Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik IV

Manual

AIMA

24. März 2015

Inhaltsverzeichnis

1 Installation

AIMA (*Artifical Intelligence - A Modern Approach*) ist in und für Python 3 geschrieben. Wir empfehlen als Entwicklungsumgebung *Spyder*, was zum Beispiel hier herunter geladen werden kann: https://pypi.python.org/pypi/spyder

2 GUI

Nach dem Start von *AIMA*, kann über *Select Application* eine Umgebung gewählt werden, auf die in den folgenden Unterkapiteln näher eingegangen wird. Über *Load Algorithm* lässt sich ein beliebiger zur Umgebung passender, Algorithmus laden. Werden mehrere Algorithmen innerhalb einer Umgebung geladen, so werden diese ebenfalls unter *Load Algorithm* angezeigt und können genutzt werden. Über *Quit* kann *AIMA* beendet werden.

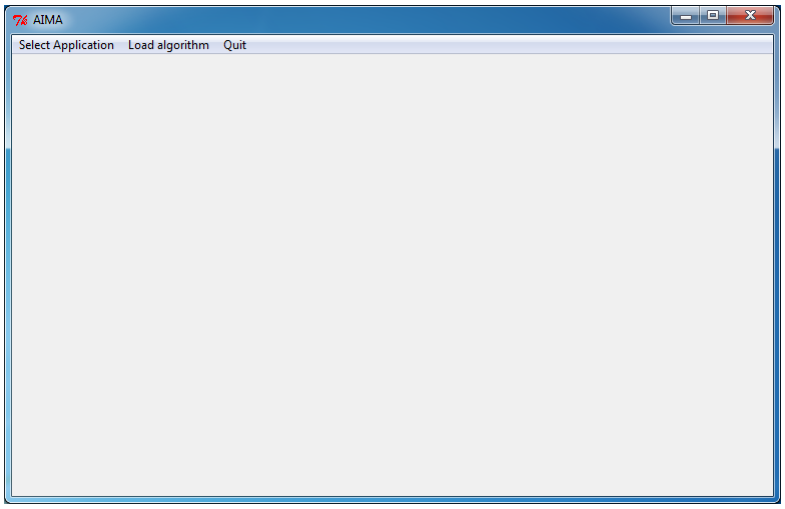


Abbildung 1: GUI nach Start des Skriptes Aima.py

2.1 Routenplanung

Um die Routenplanung zu benutzen, ist zunächst über *Select Application* -> *Routefinding* eine Karte auszuwählen (siehe Abbildung 2). Diese ist als Ascii-Datei in dem Format *.map gespeichert und muss folgende Struktur besitzen:

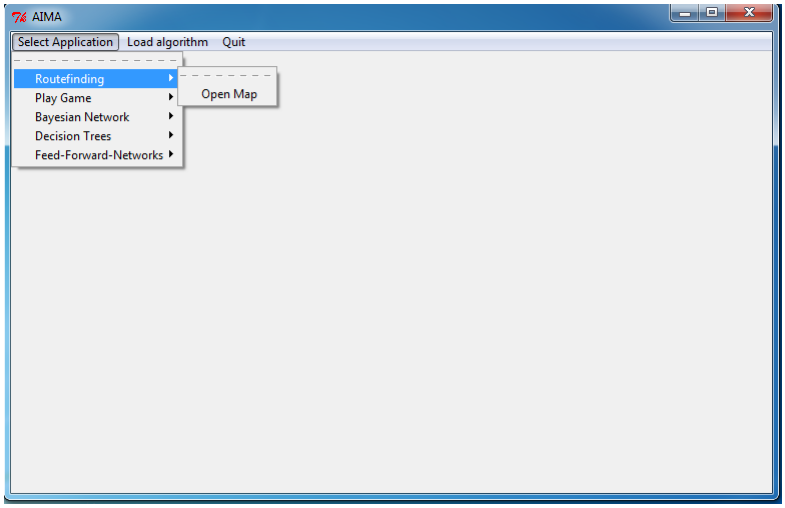


Abbildung 2: Routenplanung: Karte öffnen

Z1: Name der Karte
Z2: Anzahl Städte = N
N Zeilen: Stadtname x-Koordinate y-Koordinate (Jeweils durch Leerzeichen getrennt)
ZN+2:Anzahl Straßen = M
M Zeilen : StadtA StadtB (Jeweils durch Leerzeichen getrennt)

Daraus ergibt sich dann zum Beispiel eine Karte wie in Abbildung 3. Dort können nun über zwei Pulldown-Menüs der Start und das Ziel bestimmt werden.

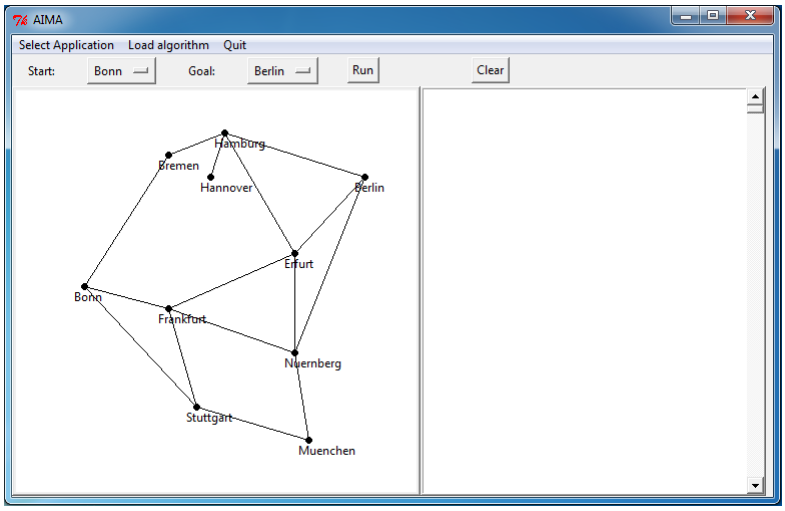


Abbildung 3: Deutschlandkarte

Über *Run* können die geladenen Algorithmen auf der Karte ausgeführt werden.

Im rechten Protokollfenster wird die Vorgehensweise des Algorithmus angezeigt, während diese (und auch später der fertige Weg) im linken Fenster mit blauen Kanten visualisiert wird (siehe Abbildung 4).

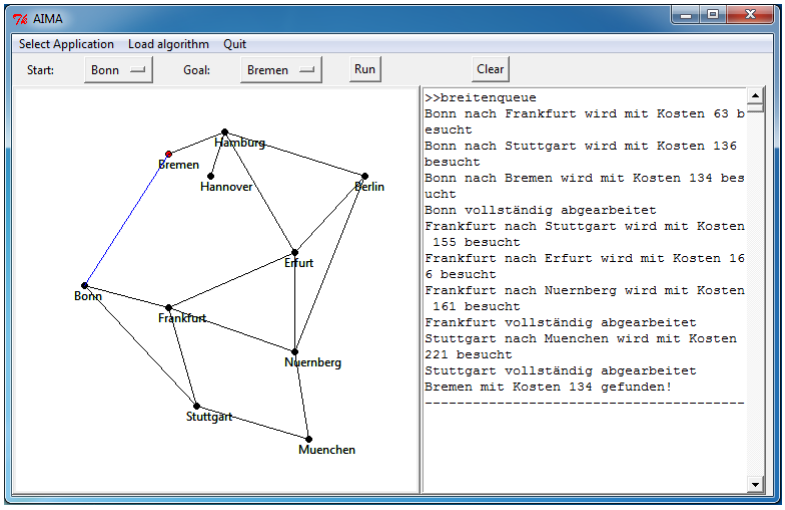


Abbildung 4: Routenplanung: Anzeige des Suchalgorithmus auf der Karte

Clear löscht in dieser Umgebung die Ausgaben im Protokollfenster und setzt den Graphen wieder auf den Ausgangspunkt zurück. Alle bisher gezeichneten Wege werden gelöscht.

2.2 Spiele

In *AIMA* besteht die Möglichkeit ein Spiel zu spielen. Die Auswahl erfolgt über *Play Game*. Die Regeln erscheinen jeweils im rechten Protokollfenster (siehe Abbildung 5).

Nachdem ein Algorithmus geladen wurde, kann das Spiel über die Buttons (im Fall von Matches über *Take 0*, *Take 1* und *Take 2*) gespielt werden. Der Zug des Computers erfolgt dabei direkt im Anschluss an den eigenen Zug. Im Protokollfenster wird dabei der Fortgang des Spiels dokumentiert (siehe Abbildung 6).

Clear setzt das Spiel wieder auf den Anfangszustand zurück, im Fall von Matches liegen wieder alle Streichhölzer in der Mitte.

2.3 Bayes-Netze

Zur Nutzung der Bayes-Netze ist zunächst über *Select Application -> Bayesian Network* ein Bayes-Netz auszuwählen. Dieses ist als Ascii-Datei in dem Format *.bayes

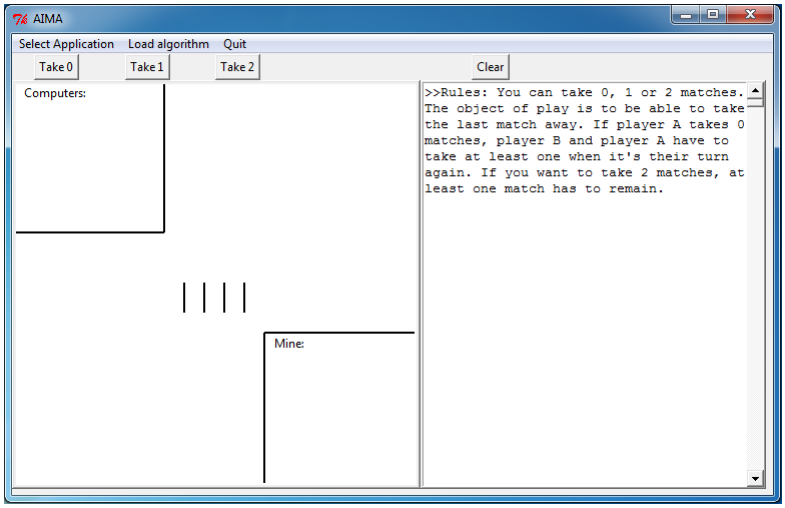


Abbildung 5: Streichholzspiel

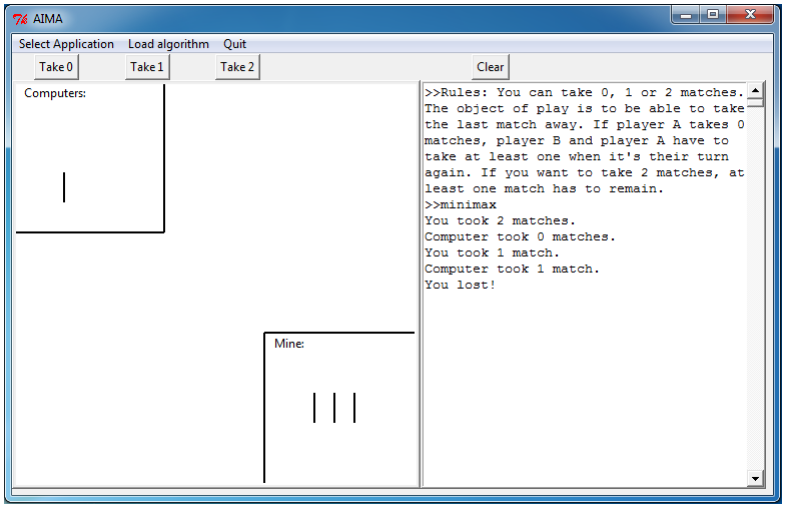


Abbildung 6: Fertig gespieltes Streichholzspiel

gespeichert und muss folgende Struktur aufweisen:

Z1: Name
Z2: Anzahl Knoten = N
N Zeilen: Knotenname; Ebene; Anzahl Elternknoten = E; E-mal: Name des Elternknoten; für jede mögliche Belegung der Elternknoten: 'Belegung Elternknoten' Wahrscheinlichkeit dass dieser Knoten positiv ist, usw..

Es wird dann wie in Abbildung 7 dargestellt gezeichnet.

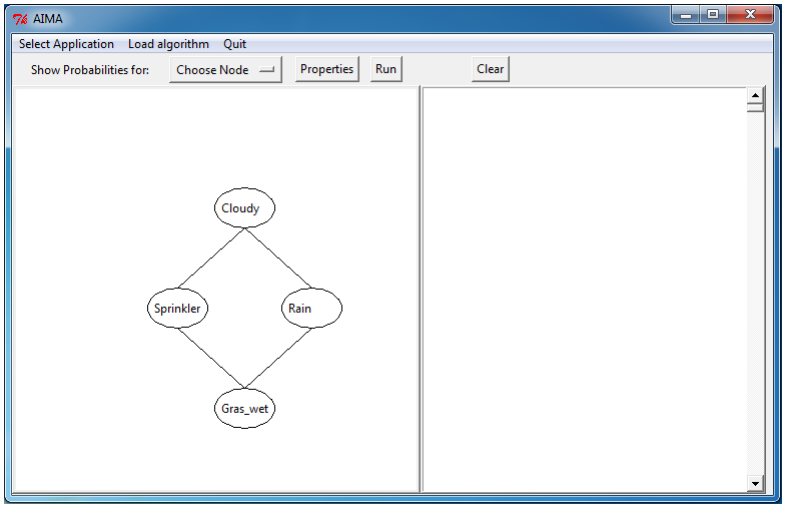


Abbildung 7: Darstellung des eingelesenen Bayes-Netzes

Die Wahrscheinlichkeitstabellen der einzelnen Knoten können über ein Pulldown Menü ausgewählt und dann in einem neues Fenster angezeigt werden (siehe Abbildung 8).

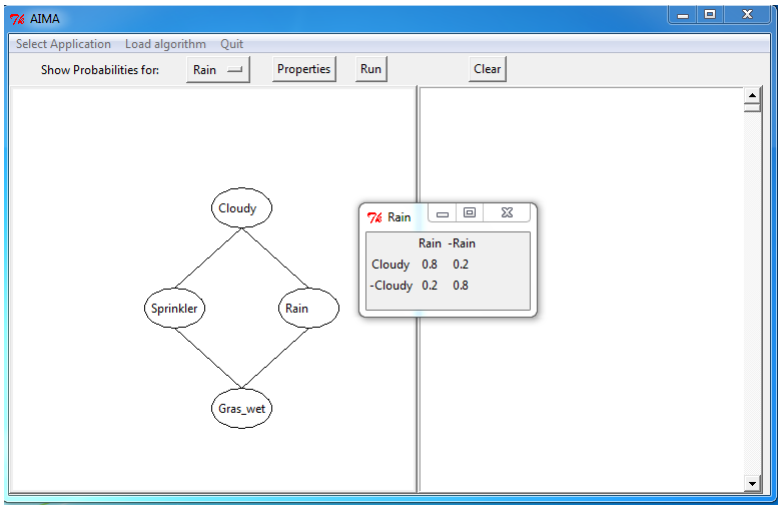


Abbildung 8: Wahrscheinlichkeitstabelle des Knotens *Rain*

Der gegebene Prior-Sampling-Algorithmus kann ohne weitere Eingabe von Eigenschaften gestartet werden und führt beispielsweise zu einer Ausgabe, wie in Abbildung 9 zu sehen.

Für die Benutzung des Rejection-Sampling-Algorithmus muss zunächst über *Properties* gewählt werden, wie viele mit Prior-Sample gesamplete Belegungen des Netzes in die Entscheidung einfließen und welcher Knoten gesampled werden soll. Abhängig

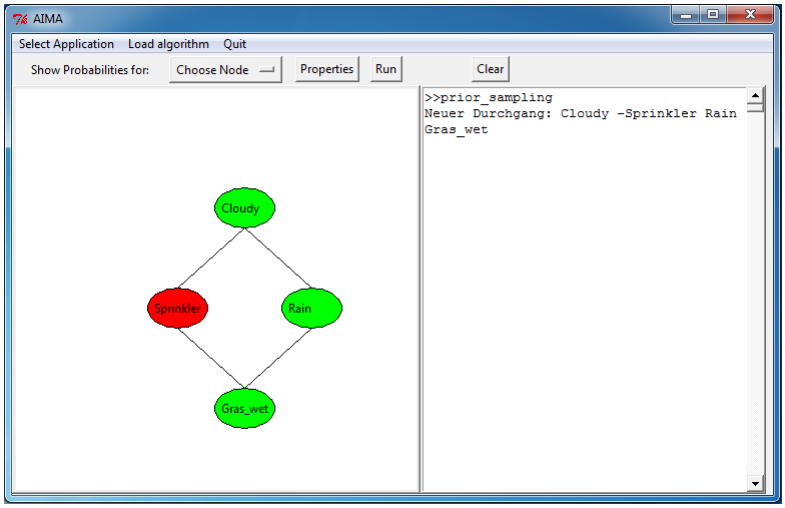


Abbildung 9: Ausgabe des Priorsamplings auf dem gegebenen Netz

von diesem gewählten Knoten kann dann die Belegung der anderen Knoten bestimmt werden (siehe Abbildung 10).

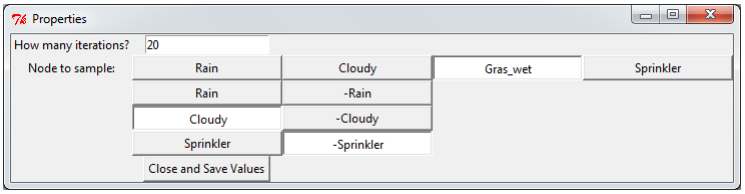


Abbildung 10: Eigenschaften für *Eigenschaften für Rejection Sampling*

In der Ausgabe wird der gesampelte Knoten blau dargestellt, während die anderen entsprechend ihrer Belegung rot oder grün gefärbt werden. Im Protokollfenster lässt sich die berechnete Wahrscheinlichkeit ablesen (siehe Abbildung 11).

2.4 Entscheidungsbäume

Um Entscheidungsbäume erstellen zu können, muss über *Select Application -> Decisiontrees* eine Menge an Trainingsbeispielen geladen werden. Diese werden als Ascii-Datei mit dem Foramt *.feat gespeichert. Es weist folgende Struktur auf:

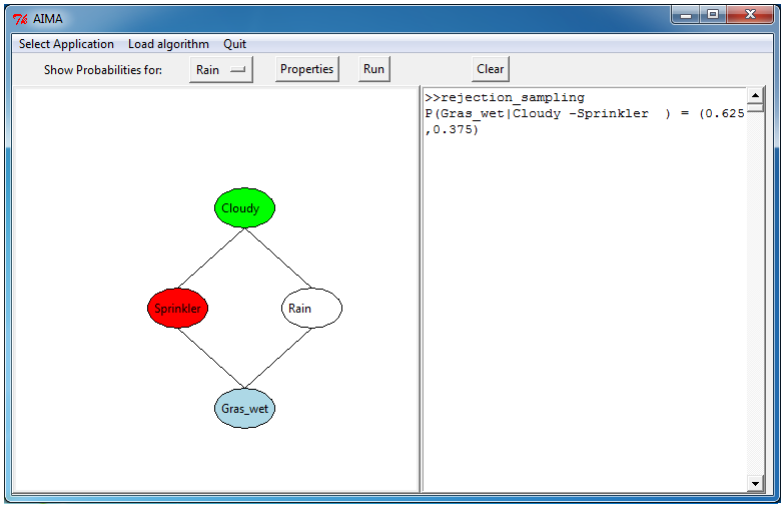
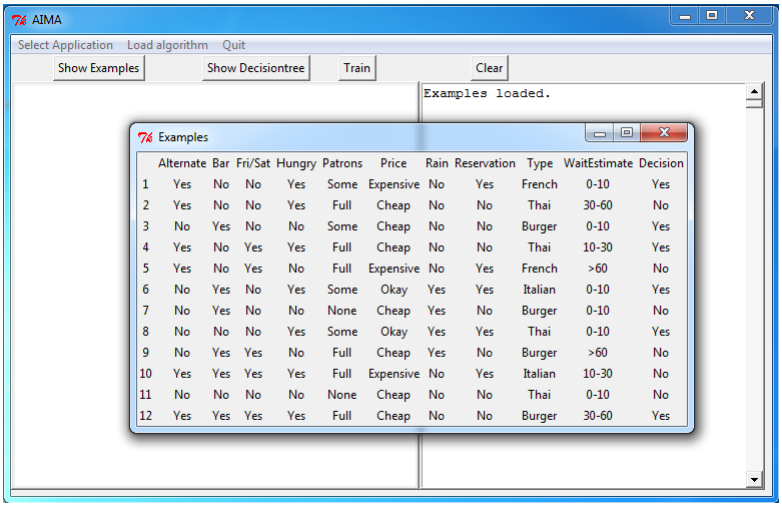


Abbildung 11: Ausgabe von Rejection Sampling

Z1: Name der Beispiele
Z2: Anzahl der Attribute = N
Z3: Attribut1 Attribut2 ... AttributN (Duch Leerzeichen getrennt)
Z4: Anzahl der Beispiele = M
M Zeilen: Belegung der Attribute in der in Zeile 3 festgelegten Reihenfolge. Letztes Element ist die Entscheidung (alle Elemente sind durch Leerzeichen getrennt).

Sind die Beispiele geladen, können sie über *Show Examples* angesehen werden (siehe Abbildung 12).



	Alternate	Bar	Fri/Sat	Hungry	Patrons	Price	Rain	Reservation	Type	WaitEstimate	Decision
1	Yes	No	No	Yes	Some	Expensive	No	Yes	French	0-10	Yes
2	Yes	No	No	Yes	Full	Cheap	No	No	Thai	30-60	No
3	No	Yes	No	No	Some	Cheap	No	No	Burger	0-10	Yes
4	Yes	No	Yes	Yes	Full	Cheap	No	No	Thai	10-30	Yes
5	Yes	No	Yes	No	Full	Expensive	No	Yes	French	>60	No
6	No	Yes	No	Yes	Some	Okay	Yes	Yes	Italian	0-10	Yes
7	No	Yes	No	No	None	Cheap	Yes	No	Burger	0-10	No
8	No	No	No	Yes	Some	Okay	Yes	Yes	Thai	0-10	Yes
9	No	Yes	Yes	No	Full	Cheap	Yes	No	Burger	>60	No
10	Yes	Yes	Yes	Yes	Full	Expensive	No	Yes	Italian	10-30	No
11	No	No	No	No	None	Cheap	No	No	Thai	0-10	No
12	Yes	Yes	Yes	Yes	Full	Cheap	No	No	Burger	30-60	Yes

Abbildung 12: Tabelle der geladenen Trainingsbeispiele

Nachdem ein Algorithmus geladen wurde, kann dieser mittels *Train* auf die Beispiele angewendet werden. Ist der entstandene Entscheidungsbaum klein genug, so wird er direkt im linken Fenster angezeigt. Ist er zu groß, kann dieser über *Show Decisiontree* in einem neuen Fenster geöffnet werden (siehe Abbildung 13).

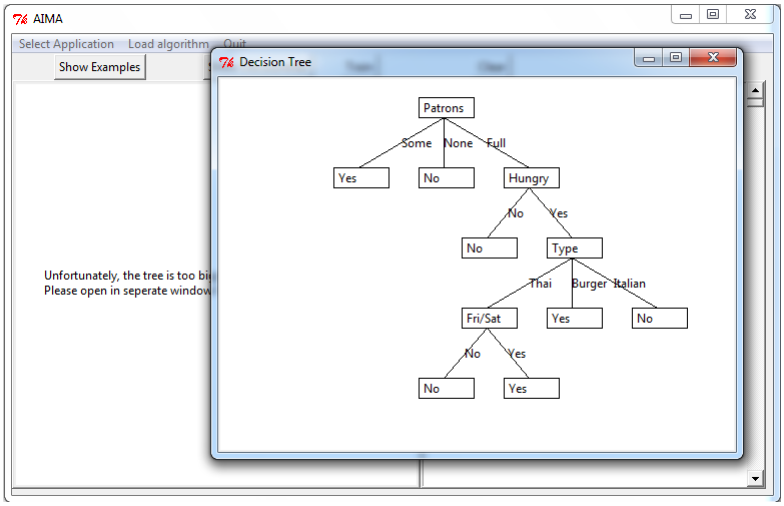


Abbildung 13: Fertiger Entscheidungsbaum basierend auf Trainingsbeispielen

2.5 Neuronale Feed-Forward-Netze

Zur Nutzung der Feed-Forward-Netze muss über *Select Application -> Feed-Forward-Networks*, wie auch schon bei den Entscheidungsbäumen, eine Features-Datei geladen werden welche die Trainingsbeispiele beinhaltet.
Abhängig von dieser *.feat Datei wird eine Netzwerktopologie geladen, welche anschließend im linken Fenster zu sehen ist (siehe Abbildung 14).
Die Netztopologie wird ebenfalls als Ascii-Datei gespeichert. Diese besitzt das Format *.ffn, welches folgende Struktur aufweist:

Z1: Name des Netzes
Z2: Anzahl an Knoten = N
N Zeilen: Knotenname Knotentyp (durch Blanks getrennt)
ZN+2: Anzahl Kanten = M
M Zeilen: KnotenA KnotenB (durch Blanks getrennt)

Die Kantengewichte werden initial auf einen zufälligen Wert zwischen 0 und 1 gesetzt. Die Gewichte werden mittels eines Hinton-Diagramms verdeutlicht(siehe Abbildung 15).

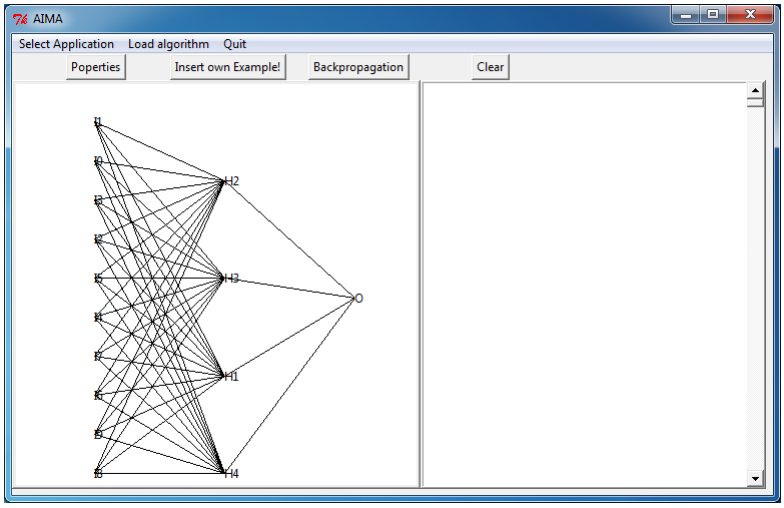


Abbildung 14: Netztopologie

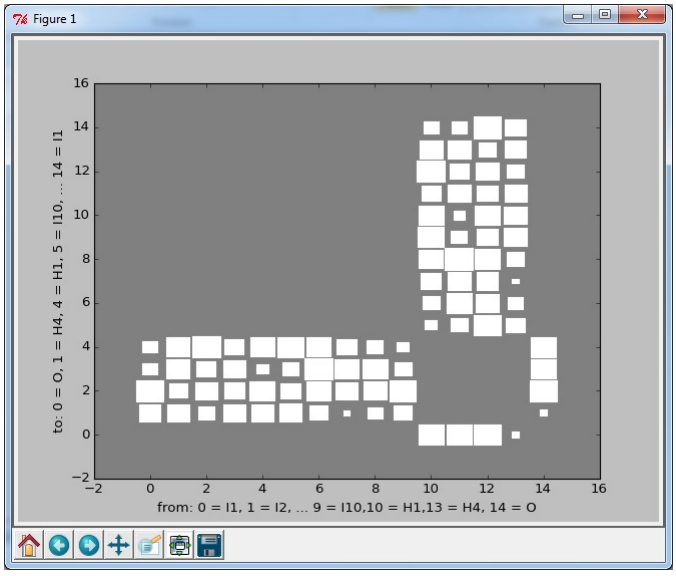


Abbildung 15: Hinton Diagramm

Jedes Kästchen steht dabei für ein Kantengewicht. Je größer es ist, desto näher ist das Kantengewicht an 1. Ist ein Kästchen weiß, so ist das Kantengewicht positiv, ist es schwarz, so ist das Gewicht negativ.
Nachdem der Backpropagation Algorithmus geladen wurde, kann über *Properties* eingestellt werden, wie hoch der Schwellwert sein soll, bei welchem der Algorithmus abbricht; nach jedem wievielten Durchgang das Hinton-Diagramm erneuert wird und wie hoch die Lernrate sein soll (siehe Abbildung 16).

Danach kann über *Backpropagation* der Algorithmus gestartet werden.

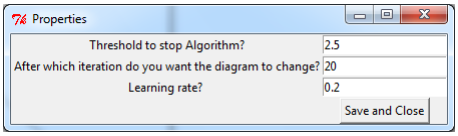


Abbildung 16: *Properties* für die Backpropagation

Ein eigenes Beispiel kann im Anschluss über *Insert own Example* eingegeben und entschieden werden (siehe Abbildung 17).

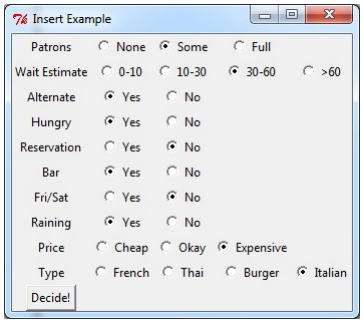


Abbildung 17: Eigenes Beispiel eingeben

Clear löscht zum einen erneut das Protokollfenster und setzt zum anderen die Kantengewichte wieder auf zufällige Werte.

3 Datenstrukturen und ihre Anwendung

Jedes der fünf Themengebiete wird in Form von Graphen mit Knoten und Kanten gespeichert. Wie diese generell und im speziellen verwendet werden können, wird in den einzelnen Abschnitten erklärt.

3.1 Graph

3.1.1 Generelle Funktionen

graph.get_node(Nodename)	Gibt Referenz auf den Knoten, der den Namen *Nodename* besitzt.
graph.get_nodes()	Gibt Referenz für alle im Graphen vorhandenen Knoten zurück.

3.1.2 Spezielle Funktionen

Routenplanung

graph.luftlinie(NodenameA, nodenameB)	Berechnet die direkte Distanz zwischen KnotenA und KnotenB.
graph.ausgabe(currendnodename, nextnodename, fcost, gcost, action)	wobei action eins der folgenden sein kann: *bereitsbesucht*, *gefunden*, *expandieren*, *abgearbeitet*. Ist für die textuelle und visuelle Darstellung des Suchalgorithmus in der GUI.

Spiele

graph = GameGraph(Name)	Initialisiert einen Spielbaum
graph.add_node(Name, Typ, Elternknoten, Bewertung)	Fügt dem Graphen einen Knoten hinzu.
grapg.add_edge(NameA, NameB, Gewicht)	Fügt zwischen Knoten A und Knoten B eine Kante mit dem Gewicht hinzu.

Entscheidungsbäume

graph = DecisionTree()	Initialisiert einen Entscheidungsbaum
graph.add_node(name, isroot?)	Fügt dem Graphen einen Knoten hinzu.
graph.add_edge(NameA, NameB, Title)	Fügt eine Kante zwischen KnotenA und KnotenB und der Bezeichnung *Tilte* hinzu.

3.2 Node

3.2.1 Generelle Funktionen

node.name()	Gibt den Namen eines Knoten zurück
node.get_edges()	Gibt Referenzen für jede Kante, die vom Knoten ausgeht.

3.2.2 Spezielle Funktionen

Routenplanung

node.set_parent(Elternknoten)	Setzt den Elternknoten von node

Spiele

node.set_value(value)	Setzt den Wert des Knotens auf *value*
node.get_value()	Gibt den Wert des Knotens zurück

Bayes Netze

node.ebene()	Gibt die Ebene des Knotens zurück
node.get_probs()	Gibt die Wahrscheinlichkeitstabelle des Knotens zurück.

3.3 Edge

3.3.1 Generelle Funktionen

edge.end()	Gibt eine Referenz auf den Knoten am Ende der Kante
edge.weight	Gibt das Gewicht der Kante zurück

3.3.2 Spezielle Funktionen

Entscheidungsbäume

edge.set_weight(weight)	Setzt das Gewicht der Kante neu.