Automatisierte, prozessbegleitende Identifizierung der Problemlösestrategien Lernender unter Verwendung von Mustererkennungsmethoden

Ulrich Kiesmüller

Didaktik der Informatik
Friedrich-Alexander-Universität Erlangen Nürnberg
Martensstraße 3
91058 Erlangen
ulrich.kiesmueller@informatik.uni-erlangen.de

Abstract: Im Informatikunterricht eingesetzte Lern- und Programmierumgebungen geben den Benutzenden Feedback in Form von Systemmeldungen, die durch Programmfehler ausgelöst und gesteuert werden und oft nur technische Hinweise enthalten ohne Bezug zum Problemlöseprozess der Lernenden. Um diese Rückmeldungen nicht nur an das Faktenwissen der Lernenden, sondern auch an ihr prozedurales Wissen zu adaptieren, müssen deren Vorgehensweisen bei der Problemlösung automatisiert prozessbegleitend identifiziert werden. Dieser Artikel beschreibt einen Weg, dieses Ziel unter Verwendung von Mustererkennungsmethoden zu erreichen. Die in einer Studie von 65 Lernenden im Alter von 12 bis 13 Jahren erhobenen Daten werden verwendet, um ein auf verborgenen Markow-Modellen basierendes Klassifikationssystem zu trainieren. Dieses System wird integriert in die Programmierumgebung und ermöglicht somit die automatisierte Identifizierung der Vorgehensweise der Lernenden. In diesem Artikel werden die Funktionsweise der automatisierten prozessbegleitenden Identifizierung beschrieben und Ergebnisse aus den erhaltenen Daten diskutiert.

1 Motivation

Informatiklehrkräfte setzen oft visuelle Programmierung unterstützende Umgebungen ein, wie zum Beispiel Scratch [Ma04] oder Kara [Re03]. In einigen Bundesländern (z. B. Bayern) werden Grundlagen der Algorithmik bereits in der 7. Jahrgangsstufe gelehrt. Während ihrer ersten Schritte im Bereich der Programmierung lösen Lernende kurze Aufgaben mit den eingesetzten Programmierumgebungen. Selbst wenn ihnen eine bestimmte Vorgehensweise zur Lösungsfindung gelehrt wurde, können bei den Lernenden verschiedene Problemlösestrategien beobachtet werden. Aus konstruktivistischer Sicht sollen die Lehrenden [Be98] das bestehende (prozedurale) Wissen der Lernenden nicht außer Acht lassen, auf die individuellen Vorgehensweisen der Lernenden eingehen und sie unter deren Berücksichtigung bei einer selbstständigen Problemlösung unterstützen. Im Sinne von [Dr04] ist die Gestaltung von glaubwürdigem motivationalem Feedback möglich, durch das bei den Lernenden Motivation erzeugt, gehalten oder gar gesteigert wird.

Dieser Artikel ist wie folgt aufgebaut. In Kapitel 2 werden Vorüberlegungen bezüglich Problemlösestrategien, Lern- und Programmierumgebung sowie Kategorisierung von Lerner-System-Interaktionen dargelegt. An die in Kapitel 3 folgende Beschreibung der bereits mit Automaten-Kara aufgenommenen Daten schließt sich ein kurzer Überblick über die beobachteten Problemlösestrategien an. Im Kapitel 4 wird die Mustererkennungsmethode beschrieben, die zur automatisierten, prozessbegleitenden (d. h. bereits während die Lernenden mit dem System interagieren) Identifizierung eingesetzt wird. Eine Analyse der Ergebnisse des Einsatzes der vorgestellten Methode und erste Schlussfolgerungen schließen sich im Kapitel 5 an. Abschluss bildet ein Ausblick auf zukünftige Arbeiten im Kapitel 6.

2 Vorüberlegungen

2.1 Problemlösestrategien

Problemlösen bedeutet aus psychologischer Sicht die Suche eines korrekten Weges durch den Problemraum, der sich zwischen einem (unerwünschten) Ausgangszustand und einem (erwünschten) Endzustand aufspannt [Ma92, CG85, NS72]. Wie in [Ki09] dargestellt, sind bei Lernenden während der Lösung der gestellten Aufgaben verschiedene Problemlösestrategien zu beobachten. Diese lassen sich aufteilen in die (vor-)strukturierten Methoden *top down* und *bottom up* sowie die Methoden *hill climbing* und *trial and error*, bei denen die Lernenden jeweils nur einen einzelnen Schritt der Lösung betrachten.
Abbildung 1 zeigt die Grobstrukturierung des Gesamtproblems als Aufteilung in Teilprobleme, die innerhalb der Feinstrukturierung weiter in Unterprobleme zerlegt werden, bevor sich letztendlich ohne weitere Verzweigungen die Teillösungen anschließen. Hierbei ergibt sich eine baumähnliche Struktur, die von Lernenden, die eine *top down*-Strategie anwenden, Ebene für Ebene wie bei einer Breitensuche abgearbeitet wird (Pfeile in Abb. 1). Somit ist eine (ebenenweise) sukzessive Annäherung an die Gesamtlösung gewährleistet.
Bei der *bottom up*-Strategie wird eine Tiefensuche durchgeführt (Abb. 2). Die Gesamtlö-

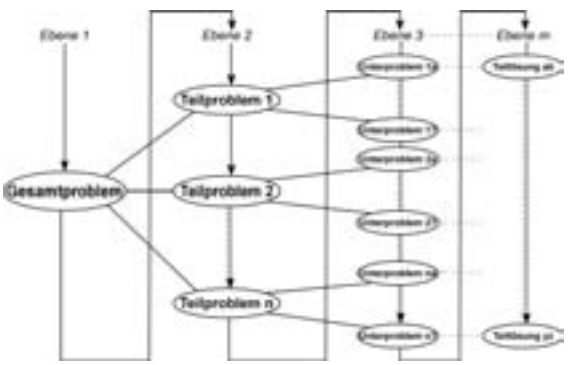

Abbildung 1: Überblick über den Lösungsablauf bei der *top down*-Strategie

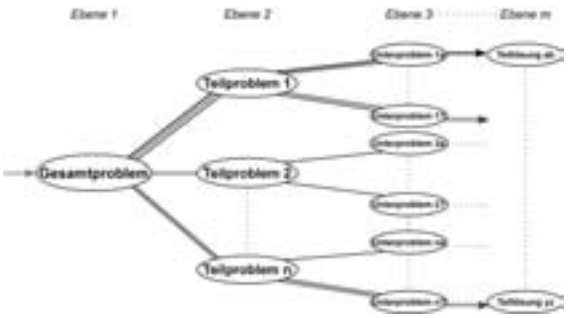

Abbildung 2: Überblick über den Lösungsablauf bei der *bottom up*-Strategie

sung entsteht, wenn das letzte Teilproblem vollständig gelöst ist. Lernende mit einer *hill climbing*-Strategie strukturieren weder das gesamte noch die Teilprobleme, haben aber eine klare Vorstellung von der erwünschten Arbeitsweise des Programms. Sie kontrollieren ihre Lösungsversuche aktiv durch wiederholte Programmausführungen. Hierfür benötigen sie nicht unbedingt Systemfehlermeldungen der Programmierumgebung. Sie brechen den Programmlauf bei unerwartetem Programmverhalten häufig ab („Schleifen“ in Abb. 3). Ihr Fokus liegt jeweils nur auf einem Schritt, dessen optimaler Lösung und der Suche nach dem sinnvollsten nächsten Schritt. Durch ihre klare Vorstellungen vom Programmablauf, ihre Suche nach dem jeweilig nächsten Schritt (Abb. 3: durchgezogene Pfeile – gestrichelt: alternative Fortsetzungen) und anschließend dessen (optimierter) Lösung werden sie jeden zur Lösung notwendigen Schritt durchlaufen und sich sukzessive in allen Teilzweigen an die Teillösungen und somit an die Gesamtlösung der Aufgabenstellung annähern.
Im Gegensatz zu allen bisher beschriebenen Vorgehensweisen liegt bei der *trial and error*-Strategie (Abb. 4) keine systematische Annäherung an die Teillösungen vor. Die Gesamtlösung entsteht (wenn überhaupt) eher „zufällig“. Die Aufmerksamkeit der Lernenden liegt hier wiederum jeweils auf einem einzelnen Schritt. Sie warten bei dieser Vorgehensweise bei Testläufen stets so lange, bis sie eine Systemfehlermeldung erhalten und versuchen jeweils anschließend, den dafür ursächlichen Fehler zu beheben. In dieser Gruppe von Lernenden tritt häufig die typische unerwünschte Situationen auf, dass die Lernen-

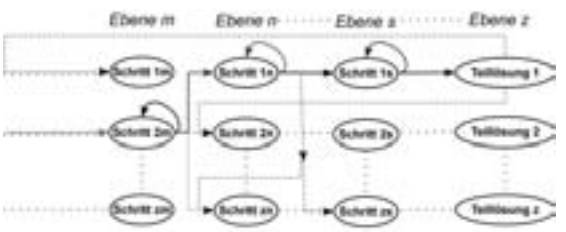

Abbildung 3: Überblick über den Lösungsablauf bei der *hill climbing*-Strategie

den behaupten, die Aufgabe gelöst zu haben (weil keine Systemfehlermeldungen mehr auftreten), aber im Sinne der Aufgabenstellung keine Lösung vorliegt.

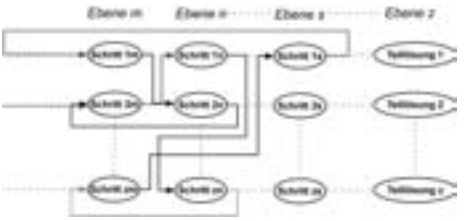

Abbildung 4: Überblick über den Lösungsablauf bei der *trial and error*-Strategie

2.2 Lern- und Programmierumgebung

In der Informatik ist Kreativität ein wichtiger Faktor, um bei Lernenden Interesse zu wecken, Motivation zu erzeugen und zu erhalten [Ro07]. In [Sh07] finden sich als Kriterien für kreativitätsfördernde Software u. a. Unterstützung hilfreichen Feedbacks, Zulassen straffreien Experimentierens sowie visuellen Programmierens und Erlauben des schrittweisen Lösens der gestellten Aufgaben. Diese Bedingungen werden von Umgebungen wie Kara in seiner automatenbasierten Version [Re03] erfüllt. Eine typische Aufgabe („Kara und die Blätter“), mit der die Lernenden (auch bei den unten beschriebenen Untersuchungen) konfrontiert werden, ist es, Kara auf seinem Weg zu einem Baum, der sich geradlinig vor ihm befindet, ein Muster aus Kleeblättern invertieren zu lassen. Das zugehörige Programm muss unter Verwendung der Sensoren des Käfers mittels endlicher Automaten modelliert werden. Bei den hier beschriebenen Studien wurden die Lerner-System-Interaktionen bei Automaten-Kara protokolliert und deren zeitliche Abfolge ausgewertet.

2.3 Kategorisierung der Lerner-System-Interaktionen

Bei den im Rahmen des hier beschriebenen Forschungsprojekts durchgeführten Studien wird nur eine Auswahl von Lerner-System-Interaktionen (LSI) aufgezeichnet, nämlich für den Problemlöseprozess relevante (Tab.1). Wie in [Ki09] erläutert, wird die *Strukturierung des Problems als solche* durch die LSI „Bearbeitung und Änderung des Automaten“ der Kat. 0 zugeordnet. Die *Feinstrukturierung des Problems* in verschiedene Teilprobleme ist gleichbedeutend mit den LSI „Erzeugung“ und „Bearbeitung bedingter Verzweigungen“ (Kat. 1, 2, und 3). Wiederholungen, Verzweigungen und Sequenzen von Befehlssequenzen rangieren zwischen Feinstrukturierung und *Lösung eines Teilproblems*. Sie werden durch die Bearbeitung von Transitionen umgesetzt. Die Lösung eines Teilproblems wird repräsentiert durch die Bearbeitung von Sequenzen (Kat. 4). Zusätzliche LSI sind Programmausführungen („play“, Kat. 5) zur Überprüfung der Korrektheit von Programmen, sowie deren Abbruch („stop“, Kat. 6). Kat. 7 schließlich fasst alle Systemfehlermeldungen zusammen, so dass sich insgesamt acht Kategorien von LSI ergeben.

Kat.	LSI	Kat.	LSI
0	STATE_ADDED STATE_REMOVED	4	TRANSITION_COMMAND_ADDED TRANSITION_COMMAND_REMOVED
1	TRANSITION_TO_CHANGED	5	PLAYING
2	TRANSITION_ADDED TRANSITION_REMOVED	6	STOPPED
3	TRANSITION_INPUT_CHANGED STATE_SENSORS_SET	7	AMBIGUOUS_TRANSITION_EXCEPTION COMMAND_EXCEPTION NO_TRANSITION_EXCEPTION NO_START_STATE_EXCEPTION

Tabelle 1: Kategorisierung und Gruppierung der relevanten Lerner-System-Interaktionen (LSI).

3 Untersuchungsdaten

3.1 beobachtete individuelle Vorgehensweisen

Die oben eingeführte Automaten-Kara-Software wurde durch ein Modul erweitert, das alle LSI protokolliert und diese Daten entweder in einer Datei speichert oder direkt weiter verwendet (Kap. 4.3, Abb. 12). Die bei an zwei bayerischen Gymnasien durchgeführten Studien aufgenommenen Daten [Ki09] wurden als Ausgangsdaten für die hier beschriebenen Untersuchungen verwendet. Aus 188 Einzelsitzungen (je eine Aufgabe pro Lernendem) ergaben sich ca. 13000 protokollierte lösungsrelevante LSI. Unter Verwendung der Protokoll-Dateien, automatisch protokollierten Bildschirmaufnahmen und zusätzlichen mittels der Methode des „lauten Denkens“ und Interviews erhobenen Daten konnten die Vorgehensweisen der Lernenden in den einzelnen Sitzungen identifiziert werden. Wie sich diese bei einer Aufgabenlösung mit Automaten-Kara darstellen, wird nun beschrieben.
Top down – Zuerst wird das Problem als Ganzes strukturiert. Hierbei fügen die Lernenden Zustände und Teilzweige (Bild 1 und 2 in Abbildung 5) hinzu und bearbeiten diese. Anschließend stellen sie Karas Sensoren ein (Bild 3 in Abbildung 5), was einer Feinstrukturierung des Problems entspricht. Abschließend ergänzen sie die Befehle (Bild 4 in Abbildung 5), um alle Teilprobleme und damit die gestellte Aufgabe insgesamt zu lösen.
Bottom up – Die Lernenden erzeugen und bearbeiten zuerst nur einen Teilzweig, in den sie anschließend die Befehle einfügen, um dieses Teilproblem vor der Bearbeitung des nächsten Teilzweiges zu lösen (Bild 1 in Abbildung 6). Diese Schritte werden so oft wie-

Abbildung 5: Screenshot-Folge (Ausschnitt) einer *top down*-Vorgehensweise

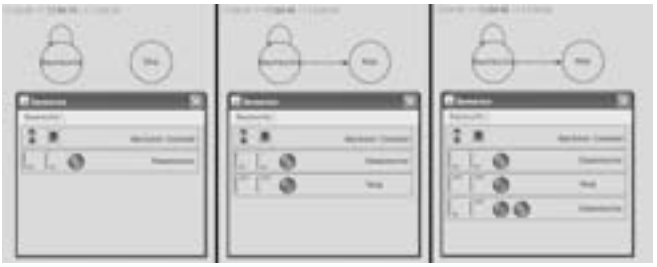

Abbildung 6: Screenshot-Folge (Ausschnitt) einer *bottom up*-Vorgehensweise

derholt (Bild 2 und 3 in Abbildung 6) bis das letzte Teilproblem (nicht mehr in Abbildung 6 enthalten) und somit die gesamte gestellte Aufgabe gelöst ist.

Hill climbing – Die Lernenden konzentrieren sich jeweils auf einen einzelnen Programmschritt. Sie haben eine klare Vorstellung vom gewünschten Programmablauf und starten die Ausführung, um den aktuell von ihnen bearbeiteten Programmschritt zu überprüfen. Diese brechen sie sofort ab, wenn sie erkennen können, ob das Programm an dieser Stelle korrekt läuft oder nicht (z. B. viermaliger Programmlauf zwischen Bild 2 und 3 in Abbildung 7). Gegebenenfalls korrigieren sie anschließend ihre Fehler (in der zu gehörigen log-Datei zwischen Bild 2 und 3 vor erneuter Korrektheitsprüfung durch Programmtestlauf) oder suchen nach dem nächsten zu bearbeitenden Schritt. Diese Prozedur wird solange wiederholt bis die Gesamtlösung der gestellten Aufgabe erreicht wurde. In manchen Fällen kommen die Lernenden nicht dazu (wie z. B. zwischen Bild 3 und 4 in Abbildung 7) den Programmlauf wegen eines unerwünschten Effekts selbst abzubrechen, weil sofort im ersten Schritt eine Systemfehlermeldung auftritt. Auf diese reagieren sie wie oben beschrieben mit der Verbesserung und erneuten Überprüfung des aktuellen Programmschrittes.

Trial and error – Auch hier konzentrieren sich die Lernenden nur auf einen einzelnen Programmschritt und führen nach jedem ihrer Programmier- oder Verbesserungsschritte jeweils das Programm aus. Im Gegensatz zur gerade beschriebenen Vorgehensweise machen sie sich vorher keine klare Vorstellung vom erwünschten Programmablauf und brechen diesen selbst ab, sondern warten vielmehr bis bzw. ob das System den Programmlauf mit einer Fehlermeldung abbricht. Den gemeldeten Fehler versuchen sie zu korrigieren

Abbildung 7: Screenshot-Folge (Ausschnitt) einer *hill climbing*-Vorgehensweise

und wiederholen den beschriebenen Prozess, um sich der korrekten Lösung anzunähern. Zwischen den Bildern 3, 4 und 5 sowie 6 und 7 der Abbildung 8 werden jeweils Fehlermeldungen vom System ausgegeben. Auch direkt nach der Aufnahme des Screenshots 8 wird der Lernende mit einer Systemrückmeldung konfrontiert. Im dargestellten Fall liegt sicher nicht die *trial and error*-Variante des *systematischen* Ausprobierens aller Möglichkeiten vor, denn die vom Lernenden jeweils geänderten Programmschritte sind, wie in den log-Dateien erkennbar, meist nicht diejenigen, an denen sich die Fehlerstelle befindet. Für den Lernenden wird es somit sehr schwer, sich an eine korrekte Lösung anzunähern.

Abbildung 8: Screenshot-Folge (Ausschnitt) einer *trial and error*-Vorgehensweise

3.2 Datenaufbereitung

In den Protokoll-Dateien wurden bei der manuellen Aufbereitung alle Vorkommen der die oben beschriebenen Vorgehensweisen repräsentierenden Muster mit XML-ähnlichen Markierungen versehen. Abbildung 9 zeigt den Auszug aus einer log-Datei mit einem markierten Muster der *bottom up*-Vorgehensweise. Die weiteren zu entnehmenden Daten sind Zeitstempel, Bezeichnung der gestellten Aufgabe und jeweils gerade durchgeführte LSI. Um eine Vollständigkeit der Labelung zu erreichen werden alle (Sequenzen von) LSI, die keiner der vier aufgeführten Problemlösestrategien zugeordnet werden können, als „unidentifiziert"-Muster markiert. In einem nächsten Schritt werden anhand der Markie-

```
<HMMBU>
12:04:43    Kara und... (leicht)    TRANS_ADDED
12:04:43    Kara und... (leicht)    TRANS_INPUT_CHANGED
12:04:44    Kara und... (leicht)    TRANS_INPUT_CHANGED
12:04:45    Kara und... (leicht)    TRANS_INPUT_CHANGED
12:04:49    Kara und... (leicht)    TRANS_COMMAND_ADD
</HMMBU>
```

Abbildung 9: Auftreten eines XML-ähnlich markierten *bottom up*-Musters in einer log-Datei

```
2; 3; 3; 3;
3; 3; 4; 4; 4; 4; 4;
0; 2; 3; 3; 3; 4;
```

Abbildung 10: Beispielssequenzen von LSI für das *bottom up*-Muster

rungen und der LSI-Kodierung und -Gruppierung (Tab. 1) die Daten aus den log-Dateien von einem speziell dafür entwickelten Softwaremodul in Sequenzen von LSI-Kodierungen umgewandelt. Dies ist notwendige Voraussetzung für die Kapitel 4 beschriebene automatisierte Identifizierung der Muster. Abbildung 10 zeigt drei Instanzen des *bottom up*-Musters mit verschiedener Länge und ähnlicher, aber an Einzelstellen unterschiedlicher Struktur.

3.3 Statistische Ergebnisse

Erstes Resultat ist die Möglichkeit, eine LSI-Sequenz einer von vier Problemlösestrategien oder dem „unidentifiziert"-Muster zuzuordnen. Des Weiteren ist bemerkenswert, dass Lernende im Laufe der Lösungserstellung für eine Aufgabe gelegentlich die Vorgehensweise wechseln und im Laufe der Zeit Kennzeichen verschiedener Problemlösetypen aufzeigen. In [FS88] wird hierzu erläutert, dass es grundlegende Problemlösetypen gibt, aber bei Lernenden unterschiedliche individuelle Ausprägungen davon beobachtbar sind. Für die hier beschriebenen Studien bedeutet dies, dass in der Chronologie der LSI eines Lernenden bei der Lösung einer Aufgabe verschiedene Vorgehensmuster identifiziert werden können. Die Wahrscheinlichkeit, zu welcher Vorgehensweise der Lernende jeweils wechselt, ist variabel. In Tabelle 2 sind als Wahrscheinlichkeiten für jeden möglichen Wechsel zwischen zwei Mustern die relativen Häufigkeiten, die sich aus den aufgenommenen Daten ergaben, aufgeführt. Das UI-Muster muss hierbei separat betrachtet werden, da es üblicherweise als Übergang zwischen zwei Vorgehensmustern auftritt. Bemerkenswert ist, dass Lernende, deren Chronologie ein zu einer bestimmten Problemlösestrategie gehörendes Muster enthält, mit sehr hoher Wahrscheinlichkeit dieses Vorgehensmuster mehr als einmal zeigen (Diagonalenwerte in Tab. 2) – mit Ausnahme des *top down*-Musters, das nicht wiederholt, sondern oft gewechselt wird zu *bottom up* (näheres dazu in Kap. 5). Zumindest wenn Lernende mit einem neuen Lösungsversuch beginnen, verwenden sie wieder die selbe Strategie wie beim ersten Ansatz, was auf die Existenz einer *individuell* bevorzugten Strategie hindeutet.

	TD	**BU**	**HC**	**TE**	**UI**
start	0.127	0.530	0.052	0.030	0.261
TD	0.017	***0.414***	0.103	0.172	0.275
BU	0.016	**0.518**	0.069	0.153	0.216
HC	0.011	0.156	**0.218**	0.117	0.413
TE	0.003	0.117	0.075	**0.482**	0.270
UI	0.007	0.094	0.169	0.135	**0.397**

Tabelle 2: Transitionsmatrix zwischen Vorgehensmustern (Transition von Reihe zu Spalte) – TD: top down, BU: bottom up, HC: hill climbing, TE: trial and error, UI: unidentifiziert

4 Methode

4.1 Einführung

Die Analyse und manuelle Labelung der Daten bestätigt die Existenz der vier Strategien *top down*, *bottom up*, *hill climbing* und *trial and error*, die durch LSI-Sequenzen, in denen die Kategorien mit den Ziffern 0 bis 7 kodiert werden, dargestellt werden können. Um auch solche LSI-Sequenzen weiter verarbeiten zu können, die keiner dieser Strategien zugeordnet werden können, wird ein „Ausschuss-Muster“ eingeführt. Die *automatisierte* Identifizierung der von den Lernenden eingesetzten Problemlösestrategien in den LSI-Chronologien besitzt viele Gemeinsamkeiten zu im Bereich der Mustererkennung wohl bekannten Problemen – insbesondere zur automatischen Spracherkennung. Dort muss der Computer die Laute des Sprechenden erkennen, hier sind dies die von den Lernenden eingesetzten Strategien. In der Spracherkennung stellen Lautäußerungen die Eingangsdaten dar, hier werden die beobachteten LSI-Sequenzen verwendet. Ähnlich zum oben erwähnten *unidentifiziert*-Muster setzt die Spracherkennung eine Klasse „Geräusch“ ein.

4.2 Einsatz verborgener Markow-Modelle

Im Gegensatz zum starren Mustervergleich berücksichtigt ein erfolgreiches Modell auch Variationen in beobachteten Sequenzen. Dies ist in der Spracherkennung z. B. beim Sprecherwechsel notwendig; bei der Identifizierung von Problemlösestrategien können ähnliche, aber trotzdem verschiedene LSI-Sequenzen der selben Strategie zugeordnet sein. Um dies zu bewältigen, hat sich in der Mustererkennung die Verwendung von Automaten, die ihre Zustände nach einer Wahrscheinlichkeitsverteilung ändern, bewährt. Die weiteste Verbreitung hierbei haben verborgene Markow-Modelle (hidden Markov models – HMM) [Ch67], mit denen zeitliche statistische Abhängigkeiten modelliert werden. Im Gegensatz zum regulären Automaten, der für jedes Eingabesymbol festgelegte Transitionen besitzt, sind die Übergänge im HMM abhängig von der *Übergangswahrscheinlichkeit* a_{ij} von Zustand i zu Zustand j und den *Ausgabewahrscheinlichkeiten* $b_j(\mathrm{O}_t)$ dafür, dass das Symbol O_t im Zustand j ausgegeben wird (Abb. 11). Zusätzlich kann das Modell *Startwahrscheinlichkeiten* statt eines definierten Startzustandes besitzen. Nach Vorgabe der Parameter und einer Beobachtungssequenz wird die Wahrscheinlichkeit berechnet, dass die

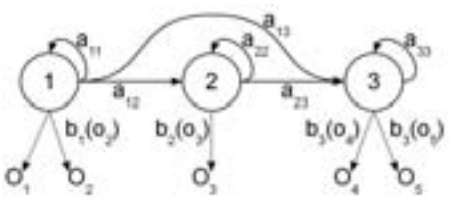

Abbildung 11: Links-Rechts-HMM (Übergangs- (a) und Ausgabewahrscheinlichkeiten ($b(\cdot)$)

Beobachtung gerade durch dieses Modell *erzeugt* wurde. Das *Training* der Modelle, insbesondere die Abschätzung der Übergangs- und Ausgabewahrscheinlichkeiten für ein bestimmtes Muster, wird mit Hilfe des iterativen Baum-Welch-Algorithmus [Ba70] durchgeführt. Für vorgegebene Trainingsfälle (bestimmten Problemlösestrategien zugeordnete LSI-Sequenzen) werden die Parameterwerte schrittweise verbessert, so dass die Wahrscheinlichkeit gesteigert wird, dass das Model genau diesen Fall generiert. Ähnlich wie es bei der Spracherkennung für jedes Wort und „Geräusch“ ein Modell gibt, wird hier für das „unidentifiziert“-Muster sowie alle vier Vorgehensmuster je ein Modell benötigt. Der Findungsprozess einer Folge von HMM in einer Beobachtungssequenz wird *Dekodierung* genannt. Zuerst wird nach der *HMM-Folge* gesucht, die am wahrscheinlichsten zur gesamten LSI-Sequenz passt. Dieses Problem wird mit dem auf dynamischer Programmierung basierenden Viterbi-Algorithmus [Vi67] gelöst, der aus einer Beobachtungssequenz und gegebenen HMM die wahrscheinlichste *Zustandsfolge* errechnet. Zur Berücksichtigung der zeitlichen Struktur der Problemlösestrategien werden hier nur sogenannte reine Links-Rechts-Modelle (d. h. $a_{ij} = 0$ für $i > j$ innerhalb der Strategiemodelle (Abb. 11)) zugelassen. Zur Dekodierung werden die fünf Modelle zu einem großen HMM kombiniert, wobei hier zusätzliche Übergangswahrscheinlichkeiten am Beginn und Ende jedes Vorgehensmodells eingefügt werden, um mittels der Eingangswahrscheinlichkeiten zu ermöglichen, dass die LSI-Folge mit jedem der Vorgehensmuster beginnen kann. Alle Wahrscheinlichkeiten wurden aus den in Tabelle 2 aufgeführten relativen Häufigkeiten als Initialwerten automatisch errechnet. Für die LSI-Sequenzen errechnet der Viterbi-Algorithmus unter Verwendung des kombinierten HMM diejenige Sequenz von Vorgehensmustern, die am wahrscheinlichsten die gegebene LSI-Sequenz erzeugt. Diese Folgen von Vorgehensmustern lassen die von den Lernenden eingesetzte Problemlösestrategie erkennen.

4.3 Identifizierungsmodul

Die gesamte Softwarearchitektur wird in Abbildung 12 dargestellt. Die Programmierumgebung Automaten-Kara wird erweitert durch das TrackingKara-Modul, welches alle Interaktionen des Hauptprogramms unter Ausnutzung des Beobachtermusters protokolliert

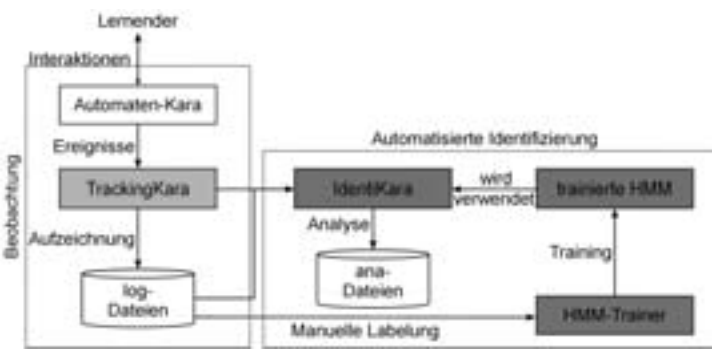

Abbildung 12: Softwarearchitektur einschließlich Automaten-Kara, TrackingKara und IdentiKara

und sie an das ebenso integrierte IdentiKara-Modul weiterleitet. Dieses verwendet die Daten einmalig zum Training der HMM und während des anschließenden Einsatzes zur Bestimmung der aktuellen (und bisher eingesetzten) Problemlösestrategie(n).

5 Ergebnisse und Schlussfolgerungen

Die eingesetzten Mustererkennungsmethoden erlauben die automatisierte, prozessbegleitende Identifizierung der Problemlösestrategien von Lernenden. Nicht nur individuelle Vorgehensmuster sondern auch deren Ausprägungen (LSI-Sequenzen), die typisch sind für die jeweils eingesetzten Problemlösestrategien, können identifiziert werden (Kap. 3.2). Die Wiederholung des *top down*-Musters in nur 1.7% aller Fälle (Tab. 2) erklärt sich durch einen Vergleich der Musterstrukturen. Alle anderen Muster beschreiben einzelne Schritte des Lösungsprozesses, von denen bis zur Gesamtlösung jeweils mehrere benötigt werden. Die *top down*-Strategie dagegen muss dazu typischer Weise nur einmal eingesetzt werden. Die hohe Wahrscheinlichkeit (41.4%) für den Wechsel von *top down* zu *bottom up* bestätigt die Tatsache, dass diese Strategie selbst von Programmierexperten nur sehr schwer konsequent beibehalten werden kann [Ho90]. Ein weiteres Resultat ist, dass die meisten der untersuchten Lernenden im Alter von 12 bis 13 Jahren die *bottom up*-Strategie bevorzugen. Zur Erstellung von individualisierten Systemrückmeldungen wird außer den automatisiert identifizierten Problemlösestrategien auch die jeweilige Lösungsqualität berücksichtigt, die mit einem in die Untersuchungssoftware eingebetteten Assessment-Modul bestimmt werden kann. Wie in [Dr04] dargestellt, kann durch die Gestaltung von Rückmeldungen als geeignete Kombination von personen-, verhaltens- und leistungsbezogenem Feedback die Motivation der Lernenden gefördert werden. Wird z. B. bei einem Lernenden während der Lösung zu „Kara und die Blätter“ die *bottom up*-Strategie identifiziert und es tritt bei einem Programmtestlauf nach der Erstellung des vorletzten Teilzweiges ein Fehler auf (angenommen: Vertauschung der Befehle „Schritt“ und „Blatt aufheben“), so könnte eine Fehlermeldung lauten: „Du hast Dein Programm gut strukturiert. Es fehlt Dir nur noch ein Teilzweig. Im aktuellen Zweig versucht Kara ein Blatt von einem Feld nehmen, auf dem keines liegt.“ Durch die Berücksichtigung der Vorgehensweise des Lernenden liegt der Fokus des Feedback dort, wo auch die Aufmerksamkeit des Lernenden gebündelt wird, er wird individuell angesprochen, motiviert und erhält Hinweise zur Lösungsfortsetzung. Der letzte Rückmeldungsteil ist eine Umformulierung der bisherigen technischen Meldung.
Für alle Kombinationen von erreichter Qualität und eingesetzter Vorgehensweise wird im hier vorgestellten Fall je eine individualisierte Meldung bereitgestellt, die die Lernenden nicht nur im Fall eines Fehlers, sondern auch bei Bedarf auf Anforderung, erhalten können. Durch die Anpassung an die Vorgehensweise der Lernenden, spricht sie sie besser an als rein technische Meldungen und kann ihre Motivation halten oder gar steigern. Für die Anwendung mit anderen Programmierumgebungen (z. B. auch aus dem Bereich der imperativen Programmierung) muss die Auswahl der relevanten LSI (Kap. 2.3) adaptiert werden. Eine vielversprechende Möglichkeit stellt hierzu die Verwendung der algorithmischen Kontrollstrukturen *Sequenz*, *Wiederholung*, *bedingte Verzweigung* dar. Zusätzlich werden die Programmausführungen sowie deren Abbruch und ggf. Systemfehlermeldungen protokolliert. Die Kategorien bleiben ebenso unverändert wie die HMM und insbesondere die

eingesetzten Mustererkennungsalgorithmen. Lediglich müssen die Modelle einmalig neu trainiert (Kap. 4) und für das automatische Assessment die Testfälle neu gestaltet werden.

6 Ausblick

In den nächsten Schritten werden die individualisierten Systemrückmeldungen (Kap. 5) kreiert und in die Software integriert. Die Programmierumgebung wird in diesem Zug so erweitert, dass Systemmeldungen von den Lernenden auch auf Anforderung erhalten werden können. Des Weiteren wird das Identifizierungsmodul angepasst an Programmierumgebungen, bei denen Programmbausteine per drag-and-drop von den Lernenden zu einem Gesamtprogramm zusammengestellt werden können (z. B. Scratch – [Ma04]).

Literatur

[Ba70] Baum, L. et al.: A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. Ann. Math. Statist., 41:164-171, 1970.

[Be98] Ben-Ari, M.: Constructivism in computer science education. SIGCSE Bull. 30, 1 (Mar. 1998), 257-261, 1998.

[CG85] Chi, M. T. H.; Glaser, R.: Problem solving ability. In Human Abilities: An Information-Processing Approach. W. H. Freeman & Co, San Francisco, CA. 227-257, 1985.

[Ch67] Chung, K.: Markov Chains with Stationary Transition Probabilities. Springer, Berlin, 1967.

[Dr04] Dresel, M.: Motivationsförderung im schulischen Kontext. Diss., München, 2000 (Münchener Universitätsschriften) – Hogrefe-Verlag für Psychologie, Göttingen, 2004.

[FS88] Felder, R. M.; Silverman, L. K.: Learning styles and teaching styles in engineering education. Engineering Education, 78(7), 674-681, 1988.

[Ho90] Hoc, J.-M.: Psychology of programming. Computers and people series. Academic, London, 1990.

[Ki09] Kiesmüller, U.: Diagnosing Learners' Problem Solving Strategies Using Learning Environments with Algorithmic Problems in Secondary Education. In: Transactions of Computing Education, 9(3), ACM-Press, New York, NY, USA, 1-26, 2009.

[Ma04] Maloney, J. et al.: Scratch: A Sneak Preview. In Second International Conference on Creating, Connecting and Collaborating through Computing. (Keihanna-Plaza, Kyoto, Japan, January 29-30, 2004) C5'04. IEEE Computer Society, Los Alamitos, CA, 104-109, 2004.

[Ma92] Mayer, R. E.: Thinking, problem solving, cognition (2^{nd} edition). W. H. Freeman and Company, New York, NY, 1992.

[NS72] Newell, A.; Simon, H. A.: Human Problem Solving. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Re03] Reichert, R.: Theory of Computation as a Vehicle for Teaching Fundamental Concepts of Computer Science. Doctoral Thesis. No. 15035. ETH Zurich, 2003.

[Ro07] Romeike, R.: Kreativität im Informatikunterricht. Diss., Universität Potsdam, 2008.

[Sh07] Shneiderman, B.: Creativity Support Tools: Accelerating Discovery and Innovation. Commun. ACM, 50(12):20-32, 2007.

[Vi67] Viterbi, A.: Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. IEEE Transactions on Information Theory, 13:260-269, 1967.