Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

De ontwikkeling van een intelligente programma-ontleder als onderdeel van een intelligent onderwijssysteem

Koen Bertels
UFSIA
Prinsstraat 13
2000 Antwerpen
België

Philip Vanneste
KULAK
Universitaire Campus
8500 Kortrijk
België

Samenvatting

In deze bijdrage bepreken we kort de verschillende manieren waarop men met een computer kan omgaan in het onderwijs. Vervolgens behandelen we meer specifiek intelligente onderwijssystemen (I.O.S.) en hun impact op het leer- en onderwijsproces. Na een voorbeeld van een I.O.S. uit het domein van programmeren, bespreken we het eigen onderzoek dat zich ook in dat domein situeert. Zowel de structuur van programmeerkennis als de oorzaken van studentenfouten worden beschreven.

1 Inleiding : de computer in het onderwijs

In het onderwijs wordt momenteel op verschillende manieren gebruik gemaakt van de computer. Er is onderwijs *over* de computer, onderwijs *met behulp van* de computer en onderwijs *door middel van* de computer.

Bij het onderwijs over de computer is de informatica het lesonderwerp. De computer kan ook gebruikt worden als hulpmiddel (onderwijs met behulp van de computer). Het gaat hier om het gebruiken van programmatuur die niet direct aan een bepaalde leerinhoud gebonden is. We denken hierbij aan het gebruik van de computer als controle-, sturings- of meetinstrument bij het uitvoeren van experimenten, of aan het gebruik van pakketten voor gegevensbeheer, tekstverwerking, statistische analyse,... Tenslotte is er onderwijs door middel van de computer. In dat geval is de leerinhoud wel vast. De computer kan drie taken op zich nemen : hij kan de student onderwijzen (nieuwe kennis bijbrengen), toetsen (kennis testen) en begeleiden (sturen doorheen de verschillende eenheden van een cursus). Indien de computer de drie taken op zich neemt, spreken we van computer ondersteund onderwijs. Bij computer beheerd onderwijs worden alleen de laatste twee taken door de computer verzorgd. Voor het opdoen van kennis wordt de student verwezen naar andere media.

Het systeem dat in deze paper besproken wordt, situeert zich bij het onderwijs over en door middel van de computer : de student leert programmeren en wordt hierbij getest door de computer. Indien hij fouten maakt, dan zouden deze door een tutor-module geremedieerd kunnen worden.

2 Intelligente Onderwijssystemen

2.1 Inleiding

Bij traditionele systemen voor computer-ondersteund onderwijs worden alle beslissingen in verband met lesverloop, inhoud, correctheid van oefeningen,... op voorhand door de ontwerper genomen, op basis van de op voorhand voorziene mogelijkheden. Artificiële intelligentie biedt de mogelijkheid om een onderwijssysteem van de nodige kennis te voorzien om zelf die beslissingen te nemen. Zo komen 'intelligente' onderwijssystemen (IOS) tot stand. Deze kunnen zelf beslissingen nemen, op het moment dat

dit nodig blijkt. Hiervoor maakt het IOS gebruik van de aanwezige kennis over de situatie.

2.2 Kennis in een intelligent onderwijssysteem

Bij een intelligent onderwijssysteem wordt kennis gebruikt over

- didactiek
- het lesonderwerp
- het stellen van een diagnose

2.3 Didactische kennis

Didactische kennis wordt gebruikt om het verloop van de les te sturen. Hierbij kan geprobeerd worden om de kennis van de student uit te breiden, om aan bepaalde misverstanden te verhelpen, om de student te testen, ... Deze beslissingen worden genomen op basis van informatie uit het student-model en kennis van het lesonderwerp.

2.4 Kennis over het lesonderwerp

Deze kennis is noodzakelijk om uitleg over het lesonderwerp te genereren (domeinkennis). Daarnaast kan deze kennis ook gebruikt worden om problemen te genereren die dan aan de student kunnen worden voorgelegd. Sommige IOS bevatten ook een expert-systeem dat in is staat om zelf de gestelde problemen op te lossen. Deze oplossing kan voor twee doeleinden gebruikt worden : enerzijds om te vergelijken met de oplossing van de student, en anderzijds om de oplossingsmethode aan de student te demonstreren. In het laatste geval is het noodzakelijk dat het expert-systeem alle stappen ook kan uitleggen, en dat problemen opgelost worden op een manier die overeenkomt met de oplossingmethode van menselijke experts. Zo'n expertsysteem wordt een glass-box expert genoemd. De domeinkennis wordt dus gebruikt als bron van kennis en van oefeningen, en dient als referentiepunt om antwoorden van de student te beoordelen.

2.5 Kennis over het stellen van een correcte diagnose

Het IOS moet in staat zijn om oefeningen die door de student worden opgelost te ontleden en te beoordelen. Er moet gedetecteerd kunnen worden of de oplossing al dan niet verkeerd is, en welke misverstanden eventueel deze fouten veroorzaken.
De kennis van de student kan gezien worden als zijnde een deelverzameling van de kennis van een ideale student. In dat geval spreken we van een *overlay*-model. Bij een *buggy*-model is er ook mogelijkheid om expliciet misverstanden van de student voor te stellen : de student kan bepaalde vaardigheden op een verkeerde manier gebruiken. Met de informatie uit de diagnose kan het model van de student opgebouwd of aangevuld worden: bij een correcte oplossing wordt eventueel genoteerd dat bepaalde nieuwe vaardigheden verworven zijn, of dat aan een misverstand verholpen is. Bij een foutieve oplossing kan een misverstand aan het model toegevoegd worden. Er dient opgemerkt te worden dat de kennis die nodig is voor het correct diagnoseren van een oefening sterk afhankelijk is van het lesonderwerp, en dus nauw kan samenhangen met de domeinkennis.

2.6 Structuur van een intelligent onderwijssysteem

Een intelligent onderwijssysteem kan worden opgesplitst in een aantal modules, die elk een specifieke taak te vervullen hebben:

- de expert-module
- de diagnoserende module
- de tutor module (bevat de didactische kennis)
- de interface

3 Impact van intelligente onderwijssystemen

Computers bieden enorme mogelijkheden aan het onderwijs. Verschillende factoren spelen hierin een rol. Vooreerst zijn kinderen enorm geboeid door computers. Het spelelement dat onmiskenbaar verbonden is met computers is daar zeker niet vreemd aan. Ook mogelijkheden voor het

uitwerken van een dynamische, sterk grafische interface dragen bij tot de aantrekkelijkheid van de computer als onderwijsmedium. Als in de toekomst deze interfaces ook nog intelligent gemaakt kunnen worden, bijvoorbeeld door het inbouwen van natuurlijke taal-mogelijkheden, dan zal deze aantrekkelijkheid nog verhogen.

De invoering van intelligente onderwijssystemen kan grote veranderingen teweegbrengen in de manier waarop het onderwijs georganiseerd wordt. Een zeer recente studie op dat gebied (zie Schofield 1990) heeft een aantal interessante zaken aan het licht gebracht. De studie beschrijft een experiment met een intelligent onderwijssysteem voor het aanleren en inoefenen van bewijsvoeringen uit de meetkunde (GPTutor). Er werden wijzigingen vastgesteld in het gedrag van zowel de leerkracht als de leerlingen. We geven een kort overzicht van de conclusies:

1) Wijzigingen in het gedrag van de leerkracht:
   a) De aandacht van de leerkracht wordt meer gelijkmatig verdeeld over goede en minder goede leerlingen, daar waar er in een gewone klassituatie meer aandacht gaat naar de goede leerlingen.
   b) De rol van de leerkracht wijzigt van autoritair expert naar medewerker. De leerkracht staat niet meer vooraan, maar kan studenten individueel begeleiden. De overdracht van kennis gebeurt vooral via het intelligent onderwijssysteem.
   c) Bij het kwoteren schenkt de leerkracht meer aandacht aan de geleverde inspanning. Niet alleen het gepresteerde resultaat is belangrijk. Wellicht speelt het feit dat de leerkracht de leerlingen meer individueel begeleidt hierin ook een rol: de leerkracht heeft een beter zicht op de geleverde inspanningen van elke individuele student.
2) Wijzigingen in het gedrag van de leerlingen
   Hier valt vooral op dat de studenten meer inspanning leveren en meer betrokken zijn bij hun werk. Dit wordt als volgt verklaard:
   a) Er is een toegenomen competitie in de klas. Leerlingen zijn niet meer gebonden aan de oefeningen

die klassikaal behandeld worden, maar kunnen op eigen tempo de aanwezige oefeningen oplossen. Hierdoor kunnen leerlingen een grotere voorsprong verkrijgen, zodat de motivatie stijgt.

b) De leerlingen beleven meer plezier aan hun werk. Volgens de studie is dit een gevolg van het feit dat de leerlingen zelfstandig kunnen werken, onafhankelijk van volwassenen. Daarbij komt dat leerlingen de computer als een spel-machine zien. Ze kunnen ook gemakkelijker hun emoties afreageren op een computer: een computer reageert niet als je erop scheldt.

c) De leerlingen zijn minder bang om iets uit te proberen. Alle fouten die ze maken blijven privé: ze worden niet opgemerkt door de rest van de klas.

d) Opmerkelijk is dat blijkt dat het veranderd kwoteringsgedrag van de leerkracht weinig of niets bijdraagt tot de verhoogde motivatie van de leerlingen.

4 Eigen onderzoek

4.1 Doelgroep

De doelgroep zijn niet-informatica studenten die een inleidende cursus programmeren volgen. Het kan dus zowel gaan om 1ste jaarsstudenten als om leerlingen in de laatste cyclus van het middelbaar onderwijs.

4.2 Doelstelling en kennisdomein

Kunnen programmeren veronderstelt het vermogen tot logisch en algoritmisch denken eerder dan syntactische kennis over variabelen en controlestructuren van een bepaalde programmeertaal.
Cognitieve studies over programmeren wijzen uit dat een programmeur niet denkt in termen van de basisconcepten van een programmeertaal (zoals een do while lus of een read instructie), maar in algemene, conceptueel hogere

schema's (zie Brooks 1977, Soloway 1983, Soloway 1986, Bonar 1985).
Ze vormen de eigenlijke bouwstenen van een programma. Deze hogere schema's of clichés zijn primitieve standaardbewerkingen die gecombineerd met elkaar een programma opleveren. Zulke clichés kunnen ook worden geparametriseerd in functie van een bepaalde toepassing (zie Waters 1986). In de praktijk betekent dit dat de programmeur tijdens het schrijven vaak terugvalt op programmaroutines die hij in het verleden aanleerde of zelf ontwikkelde. Voorbeelden van zulke clichés zijn het sequentieel zoeken in een bestand, het genereren van een rapport, het lineair zoeken in een array, e.d.
Ook bij het debuggen van bestaande programma's maakt de programmeur gebruik van clichés (zie Gould 1974). Bij het debuggen van een bestaand algoritme tracht hij immers de in het programma gebruikte structuren te vergelijken met door hem gekende clichés (zie sheil81). Op deze wijze bouwt hij een interpretatie op van het programma.

Indien we leerlingen basiskennis m.b.t. het programmeren wensen bij te brengen moet voldoende aandacht worden geschonken aan het aanleren en gebruiken van deze clichés.
Dit laatste kan als voornaamste doelstelling (en kennisdomein) gelden voor ons intelligent onderwijssysteem. We wensen dus niet dat de leerling in staat is bv. een facturatieprogramma te schrijven maar wel dat hij alle clichés beheerst die daarin kunnen worden gebruikt. Het spreekt vanzelf dat in een volledig uitgebouwd informatica curriculum het ontwerpen van grote programma's niet mag ontbreken.

4.3 Context en beperkingen van het onderwijssysteem

In de context van ons intelligent onderwijssysteem beperken we de programmeerkennis tot zuiver numerieke verwerking. Er wordt dus enkel met numerieke variabelen gewerkt waarbij we ook die categorie variabelen beperken tot de natuurlijke getallen. Ook worden het creëren en gebruiken van bestanden achterwege gelaten. Het gebruik

van procedures en functies evenals recursie worden niet expliciet behandeld.
De programmeertaal die door de studenten zal worden gebruikt is ontworpen i.f.v. de hierboven opgesomde kenmerken en beperkingen. Qua syntax is de taal een mengeling van Pascal en Basic.

In een eerste fase wordt enkel de diagnostische module uitgewerkt en veronderstellen we zowel de expert- als de pedagogische module als gegeven. In een volledig uitgewerkt systeem zou de expertmodule in staat moeten zijn om op basis van een opgave in quasi-natuurlijke taal een oplossing te genereren.

4.4 Input van het systeem

Eén van de inputs voor de diagnostische module de oplossing van de student.
Aangezien contextuele informatie echter ook nodig is om alle foutieve en correcte varianten van een algoritme te vinden, zal ook een leraaroplossing mee als input aan het systeem gegeven worden (zie Soloway 1984a). Uit deze leraaroplossing kan alle nodige contextuele informatie gedistilleerd worden.

4.5 Fasering van de analyse

Het analyseproces bestaat uit drie grote stappen :
stap 1 : De algemene programmeerkennis wordt gebruikt om het leraarprogramma te interpreteren. Het systeem zoekt naar gekende clichés waardoor het in staat is te detecteren wat het programma juist moet doen, welke variabelen gebruikt worden en welke rol ze spelen.
stap 2 : In een tweede fase zal het studentenprogramma worden bekeken. Dit interpretatieproces wordt gestuurd vanuit de specifieke kennis over het leraarprogramma die in de vorige stap werd opgebouwd. Het systeem zal trachten tot eenzelfde interpretatie te komen. Door te 'redeneren' in termen van clichés kan het systeem abstractie maken van syntactische en structurele varianten in de implementatie.

stap 3 : Indien het systeem niet tot eenzelfde interpretatie van het studentalgoritme kan komen, wordt een lijst opgesteld van de verschillen en gelijkenissen. De verdere analyse zal er dan op gericht zijn de lijst van verschillen te minimaliseren zonder de reeds bestaande gelijkenissen te verminderen. Hierbij zal in eerste instantie getracht worden om via transformaties toch verwante clichés te vinden. Indien ook deze strategie faalt, zal een specifiek zoekproces worden opgestart dat tracht fouten te localiseren om op basis hiervan ontbrekende, overbodige of verkeerde delen in het programma te wijzigen of als nieuw uitgangspunt te nemen in het interpretatieproces.

4.6 Empirische achtergrond

Het eigen empirisch onderzoek baseert zich in grote mate op studies die uitgevoerd werden teneinde programmeerkennis te beschrijven evenals te achterhalen wat de oorzaak is van programmeerfouten. In de twee volgende paragrafen beschrijven we eerst de structuur van programmeerkennis zoals ongeveer gelijklopend beschreven in (zie Soloway 1986, Rich 1986, Jonckers 1986). Deze beschrijving geschiedt in functie van de kennisbank die wordt geïmplementeerd. Daarna bespreken we mogelijke oorzaken van programmeerfouten.

4.7 Structuur van programmeerkennis

Indien we wensen dat ons diagnostisch systeem een groot aantal oefeningen kan analyseren moet het beschikken over zowel algemene programmeerkennis als strikt probleemgebonden -contextuele- informatie. We gebruiken de begrippen van Waters en Joncker (zie Waters 1986, Jonckers 1986).
In de algemene programmeerkennisbank moet informatie staan over zowel de syntactische kenmerken van de gebruikte programmeertaal -concrete programmeerconcepten- als het gebruik van standaardroutines -abstracte programmeerconcepten.

Het betreft hier het statische deel van de kennisbank. Contextuele informatie wordt niet expliciet voorgesteld in de kennisbank. Door de algemene programmeerkennis te gebruiken moet het systeem in staat zijn om de strikt probleemgebonden informatie uit het algoritme te distilleren. Hier wordt de contextuele kennisbank dynamisch (runtime) opgebouwd. Belangrijk bij het formuleren van de kennisbank is dat alle assumpties of voorkennis van dewelke een menselijke programmeur vertrekt, voorgesteld is in de kennisbank (zie Barstow 1986).

4.8 Oorzaken van de studentenfouten

Bonar en Soloway onderzochten welke denkprocessen fouten in een algoritme veroorzaakten (zie Bonar 1985).
Als algemene benaming voor het maken van fouten gebruiken Bonar en Soloway patchwork of lapwerk. Telkens de student zich in een probleemsituatie bevindt, gebruikt hij dit lapwerk om toch maar een oplossing te kunnen formuleren. Uit hun aard zijn zulke oplossingen echter foutief.
Zij kwamen tot volgende foutgeneratoren :
a) programmeertaal als natuurlijke taal gebruiken:
Hier gaat de leerling ervan uit dat programmeertaalstructuren eenzelfde betekenis hebben als hun natuurlijke taaltegenhanger.
b) programmeertaal als natuurlijke taal interpreteren:
Een fout die vaak terugkomt is dat studenten bepaalde commando's interpreteren in hun natuurlijke taalbetekenis. Vooral de read of lees en de write of schrijf commando's worden vaak op die manier geïnterpreteerd.
c) variabele vervult meerdere rollen: Ook kan het voorkomen dat de student confligerende functies toekent aan 1 variabele. Een voorbeeld daarvan is dat de somvariabele en de variabele die de op te tellen waarde bevat door één en dezelfde variabele worden aangeduid.

5 Afsluiting : toekomst van intelligente onderwijssystemen

Programmeren is een zinvolle vaardigheid om aan te leren, indien het de bedoeling is het logisch redeneervermogen van leerlingen aan te scherpen.
Het kan zeker niet worden beschouwd als een voorbereiding op latere studies of professionele bezigheden. Zeer weinig mensen zullen later immers toepassingen moeten programmeren. In dit perspectief kan een I.O.S. voor het leren programmeren een zinnige bijdrage zijn aan het onderwijs.
Het inschakelen van I.O.S. verhoogt de persoonlijke begeleiding die de leerling kan krijgen. De leraar heeft immers meer tijd om in te spelen op de individuele noden en verlangens van elke leerling.
Schank (1984) wijst er echter terecht op dat er meer slechte dan goede manieren zijn om computers in het onderwijs in te schakelen. Het mag zeker niet uitmonden in het maken van electronische versies van het klassieke oefenboek.
Tenslotte willen we erop wijzen dat ook niet-intelligente onderwijssystemen pedagogisch minstens even bruikbaar kunnen zijn. Veel vaardigheden worden immers nog het best getraind via het intensief maken van oefeningen.

Gebruikte literatuur :

Barstow D. (1986) An Experiment in Knowledge-Based Automatic Programming; in (Rich 1986a) pp. 133-156.

Bertels K. (1990); An Experiment on Novice Programming Behaviour; UFSIA-rapport 1990.

Bonar J. & Soloway E. (1985); Preprogramming Knowledge: a Major Source of Misconceptions in Novice Programmers; Human Computer Interaction, vol. 1, pp. 133-161.

Brooks R.(1977); Towards a Theory of the Cognitive Processes in Computer Programming; Int. J. Man-Machine Studies, nr. 9, pp. 737-751.

Gould J. D. & Drongowski P.(1974); An Exploratory Study of Computer Program Debugging; Human Factors, vol. 16, nr. 3, pp. 258-277.

Johnson W. L. (1986); Intention-Based Diagnosis of Novice Programming Errors; Morgan Kaufmann Publishers, 1986.

Johnson W. L. & Soloway E. (1987); PROUST: An Automatic Debugger for Pascal Programs; (Kearsley 1987), pp. 49-69.

Jonckers V. (1987); A Framework for Modeling Programming Knowledge; V.U.B. A.I.-Lab, Technical Report 87-1.

Kearsley G. (ed.) (1987); Artificial Intelligence and Instruction: Applications and Methods; Addison Wesley.

Rich C. & Waters R. C. (eds.) (1986); Readings in Artificial Intelligence and Software Engineering; Morgan Kaufmann Publishers Inc.

Rich C. (1986b); A Formal Representation for Plans in the Programmer's Apprentice; in (Rich 1986), pp. 491-506.

Schank, R. C. (1984); The Cognitive Computer; Addison-Wesley, 268 p.

Schofield J. W.; Artificial Intelligence in the Classroom: The Impact of a Computer-Based Tutor on Teachers and Students; in press, Social Science Computer Review.

Sheil B. A. (1981); The Psychological Study of Programming; Computing Surveys, vol. 13, nr. 1, pp. 101-120.

Soloway E., Bonar J. & Ehrlich K. (1983); Cognitive Strategies and Looping Constructs: An Empirical Study; C.A.C.M., Vol. 26, Nr. 11, pp. 853-860.

Soloway E. & Johnson W. L. (1984a); Remembrance of Blunders Past: a Retrospective on the Development of PROUST; Proc. of the 6th cogn. science society conf., p.57.

Soloway E. & Ehrlich K. (1984b); Empirical Studies of Programming Knowledge; I.E.E.E. trans. on software engineering, vol. 10, nr. 5, pp. 595-609.

Soloway E. & Johnson W. L. (1986); PROUST: Knowledge Based Program Understanding; in (Rich 1986a), pp. 443-451.

Vanneste Ph., Olivié H. & Dedecker B. (1990); Artificiële intelligentie en het gebruik van de computer in het onderwijs: nieuwe perspectieven; Rapport ITS-1, KULAK.

Waters R.; The Programmer's Apprentice: a Session with KBEmacs; (Rich 1986), pp.351-376.

Youngs E. A. (1974); Human Errors in Programming; Int. J. of Man-Machine Studies, nr. 6, pp. 361-376.