}
f4_2^f5_2:Atom1,{21,34},{-0.494,-0.123}
f4_5^f5_1:Atom1,{52,21},{[-0.023,0.046]}
f4_15^f5_7:Atom1,{32,10},{-0.393,0.046}
f4_18^f5_6:Atom1,{14,32},{-0.354,0.608}
f4_19^f5_9:Atom1,{14,32},{-0.354,0.608}
f4_22^f5_10:Atom1,{32,14},{-0.393,0.646}
f4_4^f5_4,f4_3^f5_3:7
f4_7^f5_11,f4_3^f5_3:1
f4_8^f5_16,f4_4^f5_4:1
f4_9^f5_17,f4_8^f5_16:2
f4_10^f5_18,f4_8^f5_16:2
f4_20^f5_15,f4_17^f5_8:1
f4_21^f5_14,f4_17^f5_8:1
f4_2^f5_2,f4_3^f5_3:7
f4_5^f5_1,f4_1^f5_5:7
f4_15^f5_7,f4_17^f5_8:1
f4_18^f5_6,f4_15^f5_7:1
f4_19^f5_9,f4_17^f5_8:1
f4_22^f5_10,f4_18^f5_6:1
f4_22^f5_10,f4_19^f5_9:1

#*********************************************************#
```

## Anhang B XVIII - Graphbeschreibung des Moleküls 'f1'

```
# autom generierte Graphbeschreibung, Graph 'f1' #

# Knoten # f1_1:c,21,0.187
# Knoten # f1_2:c,21,-0.143
# Knoten # f1_3:c,21,-0.143
# Knoten # f1_4:c,21,-0.013
# Knoten # f1_5:o,52,-0.043
# Knoten # f1_6:h,3,0.117
# Knoten # f1_7:h,3,0.117
# Knoten # f1_8:n,38,0.786
# Knoten # f1_9:o,40,-0.413
# Knoten # f1_10:o,40,-0.413
# Knoten # f1_11:c,14,0.217
# Knoten # f1_12:o,41,-0.443
# Knoten # f1_13:h,3,0.187
# Kante  # f1_1,f1_2:7
# Kante  # f1_2,f1_3:7
# Kante  # f1_3,f1_4:7
# Kante  # f1_4,f1_5:7
# Kante  # f1_5,f1_1:7
# Kante  # f1_2,f1_6:1
# Kante  # f1_3,f1_7:1
# Kante  # f1_4,f1_8:1
# Kante  # f1_8,f1_9:2
# Kante  # f1_8,f1_10:2
# Kante  # f1_1,f1_11:1
# Kante  # f1_11,f1_12:2
# Kante  # f1_11,f1_13:1
```

## Anhang B XIX - Graphbeschreibung des Moleküls 'f2'

```
# autom generierte Graphbeschreibung, Graph 'f2' #

# Knoten # f2_1:c,21,0.006
# Knoten # f2_2:c,21,-0.124
# Knoten # f2_3:c,21,-0.124
# Knoten # f2_4:c,21,0.106
# Knoten # f2_5:c,21,-0.124
# Knoten # f2_6:h,3,0.136
# Knoten # f2_7:h,3,0.136
# Knoten # f2_8:h,3,0.136
# Knoten # f2_9:n,38,0.805
# Knoten # f2_10:o,40,-0.395
# Knoten # f2_11:o,40,-0.395
# Knoten # f2_12:c,16,-0.195
```

```
# Knoten # f2_13:c,16,-0.195
# Knoten # f2_14:c,14,0.235
# Knoten # f2_15:o,41,-0.425
# Knoten # f2_16:h,3,0.205
# Knoten # f2_17:h,3,0.106
# Knoten # f2_18:h,3,0.106
# Kante  # f2_1,f2_2:7
# Kante  # f2_2,f2_3:7
# Kante  # f2_3,f2_4:7
# Kante  # f2_4,f2_5:7
# Kante  # f2_5,f2_1:7
# Kante  # f2_2,f2_6:1
# Kante  # f2_3,f2_7:1
# Kante  # f2_5,f2_8:1
# Kante  # f2_1,f2_9:1
# Kante  # f2_9,f2_10:2
# Kante  # f2_9,f2_11:2
# Kante  # f2_12,f2_13:2
# Kante  # f2_13,f2_14:1
# Kante  # f2_14,f2_15:2
# Kante  # f2_14,f2_16:1
# Kante  # f2_13,f2_17:1
# Kante  # f2_4,f2_12:1
# Kante  # f2_12,f2_18:1
```

## Anhang B XX - Graphbeschreibung des Moleküls 'f3'

```
# autom generierte Graphbeschreibung, Graph 'f3' #

# Knoten # f3_1:c,21,0.086
# Knoten # f3_2:c,21,-0.144
# Knoten # f3_3:c,21,-0.144
# Knoten # f3_4:c,21,-0.014
# Knoten # f3_5:o,52,-0.044
# Knoten # f3_6:h,3,0.116
# Knoten # f3_7:h,3,0.116
# Knoten # f3_8:n,38,0.788
# Knoten # f3_9:o,40,-0.414
# Knoten # f3_10:o,40,-0.414
# Knoten # f3_11:c,14,0.587
# Knoten # f3_12:n,32,-0.414
# Knoten # f3_13:h,3,0.036
# Knoten # f3_14:n,32,-0.414
# Knoten # f3_15:c,14,0.587
# Knoten # f3_16:h,1,0.287
# Knoten # f3_17:o,40,-0.563
# Knoten # f3_18:n,32,-0.612
# Knoten # f3_19:h,1,0.287
# Knoten # f3_20:h,1,0.287
# Kante  # f3_1,f3_2:7
# Kante  # f3_2,f3_3:7
# Kante  # f3_3,f3_4:7
# Kante  # f3_4,f3_5:7
# Kante  # f3_5,f3_1:7
# Kante  # f3_2,f3_6:1
# Kante  # f3_3,f3_7:1
# Kante  # f3_4,f3_8:1
# Kante  # f3_8,f3_9:2
# Kante  # f3_8,f3_10:2
# Kante  # f3_1,f3_11:1
# Kante  # f3_11,f3_12:2
# Kante  # f3_11,f3_13:1
# Kante  # f3_12,f3_14:1
# Kante  # f3_14,f3_15:1
# Kante  # f3_14,f3_16:1
# Kante  # f3_15,f3_17:2
# Kante  # f3_15,f3_18:1
# Kante  # f3_18,f3_19:1
# Kante  # f3_18,f3_20:1
```



## Anhang B XXI - Graphbeschreibung des Moleküls 'f4'

```
# autom generierte Graphbeschreibung, Graph 'f4' #

# Knoten # f4_1:c,21,0.107
# Knoten # f4_2:c,21,-0.123
# Knoten # f4_3:c,21,-0.123
# Knoten # f4_4:c,21,0.007
# Knoten # f4_5:o,52,-0.023
# Knoten # f4_6:h,3,0.137
# Knoten # f4_7:h,3,0.137
# Knoten # f4_8:n,38,0.808
# Knoten # f4_9:o,40,-0.393
# Knoten # f4_10:o,40,-0.393
# Knoten # f4_11:c,14,0.608
# Knoten # f4_12:n,32,-0.393
# Knoten # f4_13:h,3,0.057
# Knoten # f4_14:n,36,-0.293
# Knoten # f4_15:n,32,-0.393
# Knoten # f4_16:h,1,0.157
# Knoten # f4_17:c,10,0.007
# Knoten # f4_18:c,14,0.608
# Knoten # f4_19:c,14,0.608
# Knoten # f4_20:h,3,0.057
# Knoten # f4_21:h,3,0.057
# Knoten # f4_22:n,32,-0.393
# Knoten # f4_23:o,40,-0.542
# Knoten # f4_24:o,40,-0.543
# Knoten # f4_25:h,1,0.257
# Kante  # f4_1,f4_2:7
# Kante  # f4_2,f4_3:7
# Kante  # f4_3,f4_4:7
# Kante  # f4_4,f4_5:7
# Kante  # f4_5,f4_1:7
# Kante  # f4_2,f4_6:1
# Kante  # f4_3,f4_7:1
# Kante  # f4_4,f4_8:1
# Kante  # f4_8,f4_9:2
# Kante  # f4_8,f4_10:2
# Kante  # f4_1,f4_11:1
# Kante  # f4_11,f4_12:2
# Kante  # f4_11,f4_13:1
# Kante  # f4_12,f4_14:1
# Kante  # f4_14,f4_15:1
# Kante  # f4_14,f4_16:1
# Kante  # f4_15,f4_17:1
# Kante  # f4_15,f4_18:1
# Kante  # f4_17,f4_19:1
# Kante  # f4_17,f4_20:1
# Kante  # f4_17,f4_21:1
# Kante  # f4_19,f4_22:1
# Kante  # f4_22,f4_18:1
# Kante  # f4_18,f4_23:2
# Kante  # f4_19,f4_24:2
# Kante  # f4_22,f4_25:1
```

## Anhang B XXII - Graphbeschreibung des Moleküls 'f5'

```
# autom generierte Graphbeschreibung, Graph 'f5' #

# Knoten # f5_1:c,21,0.046
# Knoten # f5_2:n,34,-0.494
# Knoten # f5_3:c,21,0.046
# Knoten # f5_4:c,21,0.046
# Knoten # f5_5:s,72,-0.184
# Knoten # f5_6:n,32,-0.354
# Knoten # f5_7:c,10,0.046
# Knoten # f5_8:c,10,0.046
# Knoten # f5_9:n,32,-0.354
# Knoten # f5_10:c,14,0.646
# Knoten # f5_11:h,3,0.146
# Knoten # f5_12:h,3,0.096
```

```
# Knoten # f5_13:h,3,0.096
# Knoten # f5_14:h,3,0.096
# Knoten # f5_15:h,3,0.096
# Knoten # f5_16:n,38,0.846
# Knoten # f5_17:o,40,-0.354
# Knoten # f5_18:o,40,-0.354
# Knoten # f5_19:h,1,0.346
# Knoten # f5_20:o,40,-0.504
# Kante  # f5_1,f5_2:7
# Kante  # f5_2,f5_3:7
# Kante  # f5_3,f5_4:7
# Kante  # f5_4,f5_5:7
# Kante  # f5_5,f5_1:7
# Kante  # f5_6,f5_7:1
# Kante  # f5_7,f5_8:1
# Kante  # f5_8,f5_9:1
# Kante  # f5_9,f5_10:1
# Kante  # f5_10,f5_6:1
# Kante  # f5_3,f5_11:1
# Kante  # f5_7,f5_12:1
# Kante  # f5_7,f5_13:1
# Kante  # f5_8,f5_14:1
# Kante  # f5_8,f5_15:1
# Kante  # f5_4,f5_16:1
# Kante  # f5_16,f5_17:2
# Kante  # f5_16,f5_18:2
# Kante  # f5_1,f5_9:1
# Kante  # f5_6,f5_19:1
# Kante  # f5_10,f5_20:2
```

## Anhang B XXIII - Graphbeschreibung des Moleküls 'f6'

```
# autom generierte Graphbeschreibung, Graph 'f6' #

# Knoten # f6_1:c,21,0.106
# Knoten # f6_2:c,21,-0.124
# Knoten # f6_3:c,21,-0.124
# Knoten # f6_4:c,21,0.006
# Knoten # f6_5:o,52,-0.024
# Knoten # f6_6:h,3,0.136
# Knoten # f6_7:h,3,0.136
# Knoten # f6_8:n,38,0.805
# Knoten # f6_9:o,40,-0.395
# Knoten # f6_10:o,40,-0.394
# Knoten # f6_11:c,14,0.605
# Knoten # f6_12:n,32,-0.395
# Knoten # f6_13:h,3,0.056
# Knoten # f6_14:n,32,-0.395
# Knoten # f6_15:c,10,0.006
# Knoten # f6_16:c,14,0.605
# Knoten # f6_17:c,10,0.256
# Knoten # f6_18:h,3,0.056
# Knoten # f6_19:h,3,0.056
# Knoten # f6_20:o,49,-0.645
# Knoten # f6_21:o,51,-0.545
# Knoten # f6_22:h,3,0.106
# Knoten # f6_23:h,3,0.106
# Kante  # f6_1,f6_2:7
# Kante  # f6_2,f6_3:7
# Kante  # f6_3,f6_4:7
# Kante  # f6_4,f6_5:7
# Kante  # f6_5,f6_1:7
# Kante  # f6_2,f6_6:1
# Kante  # f6_3,f6_7:1
# Kante  # f6_4,f6_8:1
# Kante  # f6_8,f6_9:2
# Kante  # f6_8,f6_10:2
# Kante  # f6_1,f6_11:1
# Kante  # f6_11,f6_12:2
# Kante  # f6_11,f6_13:1
# Kante  # f6_14,f6_15:1
# Kante  # f6_14,f6_16:1
# Kante  # f6_15,f6_17:1
# Kante  # f6_15,f6_18:1
```



```
# Kante  # f6_15,f6_19:1
# Kante  # f6_17,f6_20:1
# Kante  # f6_20,f6_16:1
# Kante  # f6_16,f6_21:2
# Kante  # f6_12,f6_14:1
# Kante  # f6_17,f6_22:1
# Kante  # f6_17,f6_23:1
```

## Anhang B XXIV - Graphbeschreibung des Moleküls 'd191'

```
# autom generierte Graphbeschreibung, Graph 'd191' #

# Knoten # d191_1:c,22,-0.133
# Knoten # d191_2:c,22,-0.133
# Knoten # d191_3:c,27,-0.002
# Knoten # d191_4:c,27,-0.102
# Knoten # d191_5:c,27,-0.002
# Knoten # d191_6:c,22,-0.133
# Knoten # d191_7:c,27,-0.102
# Knoten # d191_8:c,27,-0.002
# Knoten # d191_9:c,22,-0.133
# Knoten # d191_10:c,22,-0.133
# Knoten # d191_11:c,26,0.098
# Knoten # d191_12:c,26,0.098
# Knoten # d191_13:c,22,-0.133
# Knoten # d191_14:c,22,-0.133
# Knoten # d191_15:c,22,-0.132
# Knoten # d191_16:c,26,0.098
# Knoten # d191_17:c,21,-0.102
# Knoten # d191_18:c,21,-0.002
# Knoten # d191_19:h,3,0.128
# Knoten # d191_20:h,3,0.128
# Knoten # d191_21:h,3,0.128
# Knoten # d191_22:h,3,0.128
# Knoten # d191_23:h,3,0.128
# Knoten # d191_24:h,3,0.098
# Knoten # d191_25:h,3,0.128
# Knoten # d191_26:h,3,0.128
# Knoten # d191_27:h,3,0.098
# Knoten # d191_28:n,38,0.797
# Knoten # d191_29:o,40,-0.403
# Knoten # d191_30:o,40,-0.403
# Kante  # d191_1,d191_2:7
# Kante  # d191_2,d191_3:7
# Kante  # d191_3,d191_4:7
# Kante  # d191_4,d191_5:7
# Kante  # d191_5,d191_6:7
# Kante  # d191_6,d191_1:7
# Kante  # d191_4,d191_7:7
# Kante  # d191_7,d191_8:7
# Kante  # d191_8,d191_9:7
# Kante  # d191_9,d191_10:7
# Kante  # d191_10,d191_5:7
# Kante  # d191_7,d191_11:7
# Kante  # d191_11,d191_12:7
# Kante  # d191_12,d191_13:7
# Kante  # d191_13,d191_14:7
# Kante  # d191_14,d191_8:7
# Kante  # d191_3,d191_15:7
# Kante  # d191_15,d191_16:7
# Kante  # d191_16,d191_11:7
# Kante  # d191_12,d191_17:7
# Kante  # d191_16,d191_18:7
# Kante  # d191_18,d191_17:7
# Kante  # d191_1,d191_19:1
# Kante  # d191_2,d191_20:1
# Kante  # d191_6,d191_21:1
# Kante  # d191_9,d191_22:1
# Kante  # d191_10,d191_23:1
# Kante  # d191_13,d191_24:1
# Kante  # d191_14,d191_25:1
# Kante  # d191_15,d191_26:1
# Kante  # d191_17,d191_27:1
# Kante  # d191_18,d191_28:1
```

```
# Kante  # d191_28,d191_29:2
# Kante  # d191_28,d191_30:2
```

## Anhang B XXV - Graphbeschreibung des Moleküls 'd197'

```
# autom generierte Graphbeschreibung, Graph 'd197' #

# Knoten # d197_1:c,22,-0.086
# Knoten # d197_2:c,22,-0.086
# Knoten # d197_3:c,26,-0.056
# Knoten # d197_4:c,26,-0.056
# Knoten # d197_5:c,22,-0.086
# Knoten # d197_6:c,22,-0.086
# Knoten # d197_7:n,34,-0.496
# Knoten # d197_8:c,21,0.044
# Knoten # d197_9:n,34,-0.496
# Knoten # d197_10:c,10,0.294
# Knoten # d197_11:n,38,0.844
# Knoten # d197_12:o,40,-0.356
# Knoten # d197_13:o,40,-0.356
# Knoten # d197_14:h,3,0.174
# Knoten # d197_15:h,3,0.174
# Knoten # d197_16:h,3,0.174
# Knoten # d197_17:h,3,0.174
# Knoten # d197_18:h,3,0.094
# Knoten # d197_19:h,3,0.094
# Knoten # d197_20:h,3,0.094
# Kante  # d197_1,d197_2:7
# Kante  # d197_2,d197_3:7
# Kante  # d197_3,d197_4:7
# Kante  # d197_4,d197_5:7
# Kante  # d197_5,d197_6:7
# Kante  # d197_6,d197_1:7
# Kante  # d197_3,d197_7:7
# Kante  # d197_7,d197_8:7
# Kante  # d197_4,d197_9:7
# Kante  # d197_9,d197_8:7
# Kante  # d197_7,d197_10:1
# Kante  # d197_8,d197_11:1
# Kante  # d197_11,d197_12:2
# Kante  # d197_11,d197_13:2
# Kante  # d197_1,d197_14:1
# Kante  # d197_2,d197_15:1
# Kante  # d197_5,d197_16:1
# Kante  # d197_6,d197_17:1
# Kante  # d197_10,d197_18:1
# Kante  # d197_10,d197_19:1
# Kante  # d197_10,d197_20:1
```

## Anhang B XXVI - Graphbeschreibung des generalisierten Ergebnisgraphen (Abb. 9-21)

```
#*********************************************************#
# WTA- Netz mit Beruecksichtigung lokaler Aehnlichkeiten #
#************************************************ V1.02 *#
# autom. generierte Graphbeschreibung                     #
#*********************************************************#

#*********************************************************#
# Graph wurde mit folgenden Parametern erstellt:          *
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 0
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet und mit KnotenAehnlichkeit gewichtet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
```



```
        Kanten nicht skaliert

VergleichsGuete : 0.2 0.933566 0.2

*********************************************************#
# Graph hat folgende Strukturbeschreibung:                #
Knoten:
begriff(0.8,atomhier,Gesamt),
symbol(0.4),
reell_2(0.2,0.3,0.2)
Kante:
symbol(1)

#*********************************************************#
<<<<<<f1_1^f2_4>^f3_1>^d197_4>^f5_1>^f6_1>^f4_1>^d191_12:c,(21,26),([-
0.056,0.187])
<<<<<<f1_2^f2_3>^f3_2>^d197_3>^f5_2>^f6_2>^f4_2>^d191_11:Atom1,(21,26,34),(-
0.494,[-0.144,0.098])
<<<<<<f1_3^f2_2>^f3_3>^d197_7>^f5_3>^f6_3>^f4_3>^d191_16:Atom1,(21,34,26),(-
0.496,[-0.144,0.098])
<<<<<<f1_4^f2_1>^f3_4>^d197_8>^f5_4>^f6_4>^f4_4>^d191_18:c,21,([-
0.014,0.046])
<<<<<<f1_5^f2_5>^f3_5>^d197_9>^f5_5>^f6_5>^f4_5>^d191_17:Atom1,(52,21,34,72),
(-0.496,[-0.184,-0.023])
<<<<<<f1_8^f2_9>^f3_8>^d197_11>^f5_16>^f6_8>^f4_8>^d191_28:n,38,([0.786,0.846
])
<<<<<<f1_9^f2_10>^f3_9>^d197_13>^f5_18>^f6_10>^f4_9>^d191_29:o,40,([-0.414,-
0.354])
<<<<<<f1_10^f2_11>^f3_10>^d197_12>^f5_17>^f6_9>^f4_10>^d191_30:o,40,([-
0.414,-0.354])
<<<<<<f1_11^f2_12>^f3_11>^d197_10>^f5_8>^f6_17>^f4_17>^d191_1:c,(14,16,10,22)
,([-0.195,0.294],0.587)
<<<<<<f1_13^f2_18>^f3_13>^d197_18>^f5_15>^f6_22>^f4_20>^d191_19:h,3,([0.036,0
.187])
<<<<<<f1_2^f2_3>^f3_2>^d197_3>^f5_2>^f6_2>^f4_2>^d191_11,<<<<<<f1_1^f2_4>^f3_
1>^d197_4>^f5_1>^f6_1>^f4_1>^d191_12:7
<<<<<<f1_3^f2_2>^f3_3>^d197_7>^f5_3>^f6_3>^f4_3>^d191_16,<<<<<<f1_2^f2_3>^f3_
2>^d197_3>^f5_2>^f6_2>^f4_2>^d191_11:7
<<<<<<f1_4^f2_1>^f3_4>^d197_8>^f5_4>^f6_4>^f4_4>^d191_18,<<<<<<f1_3^f2_2>^f3_
3>^d197_7>^f5_3>^f6_3>^f4_3>^d191_16:7
<<<<<<f1_5^f2_5>^f3_5>^d197_9>^f5_5>^f6_5>^f4_5>^d191_17,<<<<<<f1_1^f2_4>^f3_
1>^d197_4>^f5_1>^f6_1>^f4_1>^d191_12:7
<<<<<<f1_5^f2_5>^f3_5>^d197_9>^f5_5>^f6_5>^f4_5>^d191_17,<<<<<<f1_4^f2_1>^f3_
4>^d197_8>^f5_4>^f6_4>^f4_4>^d191_18:7
<<<<<<f1_8^f2_9>^f3_8>^d197_11>^f5_16>^f6_8>^f4_8>^d191_28,<<<<<<f1_4^f2_1>^f
3_4>^d197_8>^f5_4>^f6_4>^f4_4>^d191_18:1
<<<<<<f1_9^f2_10>^f3_9>^d197_13>^f5_18>^f6_10>^f4_9>^d191_29,<<<<<<f1_8^f2_9>
^f3_8>^d197_11>^f5_16>^f6_8>^f4_8>^d191_28:2
<<<<<<f1_10^f2_11>^f3_10>^d197_12>^f5_17>^f6_9>^f4_10>^d191_30,<<<<<<f1_8^f2_
9>^f3_8>^d197_11>^f5_16>^f6_8>^f4_8>^d191_28:2
<<<<<<f1_13^f2_18>^f3_13>^d197_18>^f5_15>^f6_22>^f4_20>^d191_19,<<<<<<f1_11^f
2_12>^f3_11>^d197_10>^f5_8>^f6_17>^f4_17>^d191_1:1

#*********************************************************#
```

# Anhang C - Konvertierungsprogramm 'convert'

Zur Bewertung der verschiedenen Fähigkeiten des Programms war es nötig, dieses mit größeren Datenmengen zu testen. Diese manuell zu erstellen und die Ergebnisse der Einzelvergleiche nachzuprüfen war natürlich nur bedingt möglich, darum wurden die in [King 1995] beschriebenen Datensätze von mutagenen Verbindungen genutzt.

Für eine Nutzung dieser Datensätze war zuallererst eine Umwandlung der Beschreibungen der Verbindungen in das in Abschnitt 7.2 vorgestellte Format notwendig. Dafür wurde ein kleines Programm entwickelt, welches die Graphen eines solchen Datensatzes bestimmt und diese auf Wunsch konvertiert.

Dieses Programm soll an dieser Stelle erwähnt werden, weil damit für spätere Konvertierungen anderer Datensätze eine mögliche Grundlage gegeben ist. Dennoch muß gesagt werden, daß dieses Programm lediglich für *eine* Konvertierung genutzt wurde und diese sinnvoll durchführen konnte- inwiefern es sinnvoll ist, dieses Programm weiter zu nutzen oder zu ändern, soll hierbei nicht meine Sache sein.

Als erstes soll hier ein Ausgabeprotokoll gezeigt werden- dabei sind alle Eingaben kursiv dargestellt.

```
Konvertierung von .pl Datensaetzen zu .graph- Dateien

Name der zu konvertierenden Datei : atom_bond.pl
durchsuche Datei nach vorhandenen Graphen
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************

d1      d10     d100    d101    d102    d103    d104    d105    d106    d107
d108    d109    d11     d110    d111    d112    d113    d114    d115    d116
d117    d118    d119    d12     d120    d121    d122    d123    d124    d125
d126    d127    d128    d129    d13     d130    d131    d132    d133    d134
d135    d136    d137    d138    d139    d14     d140    d141    d142    d143
d144    d145    d146    d147    d148    d149    d15     d150    d151    d152
d153    d154    d155    d156    d157    d158    d159    d16     d160    d161
d162    d163    d164    d165    d166    d167    d168    d169    d17     d170
d171    d172    d173    d174    d175    d176    d177    d178    d179    d18
d180    d181    d182    d183    d184    d185    d186    d187    d188    d189
d19     d190    d191    d192    d193    d194    d195    d196    d197    d2
d20     d21     d22     d23     d24     d25     d26     d27     d28     d29
d3      d30     d31     d32     d33     d34     d35     d36     d37     d38
d39     d4      d40     d41     d42     d43     d44     d45     d46     d47
d48     d49     d5      d50     d51     d52     d53     d54     d55     d56
d57     d58     d59     d6      d60     d61     d62     d63     d64     d65
d66     d67     d68     d69     d7      d70     d71     d72     d73     d74
d75     d76     d77     d78     d79     d8      d80     d81     d82     d83
d84     d85     d86     d87     d88     d89     d9      d90     d91     d92
d93     d94     d95     d96     d97     d98     d99     e1      e10     e11
e12     e13     e14     e15     e16     e17     e18     e19     e2      e20
e21     e22     e23     e24     e25     e26     e27     e3      e4      e5
e6      e7      e8      e9      f1      f2      f3      f4      f5      f6

alle Graphen konvertieren ?? (y/N)
```

Als erstes wird also nach der zu konvertierenden Datei gefragt, diese wird eingelesen und alle Graphbeschreibungen in dieser Datei werden registriert. Nun wird erfragt, ob alle gefundenen Graphen konvertiert werden sollen oder nur bestimmte- sollen alle konvertiert werden, schreibt

das Programm alle *.graph- Dateien ins aktuelle Verzeichnis und endet.

Sollen nur bestimmte Graphen konvertiert werden (Eingabe 'n' an oberer Programmstelle) - fährt das Programm mit folgender Ausgabe fort:

```
gefundene Graphen (* - ausgewaehlt)
 d1      d10     d100    d101    d102    d103    d104    d105    d106    d107
 d108    d109    d11     d110    d111    d112    d113    d114    d115    d116
 d117    d118    d119    d12     d120    d121    d122    d123    d124    d125
 d126    d127    d128    d129    d13     d130    d131    d132    d133    d134
 d135    d136    d137    d138    d139    d14     d140    d141    d142    d143
 d144    d145    d146    d147    d148    d149    d15     d150    d151    d152
 d153    d154    d155    d156    d157    d158    d159    d16     d160    d161
 d162    d163    d164    d165    d166    d167    d168    d169    d17     d170
 d171    d172    d173    d174    d175    d176    d177    d178    d179    d18
 d180    d181    d182    d183    d184    d185    d186    d187    d188    d189
 d19     d190    d191    d192    d193    d194    d195    d196    d197    d2
 d20     d21     d22     d23     d24     d25     d26     d27     d28     d29
 d3      d30     d31     d32     d33     d34     d35     d36     d37     d38
 d39     d4      d40     d41     d42     d43     d44     d45     d46     d47
 d48     d49     d5      d50     d51     d52     d53     d54     d55     d56
 d57     d58     d59     d6      d60     d61     d62     d63     d64     d65
 d66     d67     d68     d69     d7      d70     d71     d72     d73     d74
 d75     d76     d77     d78     d79     d8      d80     d81     d82     d83
 d84     d85     d86     d87     d88     d89     d9      d90     d91     d92
 d93     d94     d95     d96     d97     d98     d99     e1      e10     e11
 e12     e13     e14     e15     e16     e17     e18     e19     e2      e20
 e21     e22     e23     e24     e25     e26     e27     e3      e4      e5
 e6      e7      e8      e9      f1      f2      f3      f4      f5      f6

Bitte waehlen Sie aus:
 GraphAuswahl gesamt invertieren ------- i
 Auswahl fuer EinzelGraph invertieren -- e
 Graphen konvertieren ------------------ a
```

Alle Graphen werden aufgelistet und können nun einzeln ausgewählt werden. Dabei werden ausgewählte Graphen mit einem '*' vor dem Graphbezeichner markiert (im oberen Beispiel sind keine Graphen ausgewählt). Eine weitere Auswahlmöglichkeit besteht darin, die aktuelle Auswahl aller Graphen zu invertieren, was bedeuten soll, daß alle Graphen, welche vorher ausgewählt waren nach der Invertierung nicht mehr ausgewählt sind und umgekehrt. Ist die gewünschte Auswahl getroffen, können die Graphen konvertiert werden.

Bei der Graphausgabe wird darauf geachtet, daß keine Dateien überschrieben werden- falls dies nötig ist erfolgt eine Sicherheitsabfrage.

# Beschreibung der Programmierschnittstelle zum WTA- Netz mit Berücksichtigung lokaler Ähnlichkeiten

*Anhang D zur Diplomarbeit*

*"Entwurf und Implementierung eines WTA- Netzes für Graphmatching unter Einbeziehung lokaler Ähnlichkeiten"*

*René Ejury, Februar 1998*

In diesem Teil des Anhangs soll eine Kurzbeschreibung der Programmierschnittstelle angeführt werden. Dabei wird auffallen, daß wesentliche Teile dieser Kurzbeschreibung auch innerhalb der Arbeit selbst auftreten. Dennoch war es schon während der Implementierungsphase des Programms nötig, Studierenden, welche das Programm nutzen wollten, eine kompakte Beschreibung der Schnittstelle zur Verfügung zu stellen.

Natürlich bestände die Möglichkeit, Studierenden für die Nutzung des Programms die gesamte Arbeit zukommen zu lassen- allerdings verringert dies auch die Möglichkeit, *schnell* die *nötigen* Informationen zu finden.

Aus den eben beschriebenen Gründen und aus der eigenen Erfahrung mit Programmunterlagen habe ich mich daher entschlossen, eine komplette Kurzbeschreibung zusammenzustellen, zu finden auf den folgenden Seiten. Alle Verweise innerhalb dieser Beschreibung beziehen sich demzufolge nur auf die entsprechenden Quellen innerhalb dieser Beschreibung.

## Einführung

Die Programmierschnittstelle ist wie das gesamte Programm in C++ implementiert und wurde im Hinblick auf einfache, direkte Nutzung der Möglichkeiten des WTA- Netzes durch Einbindung in eigene Programme entwickelt. Das eigentliche Programm wurde mit GNU C++ compiliert, die Nutzung unter anderen Compilern kann daher Probleme bereiten (oder ist nicht möglich). Da jedoch der GNU- Compiler auf den verschiedensten Plattformen zugänglich ist, dürfte dies im Moment keine wesentliche Einschränkung darstellen - getestet wurde das Programm unter LINUX (gnu g++ v2.7.0) und UNIX (sparc-sun-solaris 2.6, gnu g++ v.2.7.2.3).

Um die folgenden Informationen verständlicher zu machen, halte ich es für sinnvoll, zuerst ein kurzes Beispielprogramm (siehe D.1) vorzustellen, danach die einzelnen Methoden der Schnittstellenklasse (siehe D.2) zu erläutern, um später auf die Syntax aller einzulesenden bzw. einlesbaren Dateien einzugehen (siehe D.3). Zur Verdeutlichung der Merkmalsparameter werden zum Schluß alle implementierten Merkmalsarten vorgestellt (siehe D.4).





## D.1 Beispielprogramm

In diesem kurzen Programm (Programm D-1) wird die Nutzung der Schnittstelle gezeigt. Wenden wir uns gleich dem Hauptprogramm (main()) zu. Die Klasse 'work' ist extern definiert und beinhaltet alle Methoden, welche zur Nutzung des WTA- Netzes nötig sind. Zuerst wird die Struktur der Graphen eingelesen - das sind Informationen darüber, durch welche Merkmale die Knoten und Kanten beschrieben werden. Dies muß unbedingt am Anfang erfolgen, da sonst die Merkmalsausprägungen der einzelnen Graphen nicht richtig zugeordnet werden können.

```
#include <iostream.h>
#include "wta_work.h"

main()
{
   if (!work.leseStruktur("bsp1.graph"))
     {
      cout << "Da ist was schiefgelaufen...\n";
      exit(1);
     }
   if (!work.leseGraphA("bsp1.graph"))
     {
      cout << "Da ist was schiefgelaufen...\n";
      exit(1);
     }
   if (!work.leseGraphB("bsp2.graph"))
     {
      cout << "Da ist was schiefgelaufen...\n";
      exit(1);
     }
   work.compare();
   work.speicherErgebnisGraph("erg.graph");
}
```

*Programm D-1 - main.cpp*

Danach werden im Beispiel zwei verschiedene Graphen eingelesen, welche wiederum verglichen werden. Das Ergebnis dieses Vergleichs wird am Ende gespeichert und kann für neue Vergleiche genutzt werden.

Betrachten wir nun noch kurz die '#include'- Anweisungen am Anfang des Programms. Während das Einbinden von 'iostream.h' nur für die Ausgabeoperationen im Programm main() nötig ist, bindet 'wta_work.h' die eigentliche Schnittstelle ein.

Diese Schnittstelle besteht aus einer Klasse (wta_work, siehe Programm D-2), welche Methoden enthält, mit denen alle wesentlichen Ein-/ Ausgabeoperationen durchgeführt werden können und andere, mit denen die Parameter des WTA- Netzes geändert werden können (Diese wurden im Beispielprogramm nicht genutzt).

```
#ifndef WTA_WORK_H
#define WTA_WORK_H


#ifndef WTA_PARAMS_ENUM
#define WTA_PARAMS_ENUM
enum s_art{lose,fixiert,reduziert};
enum s_was{knot,kant,beide};
enum anf_art{interaktiv,graphGroesse,fest,aehnlichkeit,listenwerte};
enum exI_art{ohne,prozentGraphGroesse,anzahlKanten};
enum kanSk_art{keine,kanten,kanten_u_knoten};
#endif

extern class wta_work work;

class wta_work
{
   class wta_netz *Netz;
   class graph *graphA, *graphB, *ErgGraph;
 public:
   wta_work(void);

// Ein-/ Ausgabemethoden
   bool leseStruktur(char*);
   bool leseGraphA(char*);
   bool leseGraphB(char*);
   void speicherErgebnisGraph(char*);

// Vergleichsmethode
   int compare(void);

// Kontrollmethoden
   void gibAusStruktur(void);
   void gibAusGraphA(void);
   void gibAusGraphB(void);
   void gibAusErgGraph(void);
   void gibAusParams(void);
   void gibAusAnfAktListe(void);

// Methoden zur Parameteraenderung
   void setAnfAktArt(enum anf_art);
   void setInit(float);
   void resetAnfAktListe(void);
   bool addAnfAkt(char*,char*,float);
   void setExtInpArt(enum exI_art);
   void setSkal(float);
   void setKanSkalArt(enum kanSk_art);
   void setSchrittArt(enum s_art);
   void setSchrittWas(enum s_was);
   void setAnzBerech(int);
   void setS(float);
   void setKnotenSchwelle(float);
   void addKnotenSchwelle(float);
   void setKantenSchwelle(float);
   void addKantenSchwelle(float);

   ~wta_work(void);
};

#endif
```

*Programm D-2 - wta_work.h*

## D.2 Methoden der Schnittstellenklasse

### D.2.1 Ein-/ Ausgabemethoden

#### *D.2.1.1 bool work.leseStruktur(char* Dateiname)*

Bevor irgendwelche Graphen eingelesen werden können, müssen dem System Informationen über die Bedeutung und die Anzahl der Merkmale an Knoten und Kanten gegeben werden.



Dies erfolgt einmal für alle Graphen gleicher Struktur durch eine Strukturbeschreibungsdatei (z.B. main.struct, zur Syntax dieser siehe D.3.1). Natürlich ist nur ein Vergleich von Graphen mit gleicher Merkmalstruktur sinnvoll und möglich, als Ergebnis entsteht ein Graph mit eben dieser Merkmalstruktur. Daher bedingt das Einlesen einer neuen Strukturbeschreibung auch das Löschen *aller* vorhandenen Graphen.

Als Parameter wird dieser Methode der Dateiname der Strukturbeschreibung übergeben.

Rückgabewert: true = 1 bei erfolgreicher Ausführung, sonst false = 0.

#### *D.2.1.2 bool work.leseGraphA(char* Dateiname)*
#### *bool work.leseGraphB(char* Dateiname)*

Diese Methoden lesen die Beschreibungen der Graphen ein. Zur Syntax der Graphbeschreibungsdatei sollte Abschnitt D.3.2 beachtet werden.

Rückgabewert: true = 1 bei erfolgreicher Abarbeitung, sonst false = 0 oder Programmabbruch (exit(1)) bei schwerem Fehler.

#### *D.2.1.3 work.speicherErgebnisGraph(char* Dateiname)*

Um das Vergleichsergebnis auch nutzen zu können, muß dieses abgespeichert werden. Die oben genannte Methode dient genau dazu und speichert den Graph als Graphbeschreibungsdatei ab. Diese kann wieder eingelesen werden (als GraphA oder GraphB), wenn die gleiche Strukturbeschreibung wie bei den zu vergleichenden Graphen das System vorher initialisiert hat.

ACHTUNG: Vor dem Abspeichern prüft diese Methode *nicht* den Zugriff auf die Datei. So kann es vorkommen, daß ungewollt ältere Dateien überschrieben werden oder ein Abspeichern wegen falschen Zugriffsrechten oder ungültigen Dateinamen unterbleibt. Es ist daher sinnvoll, mit eigenen Routinen den Dateizugriff zu prüfen.

### D.2.2 Vergleichsmethode int work.compare(void)

Das ist natürlich das Herzstück des Programms, der eigentliche Vergleich findet hier statt. Das bedeutet auch, daß die Struktur und zwei Graphen vorher eingelesen sein müssen. Falls Nicht-Standard- Parameter genutzt werden sollen, müssen diese auch vor Aufruf dieser Methode geändert werden.

Rückgabewert: 0 bei Erreichen des Durchlaufendes (d.h. es wurde keine Lösung gefunden, Durchlaufende ist 10 * Anzahl der Knoten im Kompatibilitätsgraphen), sonst Anzahl der Durchläufe bis zu einem stabilen Zustand des Netzes.

### D.2.3 Kontrollmethoden

Die folgenden Methoden sind nicht unbedingt für einen korrekten Programmablauf nötig. Jedoch ist es (besonders bei unverständlichen Programmabläufen) manchmal sinnvoll, die Übernahme der Dateien oder anderen Informationen ins System zu überprüfen. Alle Methoden in diesem Abschnitt geben bestimmte Informationen auf dem Terminal aus.

#### *D.2.3.1 work.gibAusStruktur(void)*

Diese Methode gibt die Beschreibung der eingelesenen Merkmalstruktur in der Syntax der Markierungsbeschreibungsdatei (siehe Abschnitt D.3.1) aus.

### *D.2.3.2 work.gibAusGraphA(void)*
### *work.gibAusGraphB(void)*
### *work.gibAusErgGraph(void)*

Die Methoden dienen zur Ausgabe der im Namen der Methoden genannten Graphen. Diese werden in der Syntax der Graphbeschreibungsdatei (siehe Abschnitt D.3.2) auf dem Terminal ausgegeben.

### *D.2.3.3 work.gibAusParams(void)*

Durch Aufruf dieser Methode wird auf dem Bildschirm die aktuelle Parametereinstellung in lesbarer Form ausgegeben (Ausgabeprotokoll D-1).

```
keine Selbsthemmung, ( d = 0 )

einzelne Berechnung (kein schrittweises Matching)
   untere AehnlichkeitsSchwelle fuer Knoten = 1
   untere AehnlichkeitsSchwelle fuer Kanten = 1

AnfangsAktivierung der Knoten :
        aus Graphgroesse errechnet

externer Input :
        Kein externer Input vorhanden

KantenSkalierung :
        Kanten nicht skaliert
```

*Ausgabeprotokoll D-1 - Ausgabe von work.gibAusParams() - Parameterinitialeinstellung*

### *D.2.3.4 work.gibAusAnfAktListe(void)*

Falls die Anfangsaktivierungen durch den Inhalt einer Liste bestimmt werden (work.setAnfAktArt(listenwerte) - siehe D.2.4.1) können alle Anfangsaktivierungen dieser Liste mit dieser Methode auf dem Bildschirm ausgegeben werden. Ist eine andere Anfangsaktivierungsart eingestellt, wird die Liste geleert und diese Methode gibt daher nichts aus.

## D.2.4 Methoden zur Parameteränderung

Zum korrekten Ablauf des Programms müssen die Parameter nicht unbedingt geändert werden, die Einstellung nach dem Programmstart ist im Ausgabeprotokoll D-1 zu finden. Zur Verdeutlichung sind im folgenden die Parameter, deren Übergabe an entsprechende Methoden die Default- Einstellungen (wieder-) herstellt, unterstrichen.

ACHTUNG: Wenn eine Netzeinstellung besondere Parameter benötigt, welche vom Programm erfragt werden, heißt dies *nicht*, daß dies unbedingt im Moment der Änderung der Netzeinstellung erfolgen muß. Vielmehr werden einige Informationen erst beim Start des Vergleichs vom Programm anhand der aktuellen Parametereinstellung erfragt.

### *D.2.4.1 work.setAnfAktArt(enum anf_art Art)*

Änderung: Anfangsaktivierung der Netzknoten

*mögliche Parameter für Art (siehe enum anf_art in Programm D-2 - wta_work.h):*

interaktiv — für alle Knoten des Kompatibilitätsgraphs wird *jeweils* nach der Anfangsaktivierung gefragt

graphGroesse — Anfangsaktivierung wird anhand Anzahl der Knoten im Kompatibilitätsgraph berechnet

fest — alle Knoten werden mit dem gleichen Wert initialisiert, welcher durch 'work.setInit(Wert)' (D.2.4.2) festgelegt wird



aehnlichkeit — Anfangsaktivierung wird wie bei 'graphGroesse' berechnet und zusätzlich für jeden Knoten mit dessen Ähnlichkeit gewichtet.

listenwerte — Den Neuronen wird eine Anfangsaktivierung zugeordnet, falls diese anhand der repräsentierten Knotenzuordnungen in einer Liste gefunden wird - ansonsten wird eine Anfangsaktivierung von 0 zugeordnet. (siehe D.2.4.3 und D.2.4.4)

### *D.2.4.2 work.setInit(float Wert)*

legt den Wert fest, welcher bei Anfangsaktivierungsart 'fest' für alle Knoten übernommen wird. (default 0.1).

### *D.2.4.3 work.addAnfAkt(char* A, char* B, float Wert)*

Wurde die Anfangsaktivierungsart auf 'listenwerte' eingestellt (work.setAnfAktArt(listenwerte) - siehe D.2.4.1) werden alle Anfangsaktivierungen für die Neuronen des Netzes aus einer Liste entnommen - der Inhalt dieser Liste wird mit dieser Methode bestimmt. Dabei ist A ein Knotenbezeichner aus dem GraphA und B einer aus dem GraphB. Wert enthält die Anfangsaktivierung, welche dem Neuron, welches die Kombination aus A und B repräsentiert, zugeordnet wird. Die übergebenen Strings werden intern kopiert und können bei Bedarf sofort nach Aufruf dieser Methode gelöscht werden. (siehe auch D.2.4.4)

Soll also dem Neuron, welches Knoten 'X' aus GraphA und Knoten 'Y' aus GraphB repräsentiert, eine Anfangsaktivierung von 0.75 zugeordnet werden, kann dies durch 'work.addAnfAkt("X","Y",0.75);' erfolgen.

### *D.2.4.4 work.resetAnfAktListe(void)*

Diese Methode löscht die Elemente der Liste von Neuronenaktivierungen (siehe D.2.4.3)

### *D.2.4.5 work.setExtInpArt(enum exI_art Art)*

Änderung: externer Input der Netzknoten, ändert die Wichtigkeit der Knoten im Vergleich zu Kanten

*mögliche Parameter für Art (siehe enum exI_art in Programm D-2 - wta_work.h):*

ohne — kein externer Input

prozentGraphGroesse — ein Knoten ist genauso wichtig wie soviel Prozent der Kantenanzahl des Kompatibilitätsgraphes, wie mit 'work.setSkal(Wert)' eingestellt werden kann

anzahlKanten — ein Knoten ist genauso wichtig wie die Anzahl Kanten, welche mit 'work.setSkal(Wert)' angegeben werden kann

### *D.2.4.6 work.setSkal(float Wert)*

legt den Wert fest, welcher für externen Input (bei Einstellung 'prozentGraphGroesse' bzw. 'anzahlKanten') als jeweiliger Parameter genutzt wird.

### *D.2.4.7 work.setKanSkalArt(enum kanSk_art Art)*

Änderung: Art der Kantenskalierung, also die Beeinflussung der an einem Knoten eingehenden Potentiale durch den Kanten zuzuordnende Werte.

*mögliche Parameter für Art (siehe enum kanSk_art in Programm D-2 - wta_work.h):*

keine — alle eingehenden Potentiale werden nicht (d.h. mit 1) skaliert

kanten — alle eingehenden Potentiale werden mit der Ähnlichkeit der jeweiligen Kante skaliert

kanten_u_knoten — die eingehenden Potentiale werden mit der Ähnlichkeit der jeweiligen Kante und der Ähnlichkeiten der Knoten, welche die Kante verbindet, gewichtet

#### *D.2.4.8 work.setSchrittArt(enum s_art Art)*

Änderung: Behandlung der Ergebnisknoten beim schrittweisen Matching. (Dies ist zum Beispiel durch Erhöhen der Berechnungszahl mit work.setAnzBerech(Wert) einstellbar). Dabei werden mehrere Berechnungen durchgeführt, wobei die Ähnlichkeitsschwelle, die übertroffen werden muß, um in den Kompatibilitätsgraph übernommen zu werden, jeweils gesenkt wird.

*mögliche Parameter für Art (siehe enum s_art in Programm D-2 - wta_work.h):*

lose — die vorherige Lösung stellt die Anfangserregung für den neuen Berechnungsschritt dar, in jedem Schritt kommen neue Knoten und / oder Kanten hinzu

fixiert — nach jedem Berechnungsschritt werden Knoten, welche nicht zur Lösung gehören, aus dem Graph entfernt, Knoten der Lösung werden auf ihrem aktuellen Wert (1) festgehalten. Dies dient dann als Anfangserregung für die nächsten Berechnungsschritte.

reduziert — nach jedem Berechnungsschritt werden Knoten, welche nicht zur Lösung gehören, entfernt, danach werden mit neuer Ähnlichkeitsschwelle neue Knoten bzw. Kanten hinzugefügt und alle Knoten neu mit einer Anfangsaktivierung gleich initialisiert.

#### *D.2.4.9 work.setSchrittWas(enum s_was Art)*

Änderung: Falls das schrittweise Matching durch die Angabe der maximalen Berechnungszahl eingestellt wurde, kann hiermit festgelegt werden, was schrittweise gematcht wird, also welche Ähnlichkeitsschwelle/ -n (Knoten u./o. Kanten) schrittweise gesenkt werden.

*mögliche Parameter für Art (siehe enum s_was in Programm D-2 - wta_work.h):*

knot — Knotenschwelle wird schrittweise gesenkt

kant — Kantenschwelle wird schrittweise gesenkt

beide — beide Schwellen werden schrittweise gesenkt

#### *D.2.4.10 work.setAnzBerech(int Wert)*

legt die Anzahl der Berechnungsschritte fest, die ausgeführt werden, um zur untersten Ähnlichkeitsschwelle zu gelangen. Dies ist eine Möglichkeit, schrittweises Matching einzustellen. Eine andere ist es, für eine Berechnung mehrere Schwellen anzugeben. (siehe D.2.4.13 bzw. D.2.4.15) Voreingestellt ist eine einzelne Berechnung (Berechnungsanzahl = 1).

#### *D.2.4.11 work.setS(float Wert)*

stellt die Größe der Selbsthemmung $d$ der Neuronen des Netzes ein ($d = s * w$, als Wert wird $s$ übergeben). Die Voreinstellung ist 0, die Selbsthemmung ist damit abgeschaltet.

#### *D.2.4.12 work.setKnotenSchwelle(float Wert)*

Diese Methode dient zur Festlegung der unteren Knotenschwelle, die nach Durchführung aller Berechnungen (siehe work.setAnzBerech(X)) erreicht wird. Bei der Bildung des Kompatibilitätsgraphes werden alle Knoten, welche als Vergleichsergebnis Werte über der aktuellen Knotenschwelle haben, in den Graphen aufgenommen.

Der voreingestellte Wert ist 1.



#### *D.2.4.13 work.addKnotenSchwelle(float Wert)*

Durch diese Methode können zur bisher eingestellten unteren Knotenschwelle weitere Knotenschwellen hinzugefügt werden. Sind mehrere Knotenschwellen im System, so werden diese mit der größten beginnend zum schrittweisen Matching genutzt - die Berechnungsanzahl nach D.2.4.10 wird dabei ignoriert.

#### *D.2.4.14 work.setKantenSchwelle(float Wert)*

Dies ist die adäquate Methode zu setKnotenSchwelle, jedoch für die Schwelle, über der Kanten zum Kompatibilitätsgraph hinzugefügt werden. Auch hier ist ein Wert von 1 voreingestellt.

#### *D.2.4.15 work.addKantenSchwelle(float Wert)*

Dies ist die adäquate Methode zu addKnotenSchwelle - es können zusätzliche Kantenschwellen hinzugefügt werden.

## *D.3 Bedeutung und Syntax der einlesbaren Dateien*

Da alle einzulesenden Dateien durch eine Klasse (readfile) bzw. deren Unterklassen eingelesen werden, sind in *allen* Dateien die folgenden Zeichen als Sonderzeichen zu betrachten, ganz gleich, ob diese nun Graphen oder Hierarchien oder geordnete Mengen beschreiben (Tabelle D-1).

| Zeichen | Bedeutung |
|---|---|
| **Zeichen mit Wortendekennung:** | |
| ' ' | Worttrennung |
| '(' | Parameteranfang |
| '/' | Merkmalstrennung bei Kantenmerkmalen |
| ':' | Knoten- bzw. Kantenbezeichnerende |
| cr = (int) 10 | Return = Zeilenende |
| **Zeichen ohne Wortendekennung:** | |
| ')' | Parameterende |
| '{' | Konzeptanfang |
| '}' | Konzeptende |
| '[' | Intervallanfang |
| ']' | Intervallende |
| '#' | Kommentaranfang bzw. -ende |

*Tabelle D-1 - Sonderzeichen- Übersicht*

Kommentare können in allen einlesbaren Dateien vorkommen, diese müssen in '#' eingeschlossen sein. Oder anders gesagt, alle Zeichen innerhalb zweier '#' werden als Kommentar angesehen und darum ignoriert.

ACHTUNG: Die Knotenbezeichner 'Knoten:' und 'Kante:' (siehe D.3.2.1) sind als Schlüsselwörter für die Markierungsbeschreibungen reserviert (siehe folgenden Abschnitt). Dies bedeutet, daß einerseits keinerlei Knoten den Name 'Knoten' bzw. 'Kante' haben dürfen ('KnotenA' ist jedoch erlaubt), andererseits kann die Markierungsbeschreibung somit auch am Anfang (!) einer Graphbeschreibungsdatei stehen. In diesem Fall muß die Gesamtbeschreibungsdatei zuerst als Markierungsbeschreibung und danach als Graphbeschreibung eingelesen werden.

### D.3.1 Markierungsbeschreibungsdatei

Bevor irgendwelche Graphen miteinander verglichen werden können, muß dem Programm mitgeteilt werden, welche Merkmale in welcher Reihenfolge den Graph beschreiben und wie diese verglichen werden sollen. Diese Aufgabe übernimmt die Markierungsbeschreibungsdatei, welche im folgenden beschrieben werden soll.

Zur Gewinnung einer größeren Übersichtlichkeit werden die genauen Verfahren, mit denen die verschiedenen Merkmalsausprägungen verglichen werden, erst im nächsten Teil, Abschnitt D.4, beschrieben. Im Moment reicht es zu wissen, daß jeder Merkmalsart eine Maximalzahl an Parametern zugeordnet ist, von denen nur der erste bei allen Merkmalen gleiche Bedeutung hat. Aber dazu später mehr.

Die Markierungsbeschreibungsdatei muß die Schlüsselwörter 'Knoten:' und 'Kante:' in dieser Reihenfolge enthalten und kann auf diese Schlüsselwörter folgend beliebig viele Merkmalsbeschreibungen enthalten, welche durch Komma und/oder Return voneinander getrennt werden müssen.

| Merkmal-bezeichner | Maximalzahl Parameter | Merkmalsausprägung |
|---|---|---|
| ganz_1 | 2 | ganzzahlig; statistischer Vergleich |
| ganz_2 | 4 | ganzzahlig; 'fuzzy'- Vergleich |
| reell_1 | 2 | reell; statistischer Vergleich |
| reell_2 | 4 | reell; 'fuzzy'- Vergleich |
| symbol | 1 | Symbole |
| begriff | 4 | Begriffe aus einer Hierarchie |
| gmenge_1 | 3 | Begriffe aus einer geordneten Menge; statistischer Vergleich |
| gmenge_2 | 5 | Begriffe aus einer geordneten Menge; 'fuzzy'- Vergleich |

*Tabelle D-2 - implementierte Merkmalsarten*

Eine Merkmalsbeschreibung besteht aus einem Merkmalsbezeichner (siehe Tabelle D-2) und optional darauf folgend einer Liste von Parametern in Klammern '(' bzw. ')'.

Der erste Parameter entspricht dem Gewicht des Merkmals, mit welchem dieses in die Gesamtbewertung für den Knotenvergleich bzw. die Kantenvergleiche eingeht. Ist dieser größer als 1 oder wird er weggelassen, wird bei einem Einzelvergleichsergebnis ungleich 1 das Gesamtvergleichsergebnis auf 0 gesetzt, was bedeutet, daß für genau dieses Merkmal Identität verlangt wird.

Parameter können weggelassen werden, sofern sie nicht unbedingt für den Programmablauf nötig sind (und keine folgenden Parameter übergeben werden sollen oder müssen).

Knotenmerkmale sind generell ungerichtet (wohin auch?), Kantenmerkmale können jedoch auch gerichtet sein. Daher werden bei der Beschreibung der Merkmale für Kanten zunächst die ungerichteten Merkmale angegeben, nach dem Schlüsselzeichen '/' jedoch Merkmalsbeschreibungen für gerichtete Merkmale angegeben.

ACHTUNG: Während die Merkmalsausprägungen natürlich in jede Richtung verschieden sein können, müssen alle gerichteten Gesamtmerkmale identisch strukturiert sein, weshalb natürlich die Struktur nur für eine Richtung angegeben werden muß (und darf).

Zur Verdeutlichung nun einige mögliche Markierungsbeschreibungen:

### *D.3.1.1 Beispiel 1*

```
Knoten:                 # SchluesselWort fuer KnotenMerkmale #
Kante:                  # SchluesselWort fuer KantenMerkmale #
```

*Programm D-3 - einfachste Markierungsbeschreibung (Kommentare in '#' können weggelassen werden)*

Die in Programm D-3 angegebene Struktur hat keinerlei Merkmale, was bedeutet, daß jeder Knoten mit jedem anderen Knoten und jede Kante mit jeder anderen Kante matcht.

eine dazu passende Knotenbeschreibung:

```
A:
```



eine dazu passende Kantenbeschreibung:

```
A,B:
```

### *D.3.1.2 Beispiel 2*

```
# Strukturbeschreibung #

Knoten  :               #Schluesselwort fuer KnotenBeschreibung  #
ganz_2(0.8,4,0,7)
begriff(0.2,testhier,Teil)
reell_1(0.3,3)

Kante:                  # Schluesselwort fuer KantenBeschreibung #
symbol(0.8,8)           # ungerichtetes Merkmal                  #
gmenge_1(0.5,testgmenge,4)
/                       # Trennzeichen                           #
ganz_1(0.7)             # gerichtetes Merkmal                    #
```

*Programm D-4 - komplexere Markierungsbeschreibung*

eine dazu passende Knotenbeschreibung wäre:

```
A:(2,5,4),Linde,3   # KnotenName A                                #
```

eine dazu passende Kantenbeschreibung wäre:

```
A,B:rot,(drei,vier)/1/2    # Kante zwischen A und B, das erste ungerichtete     #
                           # Merkmal hat die Auspraegung rot, das zweite unge-  #
                           # richtete Merkmal hat die Auspraegung (drei,vier),  #
                           # gerichtetes Merkmal von A nach B hat Auspraegung 1 #
                           # und das von B nach A hat Auspraegung 2             #
```

## D.3.2 Graphbeschreibungsdatei

Jeder Graph wird natürlich auch mit einer Datei beschrieben. Ein Graph besteht aus Knoten und Kanten, denen jeweils verschiedene Merkmale zugeordnet sind. Die Reihenfolge von Knoten- und Kantenbeschreibungen ist beliebig, da die Datei in zwei Durchläufen (erst die Knoten und dann die Kanten) eingelesen wird.

### *D.3.2.1 Knotenbeschreibung*

Betrachten wir zuerst die eben (siehe D.3.1.2) gegebene Knotenbeschreibung in Zusammenhang mit der dort gegebenen Markierungsbeschreibung.

```
A:(2,5,4),Linde,3   # KnotenName A                                #
```

Hierbei handelt es sich um den Knoten mit dem Name A (Dieser wird gespeichert und z.B. für das Finden der Kanten gematcht), welchem verschiedene Merkmalsausprägungen zugeordnet sind. Gleich am Anfang ist die Ausprägung ein Konzept (siehe dazu auch D.4.1) daß heißt dem Merkmal ('ganz_2') sind mehrere einzelne ganze Zahlen zugeordnet.

Ein Konzept wird durch '{' eingeleitet und durch '}' abgeschlossen. Im obigen Beispiel werden also die drei ganzen Zahlen 2, 5 und 4 dem Merkmal 'ganz_2' zugeordnet.

Zusätzlich können bei einigen Merkmalsarten (z.B. 'ganz_2') auch Intervalle angegeben werden, welche durch '[' eingeleitet und durch ']' abgeschlossen werden. Intervalle müssen immer innerhalb von Konzepten auftreten. (Beispiele für Intervalle: {[1,5]},{1,2,3,[4,5],6,7,8})

Das zweite Merkmal ist von der Art 'begriff', ist also ein Element einer Begriffshierarchie. Dem Merkmal wird nun der Begriff 'Linde' zugeordnet. Falls das Wort Linde nicht in der speziellen, durch die Markierungsstruktur benannten Begriffshierarchie auftreten würde, würde das Programm eine Warnung ausgeben und beim Vergleich dieses Merkmals mit einem anderen mit einer Fehlermeldung abbrechen.

obwohl es sich ja nicht um 5 handeln soll, sondern um das Resultat aus dem Vergleich von 3 und 7, was wohl weniger genau der Ausprägung 5 ähnlich sein soll als die 5 sich selbst.

Aus dem eben geschilderten Grund erschien es nötig, als Ergebnis des Vergleichs von 3 und 7 nicht 5, sondern {3,7} anzugeben, also eine Menge aus mehreren (im Beispiel den bisher verglichenen) Zahlen.

Wie wir aber an dieser Stelle sehen können, kann unter bestimmten Bedingungen ein Unterschied zwischen der Ergebnismenge selbst ({3,7}) und der Beschreibung dieser (5) bestehen, da ja auch der Mittelwert der Menge eine Beschreibung derselben darstellt.

Darum soll die *Beschreibung* der Ergebnismenge (ob durch Aufzählung der Elemente oder durch andere beschreibende Methoden) im folgenden als Konzept bezeichnet werden. Im Gegensatz dazu stehen die Ausgangsmerkmalsausprägungen als Objekte.

### D.4.2 Vergleichsmethoden für die verschiedenen Merkmalsarten

Um Ähnlichkeiten zwischen Merkmalen in einem Computer zu verarbeiten, müssen diese zuerst auf Zahlen abgebildet werden. Diese recht simpel klingende Feststellung birgt jedoch einige Probleme in sich, da wir Menschen normalerweise Ähnlichkeit sehr einfach feststellen, ja sogar den Grad der Ähnlichkeit ganz gut einschätzen können. Sollen diese Ähnlichkeitswerte nun auf exakte Zahlen zwischen 0 (maximal unähnlich) und 1 (identisch) abgebildet werden, treten Probleme auf. Zahlen sind zu fest, und Ähnlichkeiten als unscharfe, kontextabhängige Beziehungen auf feste Werte zu reduzieren bringt Schwierigkeiten mit sich - während eine Ähnlichkeit von 0.5 vielleicht noch verstanden werden kann, ist dies bei der Ähnlichkeit 0.762 schon etwas komplizierter.

Warum diese Erklärung?

Da ich die Merkmalsquantifizierung nicht für eine bestimmte Aufgabenstellung vornehmen sollte, mußte ich versuchen, allgemeine (das heißt Alltags-) Ähnlichkeiten in Zahlen umzusetzen. Da für die Ähnlichkeit von Begriffen aus Begriffshierarchien zum Beispiel keine typischen Quantisierungsvorgaben existieren, spiegeln die verwendeten Verfahren natürlich auch immer *meine* Vorstellung der Anwendung dieser Vergleiche wieder. Sollten also einige der folgenden Verfahren für ein ganz spezielles Problem nicht geeignet sein, so kann ich nur empfehlen, die implementierten Versionen zu verändern.

#### *D.4.2.1 reellwertige Merkmale ('reell_*')*

Obwohl, wie bereits in Abschnitt 2.3.1 erwähnt, zwei Implementierungen für die Quantisierung von reellwertigen Merkmalsausprägungen existieren, unterscheiden diese sich lediglich im Umgang mit Konzepten. Sollen einzelne Zahlen miteinander verglichen werden, ist ein statistischer Ansatz nicht zu verwenden und auch das statistische Verfahren nutzt an dieser Stelle den im folgenden erklärten 'fuzzy'- Vergleich.

##### D.4.2.1.1 Objekt- Objekt- Vergleich:

Sollen zwei Zahlen miteinander verglichen werden, steht fest, daß bei Gleichheit beider Zahlen die Ähnlichkeit mit 1 (also Identität) bestätigt werden muß.

Um aber Ähnlichkeit zum Ausdruck zu bringen, muß eine Zahl , welche nicht identisch der anderen ist, dieser trotzdem ähnlich sein (können). Diese Ähnlichkeit sollte sich verringern, je größer der Abstand beider Zahlen ist. Um den gesamten Vorgang überschaubarer zu halten und den Anwender bzw. die Anwenderin bei der Parameterwahl zu unterstützen, empfiehlt sich eine lineare Abnahme der Ähnlichkeit, wie sie mit Formel D-1. im Programm implementiert wurde (HardOr - Vergleich).



$$\gamma = \gamma(x_a, x_b) = \varphi[1 - abs(\frac{x_a - x_b}{l})], \quad \varphi(x) = \begin{cases} 0 & x < 0 \\ x & \text{sonst} \end{cases}, \qquad \text{(D-1.)}$$

Ein Beispielvergleich von $x$ mit 5 ist in Abbildung D-2 dargestellt.

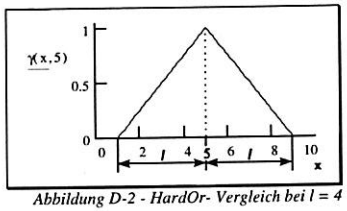



*Abbildung D-2 - HardOr- Vergleich bei l = 4*

$l$ ist der Ähnlichkeitsabstand (im Beispiel l = 4), welcher bestimmt, auf welche reelle Entfernung die Ähnlichkeit der beiden Zahlen von 1 bis auf 0 absinkt. Dieser kann als Parameter dem Programm übergeben werden und muß also von der Nutzerin bzw. dem Nutzer selbst anhand des Problems gewählt werden. (siehe Tabelle D-3 - Merkmalsparameter)

Die Umgangsweise mit Konzepten ist an dieser Stelle abhängig von der Betrachtung der Daten. Können diese als auf einen Wert weisend angenommen werden, so läßt sich aus allen Elementen des Konzeptes eine Normalverteilung ermitteln. Sind verschiedene Elemente aber auch als solche zu berücksichtigen, ist ein Vergleich über Fuzzy- Methoden eher angebracht.

##### D.4.2.1.2 statistisches Vergleichsverfahren (reell_1)

Bei diesem Verfahren werden alle Objekte (auch die innerhalb von Konzepten), welche jemals zum Vergleich angeboten wurden, im Vergleichsergebnis mitgeführt. Der Vorteil dieser Methode ist die mögliche Berechnung einer einzelnen Zahl aus allen Werten des Ergebnisses (z.B. Mittelwert), wobei *dann* das Resultat die gleiche Qualität hat wie die Ausgangsobjekte (einzelne Zahlen).

Ein weiterer Vorteil dieser Methode liegt in der Berücksichtigung der Gesamtverteilung der Elemente der Konzepte bei den Ähnlichkeitsquantisierungen - im Gegensatz zu 'fuzzy'- Methoden, bei welchen nur die näheren Objekte Einfluß auf das Ergebnis haben. Dies ist natürlich nur dann sinnvoll, wenn die Elemente auch als Verteilung betrachtet werden können.

###### *D.4.2.1.2.1 Objekt- Konzept- Vergleich*

Es sei $x_a$ das Objekt und $\{x_1, x_2, \ldots, x_n\}$ das Konzept, welche verglichen werden sollen.

Da die Elemente des Konzepts als auf eine Zahl hin weisend angenommen werden, werden aus dem Konzept zuerst der Mittelwert $\bar{x}$ und die Varianz $s^2$ berechnet, da angenommen wird, daß die Elemente im Konzept normalverteilt sind.

Berechnung der Varianz: $s^2 = \frac{1}{n-1} * \sum_{i=1}^{n} (x_i - \bar{x})$ (D-2.)

Nun sind ein Mittelwert eines Konzeptes und eine Zahl miteinander zu vergleichen. Wie eben schon angenommen wurde, soll das Konzept eine Normalverteilung beschreiben. Eine Ähnlichkeit kann damit mit der Wahrscheinlichkeit des Auftretens von $x_a$ in der durch Mittelwert und Varianz beschriebenen Verteilung beschrieben werden.

Berechnung der Ähnlichkeit: $y = e^{-\frac{1}{2} * \frac{(x_a - \bar{x})^2}{s^2}}$ (Normalverteilung, Abbildung D-3) (D-3.)

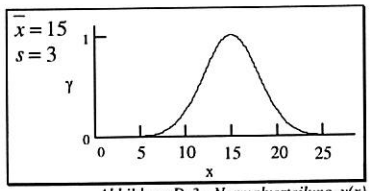


*Abbildung D-3 - Normalverteilung, y(x)*

#### *D.4.2.1.2.2 Konzept- Konzept- Vergleich:*

Es seien A und B zwei Konzepte, deren Mittelwerte ($\overline{x_A}, \overline{x_B}$) und Varianzen ($s_A^2, s_B^2$) berechnet wurden.

Fest steht, daß bei gleichen Verteilungen ($\overline{x_A} = \overline{x_B}$ und $s_A^2 = s_B^2$) die Identität der Konzepte mit 1 bestätigt werden muß. Sind die Verteilungsparameter verschieden, so soll die Ähnlichkeit abnehmen - weil diese Abnahme schon beim Objekt- Konzept- Vergleich (D.4.2.1.2.1) exponentiell erfolgte, ist es an dieser Stelle ebenfalls sinnvoll, eine exponentielle Abnahme der Ähnlichkeit vorzusehen.

Die eben beschriebenen Anforderungen lassen sich durch folgende Formel realisieren:

$$\gamma' = e^{-(\overline{x_A}-\overline{x_B})^2-(s_A-s_B)^2} \qquad \text{(D-4.)}$$

Ein weiteres Problem ist die Dimensionsabhängigkeit der ermittelten Ähnlichkeit. Es muß sichergestellt werden, daß gleiche Ähnlichkeiten errechnet werden, falls die Elemente in beiden Konzepten in eine andere Dimension verschoben (zum Beispiel verzehnfacht) werden. Dies konnte durch Skalierung der Mittelwertdifferenz mit der Summe der Standardabweichungen (Standardabweichung $= s = \sqrt{s^2} = \sqrt{\text{Varianz}}$) und umgekehrt erfolgen.

$$\gamma = e^{-\left(\frac{\overline{x_A}-\overline{x_B}}{s_A+s_B}\right)^2-200*\left(\frac{s_A-s_B}{s_A+s_B}\right)^2} \qquad \text{(D-5.)}$$

Der Faktor '200' bestimmt das Verhältnis des Einflusses der Differenzen der Mittelwerte zu dem der Differenzen der Standardabweichungen und wurde empirisch ermittelt (siehe Abbildung D-4).

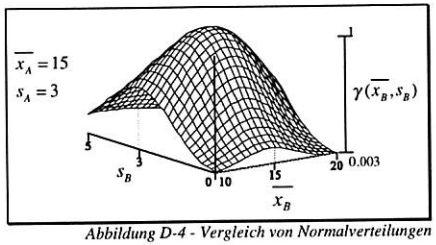


*Abbildung D-4 - Vergleich von Normalverteilungen*

### D.4.2.1.3 fuzzy- Vergleich ('reell_2')

Ist anhand der Daten vorgegeben oder erwünscht, statistische Verfahren zur Ähnlichkeitsbestimmung auszuschließen, können die nun folgenden fuzzy- Verfahren genutzt werden.



#### *D.4.2.1.3.1 Objekt- Konzept- Vergleich:*

Beim Objekt- Konzept- Vergleich ist an dieser Stelle festgelegt, daß, falls das Objekt bereits Element des Konzeptes ist, eine Ähnlichkeit von 1 festgestellt wird.

Dabei ist es wichtig, daß das Vergleichsergebnis von 1 hier nicht etwa Identität bescheinigt, sondern lediglich die größte bei einem solchen Vergleich mögliche Identität feststellt. Andererseits liegt dieser Gegensatz schon im Prinzip von Konzepten begründet, da nicht explizit festgelegt wurde (und werden sollte), ob diese eher als Sammlung von Einzelobjekten oder als Gesamtheit betrachtete werden sollen. Für fuzzy- Vergleiche sollen also im folgenden Konzepte als Sammlung derjenigen Einzelobjekte betrachtet werden, die mit diesem Konzept die Ähnlichkeit 1 haben.

Für den Fall, daß das Einzelobjekt nicht im Konzept auftritt, wird die folgende Berechnung genutzt, wobei für Vergleiche von Objekten wiederum die Berechnung nach Abschnitt D.4.2.1.1 verwendet wird.:

Es sei $x$ das Objekt und $A = \{y_1,...\}$ das Konzept, welche verglichen werden sollen.

$$\gamma(\{y_1,y_2\},x) = \gamma(y_1,x)+\gamma(y_2,x)-\gamma(y_1,x)*\gamma(y_2,x)$$

$$\gamma(\{y_1,y_2,...\},x) = \gamma(y_1,x)+\gamma(\{y_2,...\},x)-\gamma(y_1,x)*\gamma(\{y_2,...\},x) \qquad \text{(D-6.)}$$

Diese Ähnlichkeitsberechnung wird aufgrund ihrer weicheren Übergänge als Soft- Or bezeichnet (Abbildung D-5). Sie ergibt für den Fall, daß Objekte im Konzept enthalten sind eine Identität von 1, weshalb dieser Fall nicht gesondert betrachtet werden muß.

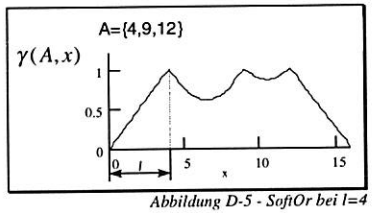


*Abbildung D-5 - SoftOr bei l=4*

#### *D.4.2.1.3.2 Konzept- Konzept- Vergleich:*

Es seien A und B die zu vergleichenden Konzepte. Dann kann eine Ähnlichkeit dieser Konzepte durch einen Flächenvergleich der eben beschriebenen Soft- Or Funktionen quantifiziert werden.

$$\gamma(A,B) = \frac{\int_{-\infty}^{\infty} \min(\gamma(A,x),\gamma(B,x))}{\max\left(\int_{-\infty}^{\infty}\gamma(A,x), \int_{-\infty}^{\infty}\gamma(B,x)\right)} \qquad \text{(D-7.)}$$

Die Fläche unter den minimalen Funktionswerten der Soft- Or Funktionen wird hierbei ins Verhältnis zur maximalen Fläche unter einer der beiden Soft- Or Funktionen für beide Konzepte gesetzt. Diese Ähnlichkeit ist nur dann maximal, wenn beide Konzepte die gleichen Elemente aufweisen, da nur dann Dividend und Divisor gleich groß sind (Abbildung D-6).

Die Flächenberechnung erfolgt programmintern durch lineare Approximation der Funktionen. Die Anzahl der Stützstellen der Berechnung in $l$ ist mit 5 voreingestellt und kann durch Parameter geändert werden (siehe Tabelle D-3 - Merkmalsparameter).

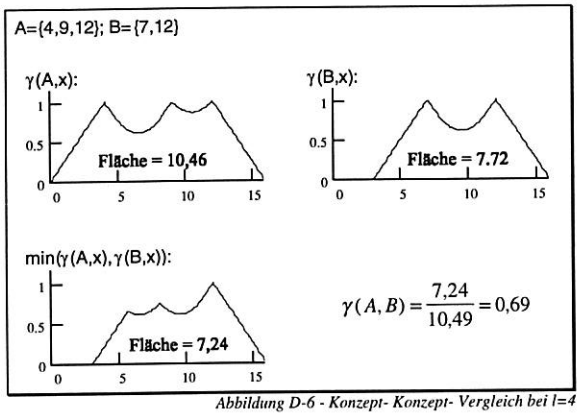



*Abbildung D-6 - Konzept- Konzept- Vergleich bei l=4*

### *D.4.2.2 ganzzahlige Merkmale ('ganz_*')*

Die eben erläuterten Verfahren zum Quantifizieren der Ähnlichkeit von reellen Zahlen bzw. Konzepten eignen sich auch hervorragend, um die Ähnlichkeit ganzer Zahlen bzw. Konzepte zu ermitteln. Lediglich die Menge der möglichen Argumente der genutzten Berechnungsfunktionen verringert sich auf die Menge der ganzen Zahlen. Da die Zahlen als solche im Ergebnis weitergeführt werden, ist nicht einmal hierbei eine exklusive Berücksichtigung ganzer Zahlen nötig.

Im Programm wurde an dieser Stelle einfach eine template[1] - Klasse implementiert, welche einmal für ganze und einmal für reelle Zahlen compiliert wurde.

### *D.4.2.3 Quantifizierung der Ähnlichkeit von Elementen geordneter Mengen ('gmenge_*')*

Die Elemente einer gerordneten Menge lassen sich einfach ganzen Zahlen zuordnen, indem zum Beispiel allen Elementen eine fortlaufende, sich erhöhende Nummer zugeordnet wird- also alle Elemente mit einem Index versehen werden.

Programmintern werden daher für die Repräsentation und den Vergleich der Elemente der geordneten Menge lediglich deren Indizes genutzt, bei einer Ausgabe werden den Indizes wieder die Elementbeschreibungen zugeordnet. Daher erfolgt der Vergleich von Elementen geordneter Mengen und dessen Quantifizierung genau wie der ganzer Zahlen, mit denselben Methoden und Vorschriften wie in (D.4.2.2). Da ganze Zahlen wiederum wie reelle Zahlen behandelt werden, sollten die Verfahren nach (D.4.2.1) ausreichend Informationen geben. Alle dort erwähnten Parameter werden um eine Stelle verrückt, da als 2. Parameter erst der Dateiname der geordneten Menge genannt werden muß. Der 2. Parameter bei 'reell_1' entspricht daher dem 3. Parameter bei 'gmenge_1'.

Aus der oben beschriebenen Abbildung auf ganze Zahlen ergibt sich natürlich auch, daß geordnete Mengen wiederum anhand statistischer Verfahren ('gmenge_1') oder anhand fuzzy-Methoden ('gmenge_2') verglichen werden können.

[1] template- Methoden (generische Methoden) bieten die Möglichkeit, Methoden, welche einmal für bestimmte Klassen geschrieben wurden, für (prinzipiell) alle Klassen zu nutzen, falls diese Methoden für die neuen Klassen compiliert werden und alle Methoden, welche innerhalb dieser Methoden auf die neuen Klassen zugreifen, auch für diese neuen Klassen existieren (zum Beispiel durch überlagerte Operatoren)



### *D.4.2.4 Merkmal symbol ('symbol')*

Das hier beschriebene Vergleichsverfahren dient dem Vergleich beliebiger Symbole.

In diesem Falle gelten Symbole als gleich, wenn die Reihenfolge und Art der Zeichen in beiden zu vergleichenden Symbolen gleich ist. Weitere Zusammenhänge zwischen den Symbolen existieren nicht oder sind nicht bekannt (bzw. werden ignoriert) und können damit keinen Einfluß auf die Ähnlichkeit haben. Weil die als Vergleichsergebnis auftretenden Konzepte sich von den Objekten unterscheiden, müssen diese natürlich auch als Vergleichs'elemente' berücksichtigt werden.

Um hierbei den Gesamtablauf durchschaubarer zu machen, wird in den folgenden Abschnitten immer noch kurz die Generalisierung der Symbole erwähnt, weil diese natürlich immer Rückschlüsse auf die zu vergleichenden Elemente liefert.

#### D.4.2.4.1 Objekt- Objekt- Vergleich:

Da, wie eben erwähnt, keine weiteren Informationen über die Symbole berücksichtigt werden, kann beim Vergleich zweier Symbole nur auf Gleichheit oder Ungleichheit entschieden werden.

Falls die Symbole gleich sind, wird eine 1 als Ähnlichkeit ausgegeben, sonst eine 0. Das Ergebnis der Generalisierung ist ein Konzept mit beiden Symbolen, wenn diese unterschiedlich waren. Waren beide Objekte gleich, wird das Objekt als Generalisierung weitergegeben.

#### D.4.2.4.2 Objekt- Konzept- Vergleich:

Es sei s ein Objekt und A ein Konzept mit Symbolen. Dann wird eine Ähnlichkeit von 1 festgestellt, wenn s in A gefunden werden kann. Sonst beträgt die Ähnlichkeit 0. Das Ergebnis der Generalisierung ist ein Konzept mit allen Symbolen je einmal.

Dies entspricht im übrigen der Herangehensweise beim Quantifizieren von reellwertigen Merkmalen nach dem fuzzy- Verfahren, wo bei Objekt- Konzept- Vergleichen (D.4.2.1.3.1) ebenfalls eine Ähnlichkeit von 1 bescheinigt wurde, wenn das Objekt im Konzept auftrat.

#### D.4.2.4.3 Konzept- Konzept- Vergleich:

Sind zwei Konzepte mit Symbolen zum Vergleich gegeben, stehen sofort einige Randbedingungen fest. Natürlich muß die Ähnlichkeit bei gleichen Symbolen in beiden Mengen 1 betragen.

Daher kann zur Bewertung das Verhältnis von identischen Symbolen in beiden Mengen zur Anzahl der Symbole in den Mengen genutzt werden. Da die Mengen der Symbole natürlich verschieden groß sein können, ist es sinnvoll, für diese Anzahl der Symbole in den Mengen immer die maximale Anzahl zu nutzen.

Seien nun A und B zwei Konzepte mit Symbolen. Dann wird die Ähnlichkeit der Konzepte nach folgender Formel bestimmt:

$$\gamma = \frac{match}{anz}, \qquad \text{(D-8.)}$$

wobei *match* die Anzahl der übereinstimmenden Symbole und *anz* die maximale Anzahl von Objekten in A und B ist.

### *D.4.2.5 Merkmal begriff ('begriff')*

Zum Abschluß soll nun noch daß Vergleichsverfahren für Begriffe aus einer Hierarchie beschrieben werden.

Da bei Hierarchien Begriffe durch Verallgemeinerung zu Oberbegriffen zusammengefaßt werden, entstehen als Generalisierung immer wieder einzelne Begriffe. Die Konzepte haben somit die gleiche Qualität wie die Objekte und müssen daher beim Vergleich nicht gesondert berücksichtigt werden.

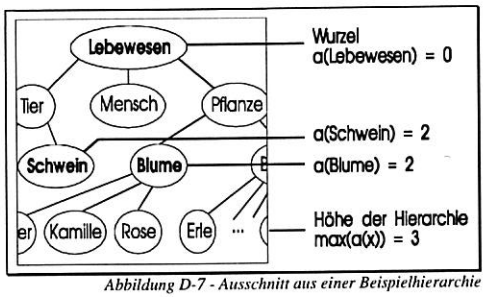



*Abbildung D-7 - Ausschnitt aus einer Beispielhierarchie*

Für den Vergleich wird jeder Begriff mit seiner Entfernung $a$ (Anzahl der möglichen Oberbegriffe) zur Wurzel der Hierarchie versehen (Abbildung D-7). Nun sei C der gemeinsame Oberbegriff von A und B. (A, B und C können auch identisch sein). Dann sei der Hierarchieabstand *dist* die minimale Entfernung zwischen einem der beiden Unterbegriffe und dem Oberbegriff.

$$dist = \min(a(A), a(B)) - a(C) \qquad \text{(D-9.)}$$

Eine Ähnlichkeit läßt sich nun wie folgt quantifizieren:

$$\gamma' = \frac{g - dist}{g}. \qquad \text{(D-10.)}$$

In der Voreinstellung ist $g$ die maximale Entfernung irgendeines Begriffes der Hierarchie von der Wurzel, also die Höhe der Hierarchie. Dies bedeutet, daß der aktuelle Schritt in der Hierarchie, welcher nötig ist, um zum Oberbegriff zu kommen, ins Verhältnis gesetzt wird zur Größe der Hierarchie.

Als Alternative dazu würde sich ein Verhältnis des aktuellen Schrittes zur Größe der Resthierarchie anbieten, so daß für die Ähnlichkeitsbestimmung nicht mehr die gesamte Hierarchie, sondern nur noch der minimale Teil, welcher zur Beschreibung beider zu vergleichender Begriffe nötig ist, berücksichtigt wird.

Da die Aufgabenstellung an dieser Stelle keine eindeutige Vorgehensweise festlegte und auch in der Literatur keine allgemeingültige Beschreibung gefunden werden konnte, wurden die beiden oben erwähnten Verfahren implementiert und können nun von dem Anwender bzw. der Anwenderin durch Parameter ausgewählt werden (siehe Tabelle D-3 - Merkmalsparameter).



## D.4.3 Kurzbeschreibung der Merkmalsparameter

| Parameter | Typ | Default | Bedeutung |
|---|---|---|---|
| 1. | float | >1 | Gewichtung des Merkmals, bei Wert größer 1 wird Identität der Merkmale gefordert |
| **Merkmal ganz_1:** | | | |
| 2. | float | 1 | Ähnlichkeitsabstand *l* für Objekt-Objekt- Vergleich |
| **Merkmal ganz_2:** (unterstützt Intervalle) | | | |
| 2. | float | 1 | Ähnlichkeitsabstand *l* |
| 3. | float | 0 | Abstand, bei dessen Unterschreitung benachbarte Elemente zu Intervallen zusammengefaßt werden (0 = keine Intervallbildung) |
| 4. | int | 5 | Anzahl der Stützstellen in *l* für die lineare Approximation der Integrale |
| **Merkmal reell_1:** | | | |
| Bedeutung und Art der Merkmale siehe ganz_1 | | | |
| **Merkmal reell_2:** (unterstützt Intervalle) | | | |
| Bedeutung und Art der Merkmale siehe ganz_2 | | | |
| **Merkmal symbol:** | | | |
| keine weiteren Parameter | | | |
| **Merkmal begriff:** | | | |
| 2. | char* | notwendig | Dateiname, aus der die Hierarchie gelesen werden soll |
| 3. | 'Gesamt' / 'Teil' | 'Gesamt' | "Gesamt" - Berücksichtigung der Höhe der Gesamthierarchie<br>"Teil" - Berücksichtigung der Resthierarchiehöhe |
| 4. | float | 0 | kleinster Wert, den der Vergleich annehmen kann, muß zwischen 0 und 1 liegen |
| **Merkmal gmenge_1:** | | | |
| 2. | char* | notwendig | Dateiname, aus der die geordnete Menge gelesen werden soll |
| 3. | siehe 2. Parameter bei ganz_1 | | |
| **Merkmal gmenge_2:** (unterstützt Intervalle) | | | |
| 2. | char* | notwendig | Dateiname, aus der die geordnete Menge gelesen werden soll |
| x. ($2 < x < 6$) | siehe (x-1). Parameter bei ganz_2 | | |

*Tabelle D-3 - Merkmalsparameter*