Der genetische Algorithmus - eine Implementierung in Prolog

Genetisches Lernen verwendet eine Menge von Genen, verbindet sie zu Lösungsansätzen für ein gegebenes Problem und bewertet diese Ansätze. Abhängig von der Leistung einer vorgeschlagenen Lösung werden die sie darstellenden Gene durch genetische Mechanismen wie Crossover, Selektion, Mutation etc. verändert. So ist eine Rückkopplung in einer sonst ungesteuerten Suche verwirklicht.
Die Qualität der Lösungsentstehung ist abhängig von der Größe des Pools, der verwendeten evolutionären Mechanismen, der Repräsentation der Gene u.A..
Für ein recht einfaches Beispiel der Planerzeugung soll ein solcher lernender Algorithmus implementiert und entsprechend seinen Parametern wie angedeutet modifiziert werden.

Voraussetzungen : Praktikum Machine Learning

Literatur :

/1/ Holland, J.H., (1975) Adaptation in Natural and Artificial Systems,
University of Michigan Press, Ann Arbor, Michigan.

/2/ Grefenstette, J.G. (1985) Proceedings of the first International Conference on Genetic Algorithms and Their Applications
Carnegie Mellon Univ. 1985

/3/ Spektrum der Wissenschaft 1/1986, A.K. Dewdeney S. 4-11

Bearbeiter : Dickmanns Dirk
Schmidhuber Jürgen
Winklhofer Andreas

Aufgabensteller : Prof. B. Radig

Betreuer : Dr. Werner Konrad

INHALTSVERZEICHNIS

1. Maschinelles Lernen

Maschinelles Lernen bezeichnet Methoden, die einen Rechner befähigen, Wissen nicht mehr nur per Eingabe durch einen Menschen zu erwerben, sondern auch durch geeignete Algorithmen aus gegebenen Daten zu extrahieren. Außerdem soll vorhandenes Wissen aktualisiert, erweitert und in eine bei den gestellten Problemen besser anwendbare Form gebracht werden. Einige schon untersuchte Lernmethoden sind folgende:

- Bei Induktivem Lernen werden aus Beispielen durch Induktion Regeln generiert. Dann kann bei weiteren Beispielen entschieden werden, welchen Regeln sie (etwa bei einem medizinischen Diagnosesystem: Symptome welchen Diagnosen) entsprechen oder nicht.
- Lernen durch Analogie verknüpft verschiedene Wissensgebiete. Durch Analogiebildung zwischen diesen versucht man Regeln und Wissen des einen Gebietes auf das andere zu übertragen.
- Das Zusammensetzen von Wissen zu komplexeren Strukturen wird mit Lernen durch Komposition erreicht.
- Genetisches Lernen versucht die Nachbildung der natürlichen Evolution. So wie sich die Arten immer besser den Bedingungen anpassen, die sie in ihrem Lebensraum vorfinden, sollen sich Ansätze zur Lösung eines Problems den an sie gestellten Anforderungen anpassen.

Der letztgenannte Prozess der Anpassung findet sich auch in der Psychologie, Wirtschaft, Regelungstechnik und Konstruktion; deshalb haben wir uns dazu entschieden, das Genetische Lernen zu implementieren und auf die "Konstruktion" von Programmen anzuwenden. Die Nachbildung der natürlichen Evolution, wenn auch nur sehr eingeschränkt, erscheint faszinierend: An der Natur ist zu erkennen, daß "Genetisches Lernen" sehr vielschichtig ist und erstaunliches zu Wege bringt. Es besteht daher ein Reiz in der Überraschung beim Betrachten der Ergebnisse eines Laufs, die manchmal nicht vorherzusehen sind. Für die anderen erwähnten Lernmethoden wurden zudem schon effiziente Algorithmen entwickelt.

2. Der Genetische Algorithmus

2.1. Anwendungsgebiet des Genetischen Algorithmus

Der Genetische Algorithmus kann besonders bei Aufgaben eingesetzt werden, für die eine ungefähre Lösung ausreichend ist, wie es manchmal in der Mustererkennung, bei Klassifikationen und dem Entwurf von Pipeline-Architekturen und Symbolen der Fall ist, da es keine "Ideallösung" gibt, sondern lediglich gute Kompromisse. Bei npc-Problemen ist die Ideallösung zwar bekannt, aber oft nicht in praktikabler Zeit zu errechnen, so daß für Näherungen auch hier bereits Genetische Algorithmen eingesetzt wurden /2 S.154 Goldberg S.160 Graefenstette/ /3/. In der Regel handelt es sich um komplexe Strukturen, die sich der Behandlung mit konventionellen Methoden entziehen. Üblicherweise werden die Aufgaben in einer Form gestellt, die es ermöglicht, die Lösung als endlichen Automaten oder als Liste von Parametern (als Eingabewerte für Funktionen bei mehrdimensionalen Optimierungsaufgaben) darzustellen. Diese Darstellungsformen sind jedoch nicht sehr mächtig, weshalb wir versuchten, mithilfe des Genetischen Algorithmus programmartige Näherungslösungen zu den als Beispielen gewählten Problemen zu erhalten.

2.2. Funktionsprinzip

Im Genetischen Algorithmus soll der Mechanismus, der in der Natur für die Weiterentwicklung der Lebewesen sorgt, auf Lösungsansätze (Pläne) zur gestellten Aufgabe angewendet werden. Dieser Mechanismus der Fortpflanzung ist in der Natur wohl auch evolutionär entstanden, diese Entwicklung ist jedoch mit der von in vorgegebener Richtung besser angepassten Arten nicht vergleichbar und daher kaum ausnutzbar (ungerichtete Suche). Deshalb besteht der Genetische Algorithmus eigentlich nur aus wiederholter Fortpflanzung und beginnt nicht im Chaos. Die dauernde Wechselwirkung zwischen den Individuen und dem Selektionsdruck, letzterer wird durch andere Individuen mitbestimmt, wird nicht realisiert, da sich die Individuen in eine bestimmte Richtung entwickeln sollen.
Es wird zunächst eine Menge von zufälligen Plänen erzeugt, die Anfangspopulation. Diese werden dann auf ihre Eignung zur Bewältigung des Problems durch kontrollierte Ausführung untersucht und entsprechend ihrer Leistung mit einer Bewertung versehen. Natürlich sind diese Programme oft fehlerhaft

oder unsinnig und nicht sehr leistungsfähig. Zur Verbesserung wird folgende Schleife durchlaufen:

i) Ein Plan wird aus der Population (dem Pool) ausgewählt und einer Veränderung unterworfen. Dazu dienen "genetische Operatoren", die ihre Entsprechung in der Natur in den Veränderungsmöglichkeiten der DNS-Sequenz bei der Fortpflanzung finden.

ii) Der neu entstandene Plan wird ausgeführt und bewertet (Lebenskampf).

iii) Anhand der Bewertung wird entschieden, ob der Plan verworfen wird, weil er zu schlecht ist, oder ob er einen anderen Plan aus der Population verdrängt. Dem entspricht in der Natur das Überleben der am besten angepassten Individuen.

Damit wird erreicht, daß die Pläne monoton (nicht streng monoton) besser werden und nach einiger Zeit einer von ihnen eine zufriedenstellende Lösung darstellen kann.

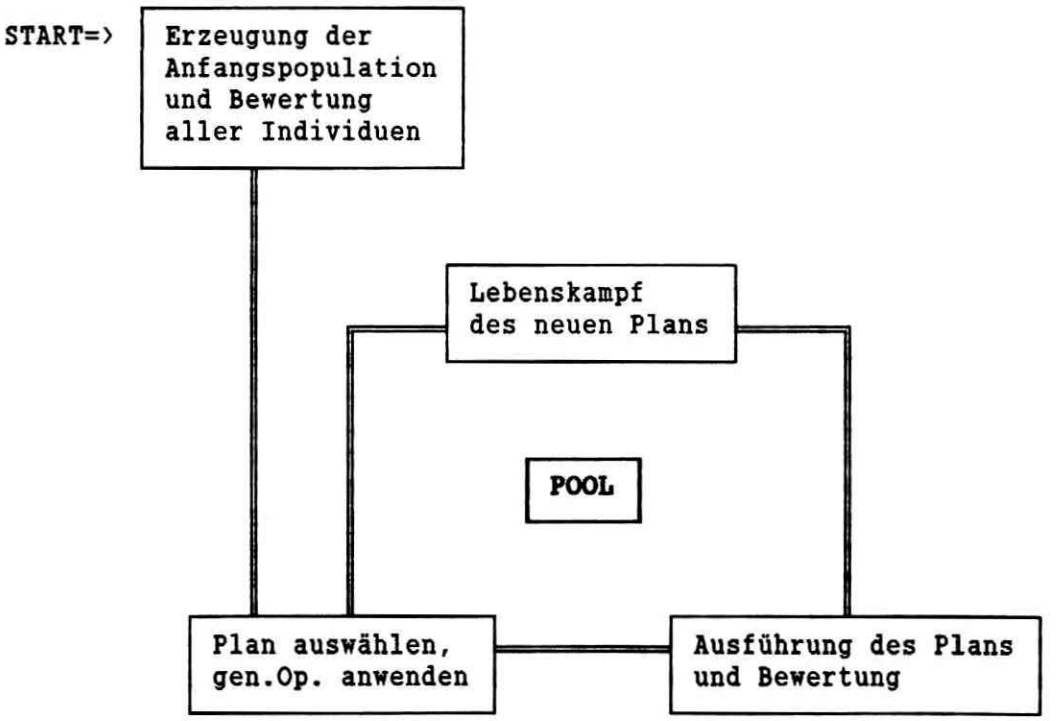

Bild 1. Schema des Genetischen Algorithmus

2.3. Genetische Operatoren

Die Pläne werden üblicherweise als String oder binärcodiert als Bitstring dargestellt. Sie sind die Eingabe für die genetischen Operatoren, die darauf kleinere und seltener auch tiefgreifende Änderungen durchführen. Die gebräuchlichen Operatoren, die im wesentlichen von der Natur übernommen wurden, sind:

a) Mutation,
b) Crossover,
c) Inversion,
d) Einfügen und Löschen.

zu a) MUTATION
ist ein Operator, der mit geringer Wahrscheinlichkeit angewandt wird. Er verändert ein zufällig ausgewähltes Element des Plans, indem es durch irgendein Primitives aus dem Alphabet ersetzt wird (Bild 2).

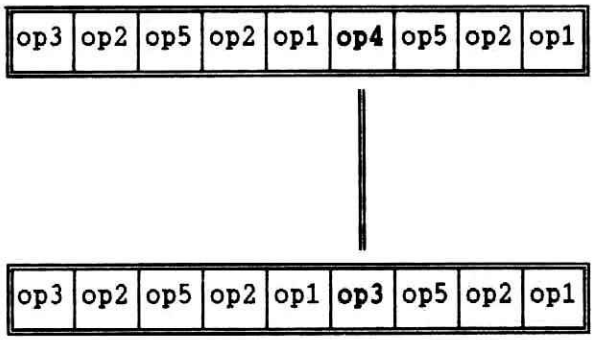

Bild 2. Mutation

zu b) CROSSOVER
wird in der Natur stets bei der Fortpflanzung durchgeführt. Aus zwei Individuen entsteht ein neues dadurch, daß der erste Plan als Basis genommen wird und ein zufällig ausgewälter Bereich seiner Stringdarstellung durch einen ebenso zufällig festgelegten Bereich des zweiten Plans (meist verwendet man hierfür den besten) ersetzt wird (Bild 3). Durch diese allgemeine Definition kann sich die Länge der Darstellung verändern, so dass bei entsprechender Berücksichtigung in der Bewertungsfunktion auch nach der Länge eines Plans optimiert werden kann. Somit wird nach möglichst kurzen Problemlösungen gesucht.

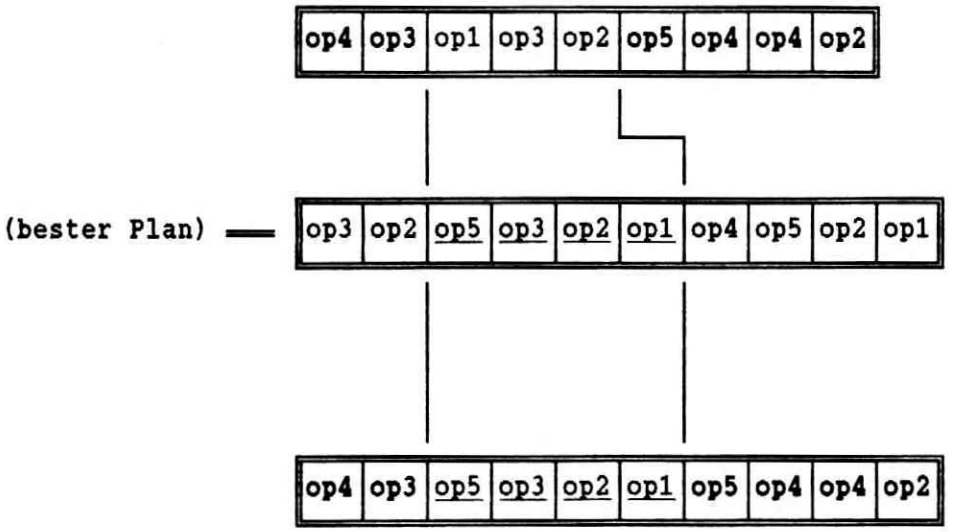

Bild 3. Crossover

zu c) INVERSION

ist ähnlich dem Crossover. Es handelt sich dabei jedoch um einen Operator, der nur einen Plan verwendet. Das einzusetzende Stück der Repräsentation ist der herausgenommene Teil rückwärts gelesen (Bild 4).

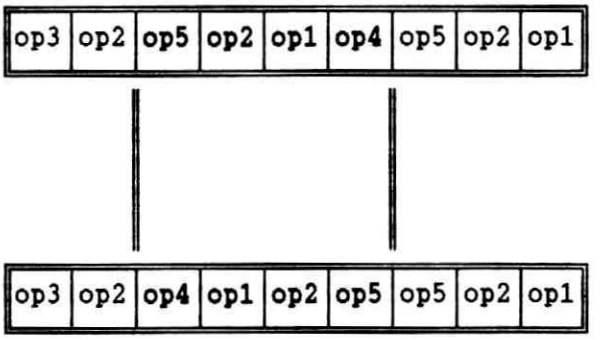

Bild 4. Inversion

zu d) EINFÜGEN und LÖSCHEN

werden mit ähnlicher Wahrscheinlichkeit wie die Mutation verwendet. An einer beliebigen Stelle des Plans wird eine zufällig gewählte Operation eingefügt bzw. eine Operation gelöscht.

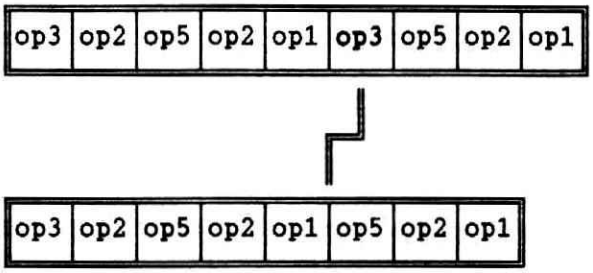

Bild 5. Löschen einer Operation

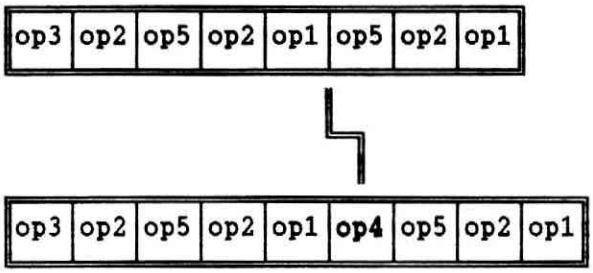

Bild 6. Einfügen einer Operation

Der Tatsache, daß sich in der natürlichen Evolution eine gut angepaßte Art stärker vermehrt und sich daher schneller noch weiter verbessern kann, wird auch im Genetischen Algorithmus realisiert, indem die Auswahlwahrscheinlichkeit des Individuums zur Anwendung eines genetischen Operators erhöht wird.

2.4. Auswirkungen von Variationen und Parametern

2.4.1. Einfluß der Repräsentation der Pläne

Da die genetischen Operatoren auf der Darstellung eines Individuums arbeiten, ist es nur anschaulich, dass die Codierung einen wesentlichen Einfluss auf die Effizienz des Genetischen Algorithmus hat. Wichtig ist es, eine Darstellung zu finden, bei der kleine Änderungen möglichst nicht allzuviel an der Leistung eines Individuums verändern. Dazu als Beispiel das Problem des Handlungsreisenden: Es ist NP-vollständig und wurde deshalb auch schon oft mit dem Genetischen Algorithmus zu lösen versucht. Die Aufgabenstellung ist folgende: Gegeben ist ein Graph mit ungerichteten, aber bewerteten Kanten. Gesucht ist ein geschlossener Weg im Graph, der mit möglichst geringen Kosten (Summe der Bewertungen der im Weg enthaltenen Kanten) alle

Knoten mindestens einmal erreicht. Würde man einfach die Reihenfolge der begangenen Kanten oder besuchten Knoten als Codierung verwenden, so würden die Pläne nach Anwendung eines genetischen Operators nicht notwendigerweise einen zulässigen geschlossenen Weg darstellen. Eine mögliche Repräsentation, die bei jedem Crossover zweier Pläne wieder einen geschlossenen Weg erzeugt, entsteht z.B. so /3/: Ein Weg durch fünf Knoten in der Reihenfolge a-c-e-d-b wird durch die Zahlenfolge 1-2-3-2-1 dargestellt. Dabei bezieht man sich auf eine Standardanordnung, hier a-b-c-d-e. Um nun eine Rundreise wiederzugeben, streicht man die Knoten der Reihenfolge des gegebenen Weges nach aus der Standardliste; a ist demnach der erste Knoten, c der zweite unter den verbleibenden, e der dritte, d der zweite und b schliesslich der erste. Allerdings führt eine geschickte Art der Darstellung nicht zwangsläufig zu einer befriedigenden Lösung des Problems. Ebenso muß darauf geachtet werden, daß die gesuchte Lösung in der gewählten Repräsentationssprache überhaupt ausdrückbar ist.

2.4.2. Einfluß der Parameter in der Bewertungsfunktion

Mithilfe der Bewertungsfunktion wird der Selektionsdruck erzeugt und daher auch festgelegt, in welche Richtung die Pläne verbessert werden: Je nachdem, welche Faktoren in welcher Gewichtung in die Bewertung eingehen, wird nach besonders schnellen, kurzen oder "gründlichen" Plänen gesucht. Wenn zum Beispiel die Länge der Pläne zu stark negativ in die Bewertungsfunktion eingeht, werden die Pläne immer kürzer, bis sie keine Operation mehr enthalten. Derartige Effekte werden bisher nur durch Probieren ausgeschaltet (siehe 4. Ergebnisse).
Die Funktion zur Bewertung ist im allgemeinen nicht monoton. Solche Funktionen bereiten fast allen Suchmethoden große Probleme; der Genetische Algorithmus unterscheidet sich von anderen aber dadurch, daß er viele der Problemstellen (Unstetigkeiten, lokale Maxima) "zufällig" überwindet. Zusätzlich wird beim Genetischen Algorithmus simultan an mehreren Stellen gesucht, da die Population eine Menge vieler Individuen ist. Dadurch läuft man weniger Gefahr, sich mit allen Individuen an lokalen Maxima der Bewertungsfunktion festzuhaken, als mit vielen anderen Suchmethoden.
Mit einer geschickten Wahl der Bewertungsfunktion läßt sich eine akzeptable Korrelation zur tatsächlichen Leistung erreichen, aber selbst dann gibt sie keinen Aufschluß darüber, wie groß und welcher Art weitere Änderungen an einem Plan sein müssen, um optimale Leistung zu erreichen.

2.4.3. Einfluß der Auswahlwahrscheinlichkeit der genetischen Operatoren

Bei der Festlegung der Auswahlwahrscheinlichkeiten der genetischen Operatoren müssen bestimmte Erfahrungstatsachen berücksichtigt werden. Die Mutation z.B. soll nur selten zum Einsatz kommen. Ein Erfahrungswert aus der Literatur ist: Auf ca. 10^3 bis 10^4 Anwendungen von Crossover kommt eine Mutation. Dies liegt an dem völlig zufälligen Charakter der Mutation, während bei Crossover im allgemeinen Teile eines schon besonders guten Plans zur Änderung eines anderen herangezogen werden. Durch etwas höhere Mutationsraten erhält man bei allerdings längerer Suchzeit auch "einfallsreiche" Änderungen in den Plänen. Ebenfalls muß zugegeben werden, daß eine optimale Verteilung von Auswahlwahrscheinlichkeiten für genetische Operatoren schlecht vorherzusagen ist.

2.5. Zusätzliche Parameter im Genetischen Algorithmus

Besonderen Einfluss auf die Effizienz des Genetischen Algorithmus haben noch folgende Parameter:

a) Anzahl der Individuen in der Population:
   Zu wenig Individuen können nicht genügend Information parallel halten, woraus sich eine zu geringe Abdeckung des Suchraums ergibt. Dagegen ist bei einer reichlich bemessenen Population zu viel Rechenaufwand auf redundanter Information erforderlich.
b) Umfang der Leistungstests:
   Die Leistungstests müssen ausführlich genug sein, um der Bewertungsfunktion hinreichend begründete Leistungswerte des Plans zu liefern. Andererseits erfordert dieser Teil die meiste Rechenzeit, so daß auch hier Probieren nötig ist.
c) Terminierung des Genetischen Algorithmus
   Es gibt viele Möglichkeiten, den Terminierungszeitpunkt festzulegen:

- Bewertungsfunktion überschreitet vorher festgelegten Wert,
- Terminierung nach einer bestimmten Anzahl von Schleifendurchläufen,
- Abbruch dann, wenn sich die Bewertung des besten Individuums während einer bestimmten Anzahl von Durchläufen nicht verändert hat,
- Rechenzeitbegrenzung.

Generell kann zu jedem Zeitpunkt abgebrochen werden und das derzeit beste Individuum als vorläufiges Ergebnis angesehen werden.

2.6. Parameterübersicht

Parameter	Auswirkungen
Poolgröße	großer Pool: Breitensuche, aber Redundanz
anfängliche Planlänge	wenig Einfluß, da Längenänderung möglich
Auswahlwahrscheinlichkeiten der Programmoperationen	höherer Rechenaufwand, wenn sie nicht den Wahrscheinlichkeiten der Operationen in einem guten Plan entsprechen
Parameter der Bewertungsfunktion	Suchrichtung, siehe 2.4.2.
Umfang der Leistunstests	Viele Tests: Rechenaufwand, sonst: Unzuverlässigkeit der Bewertung
Auswahl der Pläne zur Veränderung	Verlust der Suchbreite, wenn nur die besten Pläne gewählt werden
Anzahl Veränderungen je Generation	wenig Änderungen: kleine Entwicklungsschritte, langsame, beständige Entwicklung, kaum Sprünge zwischen lokalen Maxima-Regionen; mehr Änderungen: sprunghafte Entwicklung, wegen Zerstörung von Zwischenergebnissen seltener bessere Nachkommen
Auswahlwahrscheinlichkeiten der genetischen Operatoren	schlecht vorherzusagen, siehe 2.4.3.
Verdrängungsstrategie	fortwährende Verdrängung des schlechtesten Plans: Nivellierung mit fast gleichen Plänen, da fast jede Veränderung eines mittelmäßigen Plans besser ist als der schlechteste
Terminierung	Rechenzeit, Qualität der Ergebnisse

3. Implementierungsbeschreibung

3.1. Überblick

In den meisten bisherigen Versuchen mit Genetischen Algorithmen wurden die Pläne in Zahlenfolgen codiert, um eine leicht zu manipulierende Darstellung zu erhalten /2 S.112 Graefenstette, Fitzpatrick S.136 Davis et al. /. Wenn dabei eine Operation mit Parametern durch mehrere Zahlen dargestellt wird, dann spielt die Position einer Zahl innerhalb der Codierung bei der Interpretation eine entscheidende Rolle, so daß z.B. das Einfügen einer Zahl (= Teil einer Operation) als Seiteneffekt zu einer kompletten Uminterpretation des hinteren Planteils führt. Da diese Änderungen im Plan kaum zu überblicken sind, schlugen wir einen anderen Weg ein:
In der vorliegenden Implementierung des Genetischen Algorithmus bestehen die Individuen des Pools, die ein gegebenes Problem lösen sollen, aus einer Aneinanderreihung von eigens definierten symbolisch dargestellten primitiven Operationen auf der zum Problem gehörigen Datenstruktur. Ein Plan ist also ein Programmquellcode in einer eigens definierten kleinen Programmiersprache (Repräsentationssprache). Die Interpretation der Primitiven wird in entsprechenden Prozeduren realisiert, man kann, sofern es erforderlich oder nützlich erscheint, zusätzliche Primitive vereinbaren und benutzen. Zudem wird die Interpretation nicht von der Position im Plan mitbestimmt und damit obiger Nachteil (Uminterpretation bei Längenänderung) vermieden.
Um ein Problem zu bearbeiten, muß wie folgt vorgegangen werden:

a) Spezifikation der Problemdatenstruktur,
b) Definition der Bewertungs-Funktion
c) Festlegung und Implementierung der primitiven Operationen auf der Datenstruktur,
d) Festlegung der Parameter im Genetischen Algorithmen,
e) Abbruchkriterium bestimmen.

Im Programm für den Genetischen Algorithmus selbst sind dabei nur geringfügige Anpassungen erforderlich.
Als Impementierungssprache wurde PROLOG gewählt, da die Möglichkeit, Datenstrukturen auf einfache Weise als Programme zu verwenden, oft benötigt wurde; und PROLOG interaktives Arbeiten (Variationen im Algorithmus, Parameteränderungen zur Laufzeit) gestattet.

3.2. Behandelte Aufgaben

Die von uns gewählten Beispiele sind recht klein, damit Auswirkungen von Implementierungs- und Parameterentscheidungen besser beobachtet werden können. Wir versuchten die Lösung folgender Probleme:

3.2.1. Abschreiten eines Raumes

In einem schachbrettartig unterteilten Raum soll ein Roboter, von einem einfachen Programm gesteuert, möglichst viele Felder betreten. Die Felder des Raumes sind von West nach Ost und von Süd nach Nord durchnumeriert, der Roboter wird durch seine Position in dem Raum (Koordinate in W-O-Richtung, Koordinate in N-S-Richtung) und seine Orientierung (Blickrichtung N,O,S,W) dargestellt.

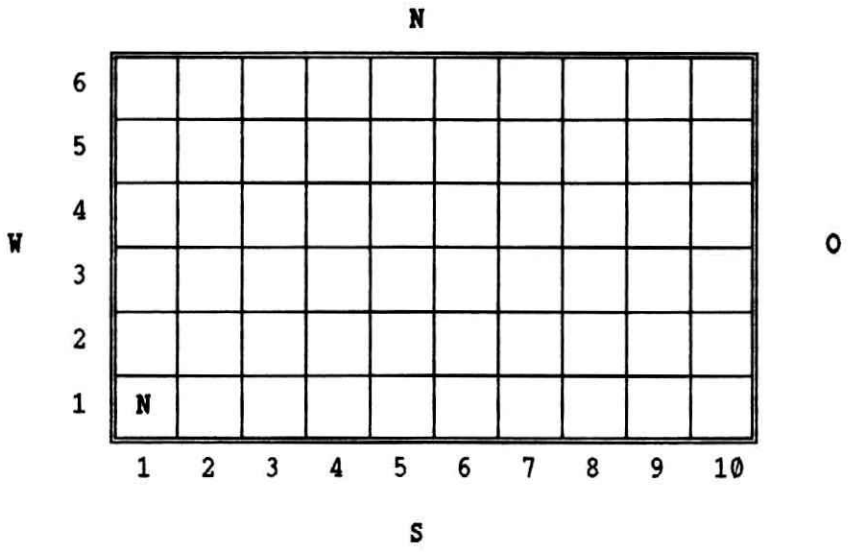

Bild 7. Roboter im Feld (1,1) und Blickrichtung Nord

Für die Steuerungsprogramme sind folgende Primitive vorgesehen:

```
step(Direction)
marke(Nr)
jump(Condition,marke(Nr))
```

"Direction" ist eine absolute (north, south, east, west) oder eine relative (forward, backward, left, right) Richtungsangabe.
"Condition" ist entweder 'true' oder wird durch 'recognize(Direction,wall)' bestimmt (statt 'wall' können in einer Erweiterung auch andere Objekte vorkommen, zum Beispiel 'Feld schon betreten').

Beispiel für einen Plan:

```
(1)  marke(1)
(2)  step(forward)
(3)  jump(recognize(forward,wall),marke(2))
(4)  jump(true,marke(1))
(5)  marke(2)
(6)  step(east)
 .
 .
 .
```

3.2.2. Sortieren

Eine Reihung natürlicher Zahlen soll aufsteigend sortiert werden. Die Datenstruktur besteht hier aus einer Liste natürlicher Zahlen und einem Zeiger auf ein Element der Liste.

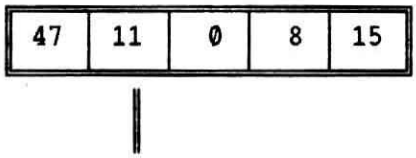

Bild 8. Datenstruktur für das Sortieren

Als Primitive für das Sortierprogramm wurden verwendet:

- step(Direction), der Zeiger auf ein Listenelement wird in die als Parameter gegebene Richtung ('forward' oder 'backward') um ein Element weitergerückt.
- exchangenext, das vom Zeiger markierte Element wird mit dem ihm folgenden vertauscht.
- jump(Condition,marke(Nr)), und marke(Nr), als Sprungkonstrukte, hierbei kann "Condition" entfallen (=true), die Zeigerposition 'am Anfang' bzw. 'am Ende der Liste' abfragen, oder prüfen, ob das markierte Element grösser als das ihm folgende ist.

3.3. Aufbau des Programms

Bei der Programmierung wurde nur recht wenig auf Effizienz geachtet; denn obwohl sehr viele Generationen durchgerechnet werden sollten, erschien es uns zuvorderst wichtig, rasch auch grundlegende (und nicht nur Parameter-) Änderungen am Genetischen Algorithmus vornehmen zu können, was dann auch mehrfach geschah.
Unsere Implementierung des Genetischen Algorithmus besteht aus vier zentralen Prozeduren: INITPOOL,
WHIRLPOOL,
EXECUTEANDCRITIC und
SURVIVAL_OF_THE_FITTEST;
davon werden die letzten drei entsprechend dem Eingangs erläuterten Schema in einer Schleife (EVOLOOP) wiederholt.

INITEVOLUTION führt Vorbereitungen durch:
INITOTHERS besetzt Parameter und Standardlisten der Elemente der Repräsentationssprache vor. Die Parameter sind:
- Anzahl der Schleifendurchläufe in EVOLOOP,
- Anzahl der Testläufe pro Plan,
- maximale Zahl der Marken in einem Programm,
- Listenlängen für Elemente der Repräsentationssprache.

Als Argument von history wird full oder off gespeichert, je nach dem, ob Meldungen über den Programmablauf erwünscht sind oder nicht.
Dann werden eventuell vorhandene alte Listen der Sprachelemente enfernt, und die Standardlisten der Primitiven, Bedingungen, Richtungen, Objekten, die der Roboter erkennen kann und der Parametertypen für die Primitiven in die globale Datenbasis des Prologinterpreters geschrieben.
INITCRITIC übernimmt die Besetzung der Parameter der Bewertungsfunktion,
INITGENOPERATORS die Besetzung der Auswahlwahrscheinlichkeiten der genetischen Operatoren, dann erzeugt
INITPOOL Pläne, indem Operationen zufällig ausgewählt, (evtl. rekursiv) parametrisiert, und zu einem Plan aneinandergefügt werden. Falls erforderlich, kann die Auswahlwahrscheinlichkeit für bestimmte Operationen bzw. Parameter verändert werden, indem sie in der entsprechenden Standardliste mehrfach aufgeführt werden.
NORMALIZE eliminiert Mehrfachvorkommen von Marken und numeriert die verbleibenden programmstrukturerhaltend durch; dies ist auch nach Anwendung

bestimmter genetischer Operatoren notwendig. Dann werden die Operationen als Programmlisten von Prologprozeduraufrufen, die die Interpretation der Primitiven durchführen, gespeichert.
CRITICALL bewertet jetzt die Pläne des Pools unter Verwendung von executeandcritic (siehe dort). Zusätzlich werden noch die Nummern des besten und schlechtesten Planes gesondert gespeichert.

EVOLUTION bzw. CONTEVOLUTION rufen die zentrale Schleife EVOLOOP auf, die aus whirlpool, executeandcritic und survival_of_the_fittest besteht.

WHIRLPOOL erzeugt einen neuen Plan durch Anwendung eines genetischen Operators auf einen alten, der von pickplan aus dem Pool gewählt wird. Gute Pläne sollen entsprechend der stärkeren Vermehrung erfolgreicher Individuen in der Natur häufiger verwendet werden. Dazu wird eine Zufallszahl R zwischen 1 und der Summe der Bewertungen aller Pläne bestimmt. Jetzt werden die Bewertungen der Pläne der Reihe nach so lange addiert, bis die Zwischensumme die Zufallszahl R übersteigt (Pläne mit negativer Bewertung, wie sie z.B. bei starker Berücksichtigung fehlerhafter Schritte auftreten können, werden hierbei mit Bewertung +5 geführt, damit auch sie gelegentlich noch ausgewählt werden in der Hoffnung, daß einer "ökologischen Nische" irgendwann ein Erfolgsrezept entspringt). Der Plan, dessen Bewertung zuletzt addiert wurde, wird für die genetische Veränderung durch Anwendung eines der folgenden genetischen Operatoren benutzt:

Parameter-Mutation:	Eine zufällig gewählte Operation wird neu parametrisiert.
Mutation:	Eine Operation wird durch eine neue ersetzt.
Crossover:	Ein Stück eines Plans wird herausgetrennt und durch ein Teil eines anderen ersetzt.
Delete:	Eine Operation wird entfernt.
Insert:	Eine Operation wird eingefügt.
Insertjump:	Fügt eine jump-Operation und eine Ansprungstelle ein (Schleife oder "Sprung nach vorn").
Invert:	Trennt ein Stück aus einem Plan und fügt die herausgeschnittenen Operationen in Umgekehrter Reihenfolge wieder ein.

Nach Anwendung eines genetischen Operators (außer Delete und Invert) muss der Plan wieder (mit normalize) normalisiert werden.
Anmerkung: In normalize werden Marken, die zwar als Sprungziel, nicht aber selbst im Programm vorkommen, belassen, so daß nach distinguish durchaus sinnlose Befehle wie "jump(true,marke(104))" vorkommen können, obwohl die Anzahl der Marken meist auf 10 beschränkt war.

EXECUTEANDCRITIC
führt den (neu entstandenen) Plan schrittweise aus, wobei zur anschliessenden Bewertung mitgezählt werden:

- ausgeführte Programmschritte,
- Ausführungszeit (einzelne Schritte können länger dauern)
- fehlerhafte Schritte (z.B. gegen Wand gelaufen),
- korrekte Schritte,
- Anzahl der Sprünge auf eine Marke.

Nach Abbruch der Ausführung durch:

- Zeitüberschreitung,
- zu viele Schritte,
- Programmende ereicht,
- zu viele Sprünge auf eine Marke

wird der Plan in Critic bewertet:

```
Bewertung =  + Anzahl betretener Felder bzw. Sortiertheitsmaß
             + korrekte Schritte * k_S_Gewichtung
             - fehlerh. Schritte * f_S_Gewichtung
             - Planlänge         * P_l_Gewichtung
             - Zeit              * Zeit_Gewichtung
```

SURVIVAL_OF_THE_FITTEST
entscheidet, ob sich der neue Plan im Pool durchsetzt oder ob er verworfen wird. Hierfür gibt es verschiedene Strategien:
(i) Ist der neue Plan besser als der schlechteste im Pool, so wird dieser verdrängt und der neue in den Pool aufgenommen. Bei diesem Verfahren wird der Pool von fast gleichartigen Individuen überschwemmt.
(ii) Der neue Plan muss besser sein als sein Vater (der Plan, aus dem er entstanden ist), welchen er dann aus dem Pool verdrängt.

4. Ergebnisse

4.1. Abschreiten eines Raumes

Schon bei diesem einfachen Beispiel zeigte sich eine sehr unklare Beziehung zwischen der Bewertungsfunktion und der Leistung eines Planes; es ist unwahrscheinlich, daß mit einer beliebigen, aber dem Menschen sinnvoll erscheinenden Bewertungsfunktion (bereits hier gibt es große Unterschiede) gleich in die richtige Richtung gesucht wird. Dies stellt unserer Meinung nach das größte Problem bei der Anwendung des Genetischen Algorithmus dar. Ganz abgesehen davon ist es unklar, mit welcher Aufteilung der Anwendungswahrscheinlichkeiten der Genetischen Operatoren gute Ergebnisse zu erzielen sind.
Dieser Parameterraum ist nicht nur wegen seiner Größe problematisch, sondern auch wegen der nicht voneinander unabhängigen Wirkungen der Parameteränderungen, wobei jeder einzelne entscheidenden Einfluß auf das Gesamtergebnis haben kann. In umfangreichen Versuchen stellten sich besonders folgende Detailprobleme heraus:

- Eine Überschwemmung des Pools mit vielen fast gleichartigen Plänen erhält man, wenn die Genetische Operatoren Crossover und Invert zu häufig angewandt werden.

- Zu fortwährender Kürzung der Pläne bis zur leeren Liste kann es kommen, wenn die Länge der Lösung zu stark negativ in die Bewertungsfunktion eingeht. Ohne Bewertung der Länge trat dagegen der umgekehrte Effekt auf, also Verlängerung der Pläne. Die verlängerten Pläne enthielten keine sinnvollen Schleifen, sondern hauptsächlich Einzelschritte, mit denen man natürlich auch einen Raum abschreiten kann, was aber nicht in unserem Sinn ist. Der Operator DELETE ist nur bedingt für die Kürzung der zu lang geratenen Pläne sinnvoll, da er sehr unkontrolliert auch eventuell wichtige Teile des Planes löscht.

- Positive Bewertung der korrekten Schritte führte teilweise zu Plänen, die zwar Schleifen enthielten, aber keine Anweisungen dazwischen. Das Durchlaufen der leeren Schleifen ergab auch eine gute Bewertung, da jede auführbare Operation auch als korrekter Schritt gewertet wurde.

- Es entstand eine große Anzahl von Plänen, die mehr oder weniger geschickt an den Wänden entlang den Raum umrundeten. Der Grund dafür ist darin zu sehen, daß wir in unserer Repräsentationssprache keinen Vergleich auf "bereits betreten" haben. Mit einer solchen Vergleichsmöglichkeit könnten die umlaufenden Pläne leicht zu spiralförmigem Abschreiten kommen. Mit der gegenwärtigen Repräsentationssprache ist ein optimal abschreitendes Programm dagegen schwieriger (siehe Bilder 9,10).

```
marke(1)
  step(forward)
  jump(recognize(forward,nicht betreten),marke(1))
  step(right)
  jump(recognize(forward,nicht betreten),marke(1))
```

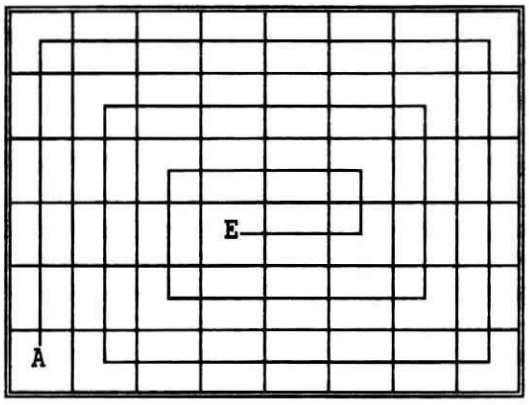

Bild 9. Optimales Abschreiten mit
Vergleichsmöglichkeit auf "schon betreten"

```
marke(1)
  step(forward)
  jump(recognize(forward,[]),marke(1))
  jump(recognize(right,wall),marke(3))
  step(right)
  step(right)
marke(2)
  step(forward)
  jump(recognize(forward,[]),marke(2))
  jump(recognize(left,wall),marke(3))
  step(left)
  step(left)
  jump(true,marke(1)
marke(3)
```

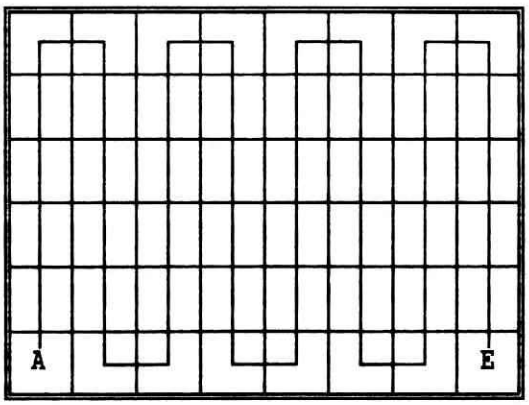

Bild 10. Optimales Abschreiten ohne
Vergleichsmöglichkeit auf "schon betreten"

Es ist nicht verwunderlich, daß das Programm mit dem neuen Vergleich kürzer ist, da die Datenstruktur zur Abarbeitung aufwendiger ist. Hier ist deutlich der Einfluß der Repräsentationssprache auf die Resultate erkennbar.

4.2. Sortieren

Ein wesentlicher Unterschied zwischen diesem und dem vorigen Problem bzgl. der Lösung durch den Genetischen Algorithmus ist die Bewertbarkeit der erzeugten Pläne: Während im ersten Problem im wesentlichen die Anzahl der tatsächlich betretenen Felder posititiv und die Zeit (≈doppelt betretene Felder) negativ in die Bewertung eingeht, stellte sich bei dem Versuch, Sortierprogramme zu erzeugen, heraus, daß "vollständige Lösungen" kaum zu erreichen sind, gute Näherungslösungen aber nicht leicht von einer Bewertungsfunktion zu erkennen oder beschreiben sind. Das Hauptproblem dabei ist: Was heißt "fast" sortiert? Wie soll die Bewertungsfunktion erkennen, ob ein Programm "fast richtig" ist, aber wegen eines kleinen Fehlers noch keinen Sortiererfolg hat? Ein Bubblesort oder Internsort wurde vom Genetischen Algorithmus in unseren Tests jedenfalls nicht gefunden.

5. Ausblick

Bei dem oben genannten Beispiel (Delete zur Kürzung zu lang geratener Pläne) ist deutlich zu sehen, daß die ungesteuerte Anwendung von Genetische Operatoren (die übrigens auch nicht dem heutigen Wissenstand der biologischen Forschung entspricht) durch eine zielgerichtete ersetzt werden sollte. Diese organisierende Instanz müßte umfangreiches Wissen über meist erfolgreiche Programmiertechniken der Repräsentationssprache besitzen oder erlernen. Bestandteile dieses Wissens könnten sein:

- Konsistenzprüfung von Marken
    - keine Sprünge zu nicht exsistenten Marken
    - keine Marken die nie angesprungen werden
- keine Schleifen mit leerem Anweisungsteil
- ...

Weiter scheint uns die Forderung nach Wissen über Strukturierung und Programmhierarchie sinnvoll, obwohl dies mit hohem Aufwand verbunden ist.

Einen kleinen Versuch in dieser Richtung stellt das von uns implementierte JUMPINSERT dar, das eingesetzt wurde, weil wir Schleifen bzw. sinnvolle Sprünge als "oft erfolgreiche" Konstruktionen betrachten und diese Meinung bestätigt sahen.
Da der Genetische Algorithmus normalerweise in Bereichen mit wenigem bekanntem problemspezifischem Wissen eingesetzt wird, muß in unserem Beispiel, in dem die beste Lösung bekannt ist, darauf geachtet werden, kein derartiges Wissen mit einzubringen.

Oft wird der Vorwurf erhoben, mit Genetischen Algorithmen eine reine Zufallssuche durchzuführen. Wir sind jedoch trotz nur zum Teil ermutigender Ergebnisse anderer Meinung, insbesondere weil es zu klären gilt, inwieweit oben genannte Erweiterungen Verbesserungen bringen.