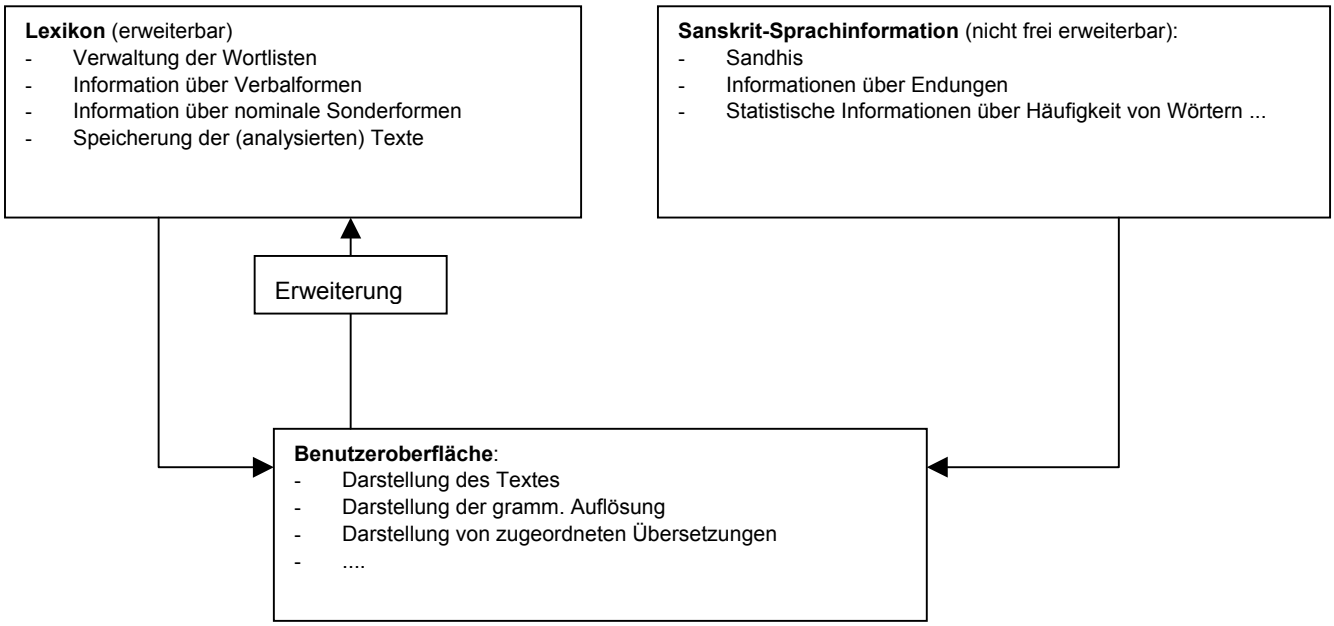
Sprachanalyse und Codierung der Sanskrit-Grammatik

In diesem Teil des Programms werden lexikalischer Aufbau und Syntax originalsprachlicher Sanskrit-Texte automatisch analysiert. Als Eingabe wird allein ein orthografisch richtiger digitalisierter Text - entweder aus dem OCR-Modul oder einer externen Textdatei - ohne manuell hinzugefügte Sandhi-Trennung benötigt. Das Programm generiert aus diesem Text eine Reihe möglicher Auflösungen, die vom Benutzer auf ihre Richtigkeit überprüft und dann gespeichert werden können.

Dieses Konzept ermöglicht es, weit genauer und treffsicherer als mit der bisher üblichen stringbasierten Suche das Auftreten von Wörtern in einer Vielzahl verschiedener Texte zu untersuchen. Suchergebnisse können mit Hilfe einfacher statistischer Funktionen auch grafisch repräsentiert werden.

Die folgende Grafik gibt einen ersten Überblick über den Funktionsfluss während der Sprachanalyse.



Vorüberlegungen

Der computergestützten Analyse von Sanskrit-Texten stellen sich eine Reihe von Hindernissen entgegen, die zum grossen Teil bei der Analyse von Texten in europäischen Sprachen nicht auftreten:

- Die Phonetik des Sanskrit verbindet Einzelwörter zu Lautfolgen. Dabei kann es sich einerseits um selbstständige Formen handeln, die mittels festgelegter Verschmelzungsregeln (Sandhi) zu grösseren Einheiten verbunden werden ("Satzsandhi"). Andererseits nimmt im Lauf der Sprachentwicklung des Sanskrit die Tendenz zu, Komposita zu erstellen, d.h. Zusammensetzungen nicht-deklinierter Formen einzelner Wörter (Stammformen) zu einem grammatikalisch nicht mehr trennbarem Ganzen, das seine Bedeutung aus der Hierarchie seiner Bestandteile erhält ("Wortsandhi"). Ein Analyse-Programm muss fähig sein, die Verschmelzung selbstständiger Wörter, Komposita und Mischformen beider Fälle

zu erkennen und angemessen aufzulösen. Besonders muss es in der Lage sein, (mehrdeutige) Sandhis zu lokalisieren und in ihre Ausgangsformen zu zerlegen.
- Ein weiteres Problem ist der Formenreichtum der Sanskrit-Grammatik, der die Auflösungszeit, die Zahl möglicher Lösungen und den Aufwand bei der morphologischen Analyse beträchtlich erhöht.
- Die lexikalische Reichhaltigkeit des Sanskrit ist schliesslich - in Verbindung mit den beiden vorher genannten Punkten - besonders dafür verantwortlich, die Zahl möglicher Auflösungen einer Lautfolge zu vergrössern. Sie erschwert weiterhin die Bemühungen, aus einer Menge von gültigen Auflösungen die wahrscheinlichste zu wählen.

Eine Auflösungsmethode muss demnach folgende Kriterien erfüllen:

- Die Auflösungszeit pro Lautfolge sollte möglichst niedrig sein.
- Das Programm sollte versuchen, aus einer Reihe von Auflösungen die wahrscheinlichste auszuwählen. So können schon bei Wörtern von ca. 10 Zeichen mehrere hundert bis tausend Auflösungsmöglichkeiten auftreten, wenn man sämtliche grammatikalischen Mehrdeutigkeiten in Betracht zieht.
- Das Programm sollte schliesslich nicht nur die wahrscheinlichsten lexikalischen Auflösungen auswählen können, sondern auch einen syntaktisch sinnvollen Text produzieren.

Die Auflösungsstrategie für Sanskrit-Wörter gliedert sich in fünf Stufen:

- **Identifizierung möglicher Sandhi-Bruchstellen** innerhalb des untersuchten Strings, d.h. derjenigen Indizes, die möglicherweise einen Sandhi oder die Bildefuge eines Kompositums darstellen.
- **Rekursives Nachschlagen** der Teilstrings im Lexikon
- **Aktivierung der lexikalisch wahrscheinlichsten Auflösung** über ein Hidden-Markov-Modell
- **Aktivierung der syntaktisch wahrscheinlichsten Auflösung** auf Basis der lexikalischen Information, die im vorhergehenden Schritt gewonnen wurde.
- **Speichern** der Auflösung

Identifizierung der Sandhi-Bruchstellen

Der erste Schritt der Auflösungsstrategie ist die Markierung von Lautkombinationen innerhalb des untersuchten Strings, die aus einem Sandhi entstanden sein könnten oder die dem Stammvokal einer Nominalklasse entsprechen. Dazu wird der String von links nach rechts gescannt. In einer Liste werden dabei die Kandidaten mit ihrem Index und mögliche Ersatzlaute für die spätere Verarbeitung gespeichert.
Die Sandhiregeln, die hierbei zum Einsatz kommen, sind fest in den Quellcode integriert und können nicht verändert werden. Jede Regel besteht aus sechs Einträgen, die u.a. die Art des Sandhis (z.B. „Satzsandhi“) und mögliche Ausgangsformen, die zu der Lautverschmelzung geführt haben, speichern.
Einen Sonderfall bilden die Sandhis, die einen konsonantischen Wortanlaut verändern. Sie werden in einer der geschilderten Sandhierkennung vorgeschalteten Stufe untersucht und auf ihre konsonantischen Ausgangsformen reduziert.

Die Identifizierung von Bruchstellen ist exemplarisch in der folgenden Grafik dargestellt. Hellgraue Buchstaben im Ausgangswort *madhurāmlavṛkṣam* bezeichnen dabei die möglichen Bruchstellen, an denen die Auflösungsstrategie ansetzen kann.

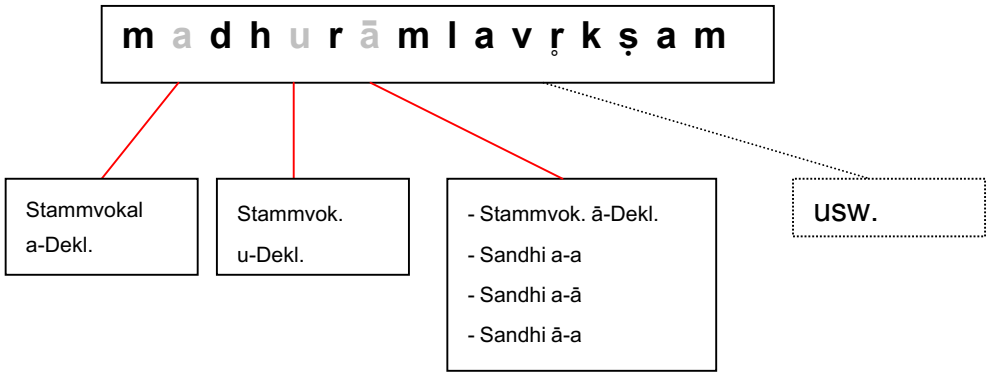


Abbildung 1: Identifizierung möglicher Sandhis

Rekursives Nachschlagen

Beim Nachschlagen wird der String *s* von links nach rechts auf mögliche Bruchstellen untersucht, die im ersten Schritt gefunden wurden. Sobald der Index einer Bruchstelle erreicht ist, wird der stringinterne Laut durch die Sandhi-Auflösungen der Bruchstelle ersetzt. Lässt sich der linke Teil von *s* korrekt grammatikalisch und lexikalisch auflösen, wird der String gekürzt und die Prozedur mit dem übrig bleibenden rechten Teil wiederholt. Sobald das Programm am letzten Laut des Ausgangswortes angekommen ist, wird die gefundene Wortfolge als gültige Auflösung gespeichert.

Der Prozess ist für die Sandhi-Bruchstelle am ā in der folgenden Grafik dargestellt. Das Programm löst das lange a in der ersten Variante in die Kombination *a* - *a* auf. Das erste *a* dient als Stammvokal für das Wort *madhura*, das zweite *a* bildet den Anlaut des Restwortes (amla...). Weil das Wort *madhura* im Lexikon gefunden wird, wird in der ersten Rekursion der mit amla... beginnende Reststring untersucht. Die

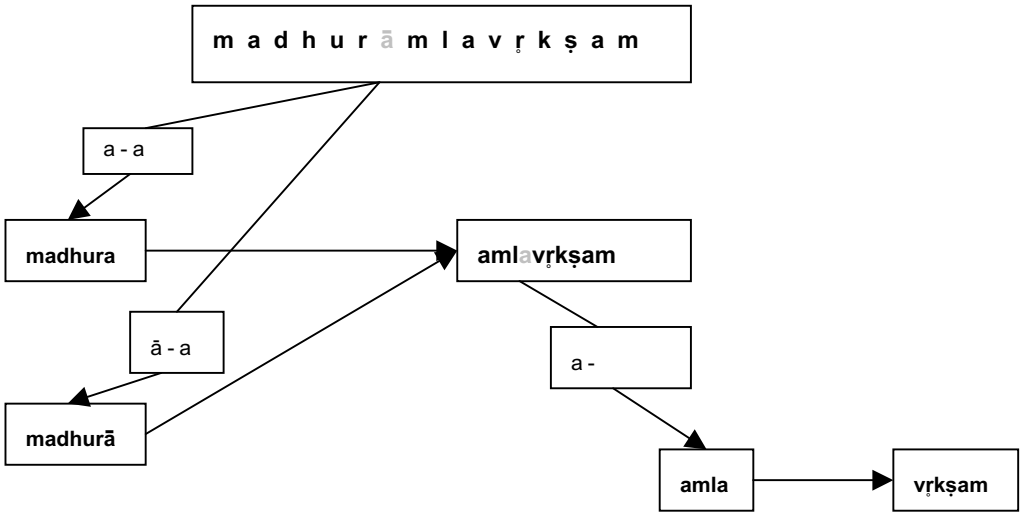


Abbildung 2: Ausschnitt aus der rekursiven Sprachanalyse

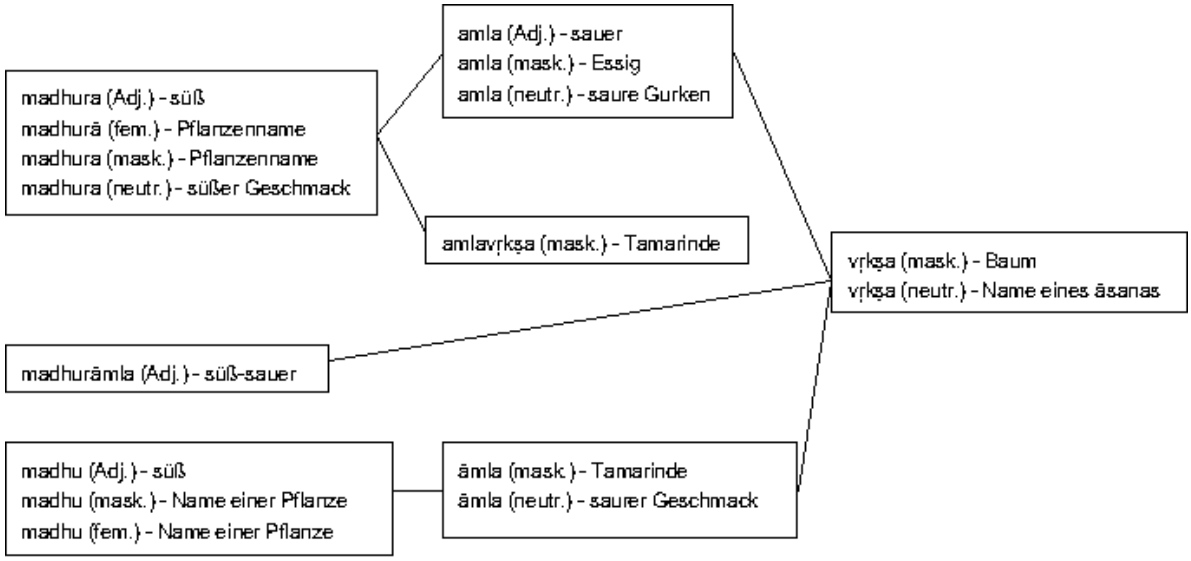
Rekursionen werden solange fortgesetzt, bis das Ende des Strings erreicht ist (erfolgreiche Auflösung) oder der rechts stehende Reststring keine Entsprechung im Wörterbuch findet (Scheitern).

Aktivierung der wahrscheinlichsten lexikalischen Auflösung

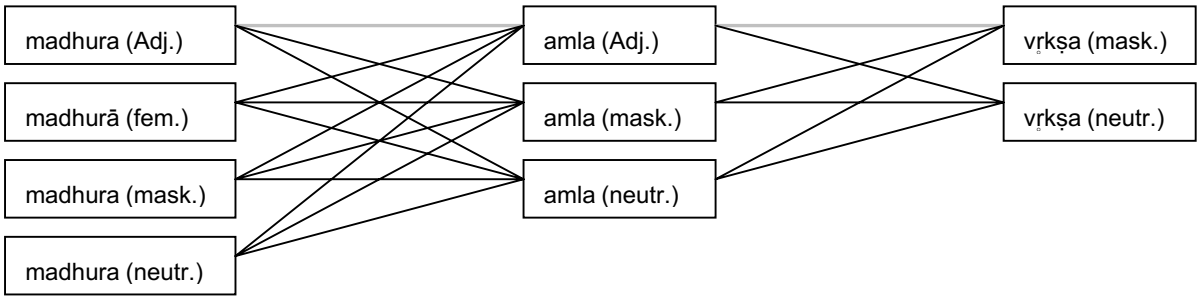
Bei einem Großteil der Sanskrit-Wörter ergibt sich aus der lexikalischen Analyse, die im vorigen Schritt vorgestellt wurde, ein komplexes Geflecht verschiedener Auflösungsmöglichkeiten, die am besten in einer Graphen-Struktur dargestellt werden können. Ein **Pfad** ist dabei ein Weg durch diesen Graphen, der eine vollständige und gültige Auflösung des untersuchten Wortes darstellt. Theoretisch könnte die Sprachanalyse an diesem Punkt abbrechen und die Auswahl des korrekten Pfades, der dann gespeichert wird, dem Benutzer überlassen. Da sich aber schon bei kurzen Wörtern mehrere hundert Pfade ergeben können, ist die Variante praktisch kaum vertretbar. Es muss also ein möglichst guter Pfad gefunden werden, der dem Benutzer als Vorauswahl zur Speicherung präsentiert wird.

Die aktuelle Programmversion führt diese Vorauswahl durch die Anwendung der Grundidee eines Markov-Modells (MM) durch (vgl. z.B. Paulus (1998), 345ff.). Im hier betrachteten Fall ist v.a. die Frage interessant, welcher Pfad (= Zustandsfolge) auf Basis statistischer Informationen, die aus bisher analysierten Texten gewonnen wurden, die grösste Wahrscheinlichkeit besitzt und daher aktiviert werden soll. Die statistischen Informationen, die bei diesem Modell zur Anwendung kommen, betreffen die Kookkurrenz von Wörtern in den analysierten Texten. Sobald ein Text gespeichert wird, werden in einer programminternen Matrix, die alle bisher aufgetretenen Wörter verzeichnet, die Übergangswahrscheinlichkeiten zwischen den Wörtern im jetzt gespeicherten Textabschnitt erhöht. Wird also ein Satz *aabac* aus dem Grundwortschatz {*a,b,c*} gespeichert, werden die Wahrscheinlichkeiten *a->a*, *a->b*, *b->a* und *a->c* erhöht. Es seien jetzt die Folgen p_1 = *aac* und p_2 = *cba* zwei konkurrierende Pfade, die bei der Auflösung eines anderen Satzes entstanden sind. Das Programm fragt - vereinfacht gesagt - zur Bewertung dieser Pfade für p_1 unter anderem die Übergangswahrscheinlichkeiten *a->a* und *a->c*, für p_2 die Übergangswahrscheinlichkeiten *c->b* und *b->a* ab. Aufgrund dieser Information fällt die Entscheidung z.B. zugunsten von p_1, der aktiviert wird. In den folgenden Grafiken wird der Auswahlprozess angedeutet.

Die erste Darstellung zeigt einen Ausschnitt der möglichen Auflösungspfade des Wortes *madhurāmlavṛkṣam*, wobei unterschiedliche Kasusformen der auftretenden Wörter nicht berücksichtigt sind.



Die zweite Grafik zum Auflösungsprozess zeigt einen Ausschnitt aus dem ersten Graphen. Aufgrund der Informationen aus den bisher analysierten Texten aktiviert das Programm die am weitesten oben stehende Wortfolge *madhura* (Adj.) – *amla* (Adj.) - *vṛkṣa* (mask.), die mit einem massiven hellgrauen Pfad gekennzeichnet ist.



Ein Problem bei der hier vorgestellten lexikalischen Analyse ist ihre Beschränkung auf ein Kompositum. In seinem Grundgedanken ist der verwendete Viterbi-Algorithmus auf Sequenzen fester Länge ausgelegt, an deren Einzelpositionen aber unterschiedlich viele Vorschläge auftreten können. Betrachtet man einen Sanskrit-Satz, ergeben sich aber andere Anforderungen, die am folgenden Beispiel illustriert werden sollen.
Gegeben sei ein Satz, der aus zwei Lautfolgen a und b besteht. Die Lautfolge a sei der Einfachheit halber nur auf eine Art zu analysieren. Die Lautfolge b aber besteht aus einem Kompositum, das wiederum in zwei oder drei Wörter aufgelöst werden kann. Der momentan implementierte Algorithmus untersucht nun b in einem separaten Schritt und aktiviert unter Berücksichtigung der unterschiedlichen Anzahl von Auflösungen lokal die wahrscheinlichste Variante in b. Angemessener wäre allerdings, dass auch die Übergangswahrscheinlichkeit von a auf das jeweils erste

Wort der Auflösungen von b in den Optimierungsprozess einbezogen wird, d.h. die Pfadwahrscheinlichkeit über den ganzen Satz und nicht nur innerhalb eines mehrdeutig aufgelösten Kompositums maximiert wird.

Aktivierung der syntaktisch/morphologisch wahrscheinlichsten Auflösung

Nachdem das Programm im vorausgehenden Schritt – unter den erwähnten Einschränkungen - die lexalisch wahrscheinlichste Variante zur Auflösung eines Satzes gefunden hat, muss nun eine syntaktisch vernünftige Sequenz generiert werden. Dieser Teilalgorithmus greift auf drei Informationsquellen aus bisher in der Programmdatenbank gespeichertem Text zurück:

1. Für Nominalformen werden Kasus, Numerus und Genus gespeichert.
2. Bei finiten Verbalformen werden Person und Numerus aufgezeichnet.
3. Die Informationen aus 1. und 2. werden für jedes Lemma nochmals gesondert im Lexikon verzeichnet (lexikalisch differenzierte morphologische Wahrscheinlichkeit).

Ein Beispiel: In einem Text findet sich die Form „*brahma*“, die als Nom. Sg., Akk. Sg. und Vok. Sg. analysiert werden kann. Aus vorher analysierten Texten ist bekannt, dass „*brahman* (neutr.)“ zwar im Nominativ und Akkusativ, aber fast nie im Vokativ auftritt. Diese letzte Variante kann daher bei der Analyse zunächst übergangen werden.

Zur eigentlichen morphologischen Analyse greift das Programm zunächst auf einen Trie zurück, in dem die Informationstypen 1 und 2 für bisher analysierte Sätze verzeichnet sind. Lässt sich einer der Pfade des zu analysierenden Satzes im Trie mit einer festgesetzten Mindesthäufigkeit finden, wird dieser Pfad ohne Rücksicht auf Übergangswahrscheinlichkeiten aktiviert.
Findet sich, und das ist beim aktuellen Umfang der Datenbank der übliche Fall, keine exakte Entsprechung im Trie, greift das Programm auf die Matrix der Übergangswahrscheinlichkeiten (Typ 1 und 2) und auf die lexikalisch gebundenen morphologischen Wahrscheinlichkeiten (Typ 3) zurück. Das MM generiert die Wahrscheinlichkeit eines Übergangs aus dem Produkt der beiden Wahrscheinlichkeiten und errechnet auf Basis dieser Information die wahrscheinlichste Zustandsfolge.

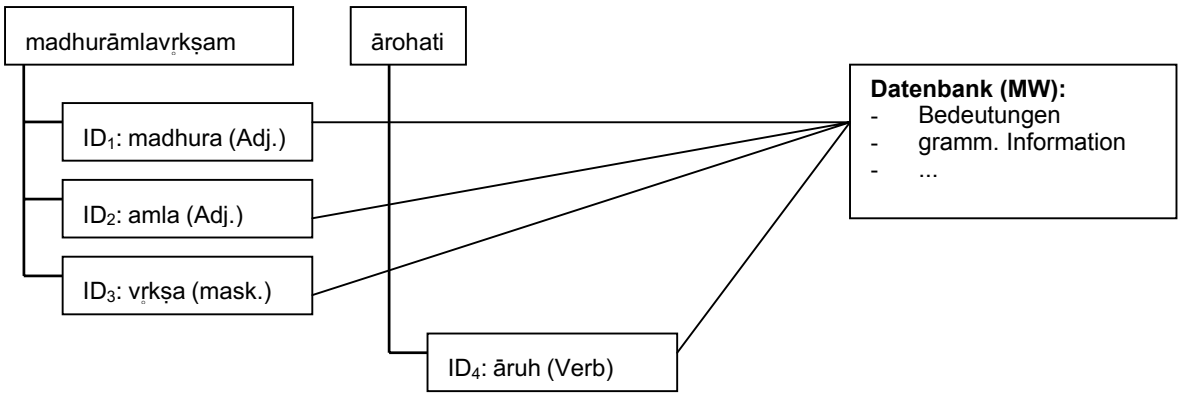
Während für die Typen 1 und 2 genügend Informationen aus den bisher analysierten Texten zu extrahieren sind, leiden die lexikalisch spezifizierten morphologischen Wahrscheinlichkeiten wieder unter dem geringen Umfang der Textdatenbank. Am oben aufgeführten Beispiel des Wortes „*brahma*“: Angenommen, in der Nähe des Wortes finden sich weitere Lemmata, die als Vok. Sg. Neutr. analysiert werden können, die Wahrscheinlichkeit, dass „*brahma*“ als Vok. Sg. Neutr. aufgelöst werden kann, ist nach Information der bisherigen Texte aber 0. Das Programm kann diese Informationen auf zwei Arten verarbeiten:

- Als Null-Wahrscheinlichkeit: Die Angaben aus den bisher analysierten Texte werden als korrekt angesetzt. „*brahma*“ kann deshalb kein Vokativ sein, eine andere – möglicherweise syntaktisch sinnlose – Variante wird aktiviert.
- Als geringe Wahrscheinlichkeit: Das Programm geht davon aus, dass eine Null-Wahrscheinlichkeit immer durch eine sehr kleine Wahrscheinlichkeit ersetzt werden kann, und aktiviert damit im gegebenen Fall die Auflösung als Vokativ. Als Resultat werden Pflanzen, Felsen und Bäume wie eine Person angesprochen – was sich in indischen Texten ja nicht per se ausschliessen lässt!

Die Reaktionen des Programms skizzieren zwei Wege zur Verbesserung der morphologischen und auch der lexikalischen Analyse. Im einfacheren Fall wird die Analyse mit wachsendem Umfang der Textdatenbank automatisch verbessert. Die zweite Möglichkeit bildet eine unüberwachte Analyse von noch nicht getaggtem Material, wie sie z.B. in Schmid (1995), 6 vorgeschlagen wird.

Speichern der endgültigen Auflösung

Nachdem der Benutzer die richtigen Auflösungen aller Wörter in einem analysierten Textausschnitt aktiviert hat, werden diese Auflösungen in der Datenbank gespeichert. Entscheidet man sich z.B. beim oben gelieferten Beispiel für die Auflösung in drei Einzelwörter, wird jedes dieser Wörter durch eine Identifikationsnummer (ID) ersetzt, die auf weitergehende lexikalische und morphologische Informationen in der Datenbank verweist.
Diese Speicherung ist der entscheidende Abstraktionsschritt im Programmverlauf. Mehrdeutige Sanskrit-Lautfolgen sind ab jetzt durch eindeutig identifizierbare Sequenzen von IDs ersetzt, die effektiv durchsucht und z.B. mit statistischen Algorithmen weiterbehandelt werden können. Alle Suchfunktionen des Programms basieren auf dieser Abstraktion von der Lautgestalt der Wörter zu ihrem Inhalt.
An dem einfachen Beispielsatz *madhurāmlavṛkṣam ārohati //* wird die Technik des Speicherns im folgenden grafisch verdeutlicht.

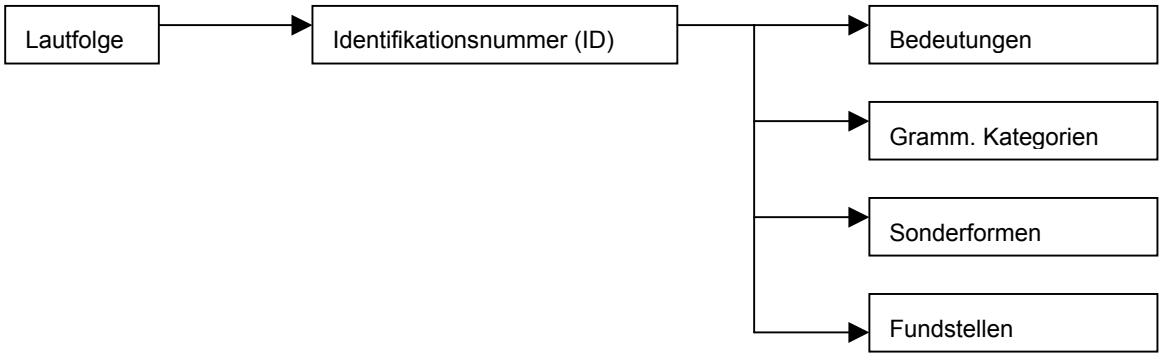


Resultat:
madhurāmlavṛkṣam ārohati -> ID_1 ID_2 ID_3 ID_4

Lexikon

Das Lexikon wird in der Programmdatenbank gespeichert, die automatisch mit dem Programm geladen wird. In früheren Versionen wurde während der Auflösung einer Lautfolge per SQL auf das Lexikon zugegriffen, was sich aber als zu zeitintensiv herausstellte - schon bei Lautfolgen von 20 Zeichen können zur Analyse tausende von Abfragen nötig sein. Der Inhalt des Lexikons wurde deshalb in eine eigene Klasse überführt, in der die Stammform des jeweiligen Wortes, seine grammatikalische Kategorie und seine ID aus der Datenbank gespeichert sind. Bei Erweiterung der Lexikon-Datenbank wird auch diese Lexikon-Klasse aktualisiert. Das Lexikon umfasst einen Großteil der Einträge des Monier-Williams mit den wichtigsten Bedeutungen. Bei Bedarf können Einträge hinzugefügt, gelöscht und verändert werden. Jede Änderung wird dabei in der Datenbank und den aus ihr extrahierten Wortlisten durchgeführt und ist damit sofort für die Sprachanalyse verfügbar.
Der interne Aufbau des Lexikons wird in der folgenden Grafik verdeutlicht. Der wichtigste Punkt ist dabei die Abstraktion von der Lautgestalt der Texte auf grammatikalische und inhaltliche Kategorien. Die Lautfolge, der Kernstamm eines Wortes, wird bei der Speicherung im Lexikon mit einem numerischen Identifikator (ID) verknüpft. Programmintern wird von nun an nicht mehr die Lautfolge, sondern nur diese ID zur Analyse der Texte und zur Suche verwendet.

Die ID zeigt auf die dynamisch erweiterbaren Sprachkomponenten des Programms. Wird z.B. einem Wort eine neue Bedeutung zugeordnet, wird diese über die ID mit dem Wort verknüpft. Sobald diese Verknüpfung hergestellt ist, ist die Bedeutung an allen Stellen des Programms mit dem gewünschten Grundwort assoziiert.



Grammatikalische Klassen

Das Programm arbeitet bei der Sanskrit-Analyse mit den drei grammatikalischen Typen ”Nomen”, ”Verbalform” und ”Verbalnomen”, die unterschiedlich implementiert sind.

Nomina werden durch ihren relevanten Stamm repräsentiert, d.h. diejenige Wortform, die beim Antritt aller möglichen Nominalendungen konstant bleibt und die sich deshalb gelegentlich von dem Stamm unterscheidet, der in Sanskrit-Grammatiken als Norm angesetzt wird. Daneben ist eine Reihe von Listen fest in den Quellcode integriert, die die möglichen Endungen einer Nominalklasse aufzählen und die zur Ausführungszeit mit dem Wortende einer vorliegenden Lautfolge verglichen werden. In diesen Endungs-Listen sind auch Nebenformen der Nominalendungen gespeichert, sofern sie im klassischen Sanskrit auftreten. Eine Sonderstellung nehmen die Pronomina ein, die in früheren Versionen ebenfalls fest im Quellcode aufgeführt waren. Sie sind in der aktuellen Version zusammen mit nominalen Sonderformen und Zahlen über eine spezielle Oberfläche zugänglich und können vom Benutzer verändert werden. Nominale Sonderformen, Pronomina und Zahlen werden bei der Sprachanalyse in ihrer Gesamtform gesucht; gespeicherte Endungen kommen hier nicht zur Anwendung.

In einem ersten Anlauf sollten finite und infinite Verbalformen genauso wie Standard-Nominalformen während der Ausführungszeit analytisch erkannt werden; eine Aufgabe, die sich wegen der verbalen Formenvielfalt des Sanskrit und der reichhaltigen Anwendung des wortinternen Sandhis bei der Verbenbildung als zumindest schwierig herausstellte. Die aktuelle Version bewegt sich zwischen reinem Lautfolgenvergleich und analytischen Fähigkeiten des Programms. Ist dem Programm die Lautform, die Verbalklasse und der Bindevokaltyp einer Verbalwurzel bekannt, kann es aufgrund fest eingebauter Regeln die Tempusformen des Verbs selbstständig bilden. Diese selbst gebildeten Formen werden dem Benutzer in einer gesonderten Oberfläche präsentiert. Sie können dort, wenn nötig, korrigiert werden (bei den meisten Verbalklassen und Tempora sind die ausgegebenen Formen glücklicherweise korrekt). Danach werden sie in der Datenbank und einer gesonderten, dynamisch erweiterbaren Liste gespeichert und stehen dem Analysevorgang zur Verfügung. Für einzelne Formen kann der Benutzer eine beliebige Anzahl von Neben- und Sonderformen angeben, die ebenfalls der Analyse zur Verfügung stehen. Ausserdem kann eine Verbalwurzel mehreren Präsensklassen zugeordnet werden.

Das Programm kann darüber hinaus präfigierte Verben verarbeiten. Dazu muss der Benutzer die zugrundeliegende einfache Verbalwurzel und die Lautform des Präfix angeben. Die Formen solcher Verben werden aus den schon gespeicherten Tempusformen des Basisverbs und dem Präfix mithilfe regulärer Ausdrücke zusammengesetzt. Momentan sind in der Datenbank die meisten Wurzeln aus Whitneys Verzeichnis mit ihren wichtigsten Tempusformen gespeichert. An präfigierten Verben ist ein Grossteil des Bestands aus dem Monier-Williams enthalten. Allerdings sind für die meisten von ihnen noch keine Tempusformen aufgenommen. Damit diese in Texten erkannt werden, müssen sie nach der beschriebenen Methode zunächst gebildet und gespeichert werden.

Verbalnomina bilden schliesslich eine Zwischenklasse zwischen Nominalformen und Verben. Sie werden bei der Erstellung eines Verbalparadigmas gebildet und mit den Verbalformen fest in der Datenbank gespeichert. Handelt es sich um finite Formen,

wird wie bei den Nomina nur ihr relevanter Stamm aufgenommen. Während der Sprachanalyse werden finite Verbalnomina wie Nomina behandelt.

Künftige Änderungen an der Sprachanalyse

Die Sprachanalyse arbeitet fehlerfrei, sofern ihr die zu analysierenden Lemmata innerhalb einer Lautfolge vollständig bekannt sind, d.h. das Lexikon hinsichtlich der untersuchten Wörter vollständig ist. Damit kann die Sprachanalyse auch eingesetzt werden, um Fehler in einem Text zu erkennen.
Eine denkbare Erweiterung ist ein fehlertoleranter Analysealgorithmus: Bisher verwirft die Sprachanalyse alle gefundenen Teilauflösungen, wenn eine Lautfolge sich nicht vollständig analysieren lässt. Will der Benutzer die Lautfolge trotzdem analysiert in die Text-Datenbank aufnehmen, muss er die Einzelbestandteile per Hand identifizieren. In einer künftigen Implementierung könnte das Programm die schon gefundenen Elemente speichern und dem Benutzer zur Ergänzung präsentieren; das Problem bei diesem Schritt ist weniger die algorithmische Grundlage als ein nachvollziehbares Design der entsprechenden Benutzerschnittstelle.

In künftigen Programmversionen sollte zudem die wahrscheinlichste Auflösung einer Textstelle sicherer erkannt werden. Ein Beispiel: In indischen Pflanzenlexika wird eine Pflanze namens *madhurā* erwähnt; anderen Pflanzen wird die Eigenschaft *madhura* (”süss”) zugesprochen. Ob an einer gegebenen Stelle der Pflanzenname oder die Eigenschaft als Femininum erwähnt ist, lässt sich mit dem geschilderten Algorithmus, der nur das bisherige Auftreten eines Lemmas bewertet, nicht immer sicher entscheiden. Das Programm müsste dahingehend erweitert werden, dass der Kontext in die Bewertung einer Wortauflösung einfliesst (ein Problem, mit dem auch Indologen bei Übersetzungen gelegentlich zu kämpfen haben). Denkbar sind verschiedene Wege:

- Erweiterung der bisher im HMM benutzten Bi- auf n-Gramme: Das Programm speichert in einem vorbereitenden Schritt typische Wortkombinationen. Tritt z.B. das Lemma *madhura* (“süss”) häufig in Verbindung mit den Eigenschaften *kaṭu* (”scharf”), *tikta* (”bitter”) u.ä. auf und befinden sich diese Wörter an einer gegebenen Textstelle ebenfalls in der Nähe des Wortes, sollte die Entscheidung eher zu *madhura* = ”süß” als zu dem Pflanzennamen tendieren. Mathematisch entspricht dieser Ansatz einer Erweiterung des oben dargestellten HMM von Wortpaaren auf Wortsequenzen grösseren Umfangs, aus denen die statistische Information zur Bewertung einer Auflösung gewonnen werden. Dieser Weg ist allerdings erst gangbar, wenn eine deutlich grössere Menge grammatikalisch und lexikalisch analysierter Sanskrittexte zur Verfügung steht, da die Zahl möglicher n-Gramme mit ihrer Länge nicht linear wächst, wie man sich an einem einfachen Beispiel verdeutlichen kann. So wären bei einem Lexikonumfang von 10 Wörtern 10x10=100 Wortpaare (Bigramme) zu untersuchen; dehnte man nach der vorhergehenden Überlegung die Dimension der n-Gramme auf 3 aus, wären schon 10x10x10=1000 Kombinationen möglich (allgemein: bei einem Umfang d des Wörterbuchs sind d^n n-Gramme zu speichern/berücksichtigen; eine Übersicht über die Problematik z.B. in Schmid (1994), 2ff.). Das Dilemma kann im einfachsten Fall durch eine massive Vergrösserung der Textdatenbank und damit des Rohmaterials zur Generierung von Übergangswahrscheinlichkeiten gelöst werden.

- Einbeziehen von Bedeutungen: Ein anderer Lösungsansatz könnte die Bedeutungen innerhalb einer untersuchten Passage benutzen, um eine möglichst korrekte Auflösung zu generieren. Momentan wird auf Basis der schon analysierten Texte manuell ein semantisches Netz erstellt, aus dem sich – ab einem gewissen Grad der Vollständigkeit, der in frühestens einem Jahr erreicht sein wird – Informationen über die inhaltliche Ähnlichkeit innerhalb verschiedener Auflösungspfade auslesen liessen. Am gegebenen Beispiel: Finden sich im Umkreis des Wortes *madhurā* v.a. medizinische Informationen oder Beschreibungen anderer *rasas*, tendiert das Programm zur Auflösung als Adjektiv ”süss” (Femininum des Adjektivs). Treten dagegen gehäuft Pflanzennamen auf, wird der Pflanzenname *madhurā* bevorzugt.

Sprachanalyse und Suchfunktionen

Der Hauptantrieb, Sanskritwörter wie beschrieben lexikalisch zu analysieren, besteht in der Absicht, sie später schnell und v.a. treffsicher wiederzufinden. Trotz der Gefahr von Wiederholungen sollen an dieser Stelle noch einmal bisherige Methoden mit der hier implementierten verglichen werden, um den Unterschied zu den üblichen Verfahren aufzuzeigen.

Die bislang verbreitetsten Ansätze, Sanskrittexte per Computer nach Lemmata zu durchsuchen, arbeiten u.a. nach folgenden Methoden:

- Der relevante Teil eines Wortes wird mittels einer Volltextsuche im unaufgeteilten Text gesucht. Ein Beispiel: Gesucht ist das Lemma *īśvara*. Damit es auch im Kompositum *rājeśvara* erkannt werden kann, muss als Suchtext **śvar** angegeben werden. Diese Einschränkung der Lautgestalt erzeugt in den meisten Fällen eine Reihe von Fehltreffern. Ausserdem ist eine Volltextsuche in umfangreichen Textcorpora nicht besonders schnell.
- Der Text wird mit aufgelösten Sandhis eingegeben. *rājeśvara* würde also als *rāja_īśvara* oder in einer anderen Codierung aufgenommen. Abgesehen davon, dass derartig codierte Texte nicht besonders gut lesbar und daher meist noch von einer unaufgelösten Form begleitet sind, gewährt auch diese Form der Aufteilung keinen sicheren Zugriff auf gesuchte Lemmata: *rāja_* z.B. ist die Stammform des Nomens *rājan*, das in dieser Form gerade nicht im Text vorkommt. Ausserdem werden lexikalische Doppeldeutigkeiten nicht vermieden - man denke an das oben zitierten Beispiel *madhurā* / *madhura*. Schliesslich ist für die manuelle Codierung der Sandhis ein erheblicher Arbeitsaufwand erforderlich, den man auch für das Taggen der Texte verwenden kann.

Beide Methoden sind oft nur für die Suche nach einem Eintrag geeignet. Gemeinsames Auftreten von Wörtern oder weitergehende Kriterien können in vielen Fällen nicht untersucht werden.

Mit der hier gebotenen Möglichkeit, Wörter auf Basis eines Lexikons und einer Grammatik zu analysieren, entfallen die beschriebenen Methoden. Der Zugriff auf die Positionsinformation für ein bestimmtes Lemma ist z.B. über das Lexikon möglich. In der einfachsten Form wählt der Benutzer ein Lemma aus dem Lexikon aus. Das Programm schickt die ID des gewählten Wortes an die Datenbank, die per SQL das Auftreten des Wortes untersucht und das Ergebnis an die Lexikon-Oberfläche zurückschickt. Darüber hinaus ist durch die Abstraktion von der Lautgestalt der Texte eine Vielzahl komplexerer Suchmethoden und Analysemethoden denkbar und

teilweise auch schon implementiert, von denen abschliessend zwei vorgestellt werden sollen.

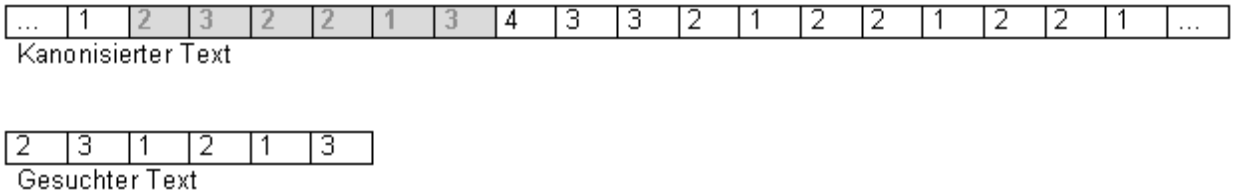
Anwendungsbeispiele für die Suchfunktionen:

Suche nach Zitaten

Der Suche nach Passagen, die einer vorliegenden Textstelle ähneln, stellt sich eine Reihe von definitorischen und programmiertechnischen Problemen entgegen:

- **Definition eines Ähnlichkeitsmasses**: Wie ähnlich sind sich zwei Textpassagen? Werden nur solche Passagen als ähnlich angesehen, in denen dieselben Wörter auftreten, oder sollen auch inhaltlich ähnliche Passagen als Treffer gewertet werden? Ein wichtiger Aspekt ist weiterhin die Reihenfolge der auftretenden Wörter.
- **Definition eines Ähnlichkeits- und Abstandsmasses**: Welche maximale Distanz darf zwischen Wörtern einer untersuchten Textpassage auftreten, damit diese noch als Treffer aufgefasst wird? Gerade in versifizierten Texten haben Satzzeichen (z.B. Dandas) nicht die normative Kraft, die ihnen in modernen europäischen Texten zugesprochen wird. Wie kann weiterhin, wenn inhaltlich ähnliche Passagen gesucht werden sollen, ein Ähnlichkeitsmass zwischen zwei Textstellen definiert werden?
- **Synonymie von Wörtern**: Das Sanskrit macht selbst in technischen Texten intensiven Gebrauch von synonymen Begriffen. Damit solche Passagen als ähnlich erkannt werden, muss zunächst der Wortschatz manuell überarbeitet und nach Synonymen gegliedert werden.
- **Geschwindigkeit der Suchfunktion**: Wird die Suchprozedur - noch dazu unter Einbeziehung der Synonymie - nur auf Basis von SQL ausgeführt, kann eine Suche nach identischen Passagen viele Minuten in Anspruch nehmen. Die Texte müssen also in eine schneller durchsuchbare Darstellung transformiert werden.

Die Implementierung dieser Suchfunktion berücksichtigt v.a. das Problem der Suchgeschwindigkeit und der Ähnlichkeit zwischen zwei Passagen. Vor Aktivierung dieser Suchfunktion werden deshalb sämtliche in der Datenbank gespeicherten Texte in eine lineare Form überführt ("Kanonisierung"). Bei diesem Kanon handelt es sich um einen Array, in dem nur noch die lexikalische und morphologische Information der Texte, d.h. die ID der Lemmata im Lexikon und Angaben zur Grammatik, codiert sind. Die Lautgestalt der Texte spielt keine Rolle mehr.
Die eigentliche Suchprozedur wird in der folgenden Grafik angedeutet.



Die obere Reihe stellt vereinfacht das Ergebnis der Kanonisierung dar, die Überführung der in der Textdatenbank gespeicherten Informationen in eine lineare

Struktur. Ziel der Suche ist es, innerhalb des umfangreichen kanonisierten Textes Ausschnitte zu finden, die optimal zum gesuchten Text (untere Reihe) passen. Ein möglicher Treffer im Kanon ist hellgrau hinterlegt.
Zur Implementierung des Ähnlichkeitsmaßes greift die Suche die Idee eines Fuzzy-Stringvergleichs auf. Sobald in der kanonisierten Sequenz eines der ersten gesuchten Symbole auftritt (im Beispiel: das Symbol "2"), wird diese Position als möglicher Start markiert und von hier ausgehend nach rechts gesucht. Stimmt ein Symbol des kanonisierten Textes nicht mit dem Ausgangstext überein, wird entweder ein interner Zähler um eins zurückgesetzt (in der Zielsequenz fehlt ein Symbol der gesuchten Sequenz) oder mit der Suche an der nächsten Position fortgefahren (Ziel- und gesuchte Sequenz unterscheiden sich an einer Position). Wird eine festgelegte Zahl von Fehlern überschritten, bricht das Verfahren an dieser Stelle ab.
Als Ähnlichkeitsmaß wird die Anzahl der Fehler im Verhältnis zur Länge der gesuchten Sequenz angesetzt.
Der hier skizzierte Ansatz ist im Programm implementiert und liefert auch Resultate. Allerdings leidet er an zwei Mängeln:

- Die Suche ist reihenfolgen-abhängig, d.h. es werden nur solche Passagen ausgegeben, in denen die gesuchten Wörter – mit Einfügungen und Auslassungen – in identischer Reihenfolge auftreten. Für das Sanskrit, in dem die Wortreihenfolge aufgrund der reichhaltigen Flexion eine eher untergeordnete Rolle spielt, ist dieser Punkt verbesserungswürdig. Ansätze dazu können im nächsten Kapitel (Generierung von Verweisen) gefunden werden.
- Konzeptuell ähnliche Passagen werden auf diese Weise nicht gefunden.

Das Finden ähnlicher Passagen könnte auf folgende Weise realisiert werden: Das Programm analysiert zunächst das Umfeld der Ausgangspassage und weist dabei jedem Wort auf Basis der Informationen eines integrierten semantischen Netzes mögliche Bedeutungen zu. Dieses Verfahren, eine automatische inhaltliche Analyse einer Passage, könnte nach ähnlichen Methoden wie die lexikalische oder morphologische Analyse eines Satzes arbeiten und ist in einer hochgradig experimentellen (d.h. nicht öffentlich zugänglichen) Version schon implementiert. Nach Festlegung möglicher Bedeutungen wird in dem semantischen Netz nach Synonymen der enthaltenen Wörter gesucht. Diese Synonyme werden als Variationen für jede Position mit den Ausgangswörtern an die Suchabfrage geschickt (Beispiel: Tritt in der Ausgangspassage das Wort „śiva“ auf, könnten über eine Synonym-Suche im semantischen Netz auch die Begriffe „kapardin“, „kāla“, „rudra“ ... in die Suche einbezogen werden).

Automatisierte Generierung von Verweisen

Die im vorigen Abschnitt vorgestellte Prozedur leidet unter zwei Mängeln:

- Abhängigkeit von der Wortreihenfolge: Diese Einschränkung sollte bei einer stark flektierenden Sprache mit relativ freier Wortstellung nicht oder nur begrenzt gelten.
- Explizite Aufforderung zur Suche: Der Benutzer muss bei jeder Passage selbst entscheiden, ob sich eine Suche nach parallelen Stellen lohnt. Angemessener wäre eine Reihe von vorgenerierten und vor allem inhaltlich relevanten Verweisen, die in den Basistext eingefügt sind und damit sofort zur Verfügung stehen.

Zur Umgehung dieser Schwierigkeiten wurde die Grundidee des Data Minings aufgegriffen. Eine der häufigsten Anwendungen dieser Techniken ist die sogenannte

Warenkorbanalyse (das folgende nach Beierle (2000), 137ff.). Die Datenbasis wird dabei z.B. von den elektronisch aufgezeichneten Einkäufen der Kunden eines Kaufhauses gebildet. Für jeden Einkauf – im aktuellen Kontext als Transaktion bezeichnet - werden die gekauften Artikel notiert. Die Aufgabe des Data Minings ist es nun, automatisch relevante Beziehungen zwischen den Artikeln zu generieren, indem zuerst häufige Teilmengen (Itemmengen) innerhalb der Transaktionen gefunden und diese dann auf ihre inhaltliche Relevanz hin untersucht werden. Wenn die Analyse z.B. ergibt, dass Kunden, die Shampoo kaufen, in der grössten Zahl der Fälle auch Lockenwickler erstehen, sollten diese beiden Artikel dann in der Verkaufsfläche nahe beieinander platziert werden.
Auf die Indologie übertragen können die Techniken des Data Minings z.B. dazu benutzt werden, aus analysierten Sätzen (= Transaktionen) häufige Itemmengen zu extrahieren und basierend auf der inhaltlichen Relevanz dieser Mengen Verweise zwischen den Texten zu generieren (für Freunde des Data Minings: Assoziationsregeln werden hier also nicht erstellt.)

Im Detail geht die Verweisgenerierung wie folgt vor:

- Die Sätze werden zunächst bereinigt. Sehr häufige Wörter (z.B. Konjunktionen, die Verben “as”, “bhū”, ...) werden entfernt, mehrfach in einem Satz auftretende Wörter werden auf eines reduziert. Schliesslich werden die IDs der Wörter im Satz nach aufsteigender Grösse geordnet, wodurch die Abhängigkeit von der Wortreihenfolge beseitigt ist.
- Die häufigen Itemmengen werden aus den Sätzen mit dem Apriori-Algorithmus extrahiert. Dieser Algorithmus bestimmt aufgrund der lexikalischen Information der Datenbank zunächst die einelementigen Itemmengen (häufige Wörter). Auf Basis dieser Ausgangsinformation wird die Datenbasis rekursiv nach n+1-elementigen Mengen durchsucht, bis keine Änderung in der Anzahl der ausgegebenen Itemmengen mehr erfolgt.
- Die im vorhergehenden Schritt gefundenen Mengen – besonders die zweielementigen, von denen eine grosse Zahl auftritt - müssen jetzt auf ihre inhaltliche Relevanz hin untersucht werden. Das Programm nutzt dazu die in Clifton (2001), 11 vorgeschlagene wechselseitige Information (mutual information). Wenn dieser Wert eine festgelegte Schwelle überschreitet, wird der Verweis in die Datenbank aufgenommen.

Der hier skizzierte Algorithmus generiert bei einem Datenbankumfang von ca. 250000 Wörtern rund 30000 Verweise, von denen die meisten bei manueller Prüfung auch inhaltlich relevant erscheinen. So wird z.B. die Diskussion der *tattvas* in der Sāṃkhya-Philosophie zuverlässig über die Textgrenzen hinweg verfolgbar. Die Ausführungszeit liegt unter zwei Stunden, wobei ein Grossteil der Zeit für die Aktualisierung der Datenbank per SQL benötigt wird.