Genetische Algorithmen: Ein anderer Ansatz

Das Prinzip Zufall

Ein ganz anderer Ansatz zur Lösung als der A*-Algorithmus verfolgt mit den Evolutionären Algorithmen[1] eine grundsätzlich andere Suchstrategie. Es ist eine Strategie, die wir am Anfang unseres Kurses nur ganz nebenbei erwähnt haben, eigentlich nur der Vollständigkeit halber, und dann gleich wieder verworfen haben. Es ist der Ansatz von zufälliger Suche!

Wir waren uns schon am Anfang klar geworden, dass wir bei diesem Ansatz damit rechnen müssen, gar nicht zur Lösung zu kommen, angewandt auf das tsp heißt das, nicht zu einem Weg minimaler Länge zu kommen.

Ein solches Verfahren liefert uns in der Regel also nicht "die" optimale Lösung, sondern nur eine suboptimale[2]. Das ist aber auch schon viel, wenn man denn keine andere Methode hat, deren Lösungen die Qualität der so erreichten haben.

Vorbedingungen

Wir wollen dabei die Grundlage nicht vergessen. An erster Stelle sollte bei der Betrachtung von Algorithmen die Frage stehen: Warum überhaupt ...?

Wie wir inzwischen herausgefunden haben, ist der wesentliche Grund, dass es Probleme gibt, die auf anderem Wege nicht oder nicht effektiv genug gelöst werden können. Wie kann das sein, wo doch die Leistung von Computern in den letzten Jahren so immens zugenommen hat?

Die Antwort ist sehr einfach: Es sind Probleme, die so groß sind, dass selbst mit den schnellsten denkbaren Computern eine Lösung innerhalb der Lebensdauer des Weltalls nicht möglich ist. Insbesondere die [np-vollständigen Probleme haben (bisher?) einer vollständigen Lösung standhaft getrotzt. Viele von ihnen haben neben dieser vertrackten Eigenschaft aber gleichzeitig die Möglichkeit, prinzipiell mögliche Zustände des Systems nicht nur anzugeben, sondern sie auch schnell zu bewerten.

Welche Bedingungen müssen für einen Ansatz mit Evolutionären Algorithmen vorgegeben sein?

1. Das Problem ist ein Suchproblem, da es kein direktes Lösungsverfahren gibt.
2. Der Suchraum ist so groß, dass eine vollständige Suche versagen muss.
3. Es gibt aber eine relativ schnelle Bewertungsfunktion für Zustände im Lösungsraum.
4. Keines der optimierten Suchverfahren liefert eine Lösung.

nicht deterministische Optimierung

Eberhard Schöneburg schreibt in seinem Buch: "Genetische Algorithmen und Evolutionsstrategien : Eine Einführung in Theorie und Praxis der simulierten Evolution"

1 Evolutionäre Algorithmen sind begrifflich etwas weiter gefasst als der Begriff der Genetischen Algorithmen, deren Anwendung im entsprechenden Abschnitt beschrieben wird

2 Beachten Sie, dass die wirklich optimale Lösung zu finden, in vielen Anwendungen eine im strengen Sinne "akademische" Frage ist, die den Anwender herzlich wenig interessiert. Der will nur möglichst wenig Kosten haben, das soll aber jetzt erreicht werden und nicht in ferner Zukunft!

Die Grundidee nicht deterministischer Optimierungsverfahren besteht darin, anstelle von komplizierten deterministischen Vorschriften bei der Suche nach den Optima in großen Suchräumen, vom Zufall Gebrauch zu machen. Dies scheint naiv betrachtet nicht sinnvoll, da der Zufall gerade in sehr großen Räumen keine brauchbare Richtschnur für das Suchen bereitzustellen scheint. Suchverfahren, die auf der Basis von Zufallsprozessen arbeiten, scheinen daher ineffizient zu sein.

Diese naive Sicht ist jedoch in der Regel falsch. Man neigt dazu, Zufall mit Willkür gleichzusetzen, und das ist ein Fehler, denn man kann sehr systematisch und effizient mit dem Zufall arbeiten, obwohl dies beinahe wie ein Widerspruch klingt. Die systematische Nutzung des Zufalls ist eines der Erfolgsrezepte der Evolution. Wenn über die Lage der Optima in einem großen Suchraum keine Kenntnisse vorhanden sind, und das ist in der Praxis leider nur zu oft der Fall, sind zufallsgesteuerte Verfahren eventuell weitaus effizienter als deterministische Algorithmen. Die Gefahr, Optima zu »verpassen«, ist beispielsweise bei einer gleichmäßigen Streuung der Zufallsstichproben im Suchraum relativ gering.

Diese Gefahr besteht jedoch bei einer deterministischen Suche immer dann, wenn die Systematik, also die Art der Suche, so angelegt ist, daß durch sie, ohne daß man es weiß, die optimalen Werte nicht gefunden werden können. Basiert das deterministische Verfahren auf einer auch nur geringfügig falschen Annahme, kann es bereits völlig nutzlos sein. Wenn man nicht weiß, wo die Optima im Suchraum liegen und wie sie aussehen, so weiß man in der Regel auch nicht, ob man die richtige Suchstrategie gewählt und die richtigen Annahmen getroffen hat. In diesen Fällen ist eine systematische, zufallsbasierte Suche daher oft zuverlässiger als ein Algorithmus, bei dem jeder Suchschritt exakt vorgegeben wird. Denn kennt man die Verteilung der Stichproben im Suchraum, also den Zufallsprozeß auf dem ein nicht deterministisches Verfahren basiert, so kann man zumindest die Wahrscheinlichkeit abschätzen, mit der man die Optima findet - oder verpaßt! Dies ist bei deterministischen Verfahren oft noch nicht einmal ansatzweise möglich.

Zufall als Prinzip

In LOG IN 21 (2001) Heft 1 finden wir einen Artikel mit dem Thema Zufall und zufallsgesteuerte Algorithmen von Juraj Hromkovic. Dort heißt es in der Einleitung:

"Zufallssteuerung ist heutzutage ein fester Bestandteil von Informatiksystemen, und wir verdanken ihr einige der faszinierendsten mathematisch-naturwissenschaftlichen Entdeckungen in der Informatik. Das erste Ziel dieses Artikels ist es, den Lesern an einem einfachen Beispiel zu zeigen, dass zufallsgesteuerte Systeme und Programme eine Leistungsfähigkeit besitzen, die deterministische nie erreichen können. Das zweite Ziel ist es, anhand der "Methode der häufigen Zeugen" zum Verständnis der Berechnungsstärke zufallsgesteuerter Algorithmen beizutragen."

Weiter heißt es dort:

"Die meisten Menschen haben zum Zufall eine negative emotionale Einstellung, weil sie Zufall mit Ungewissheit verbinden. Am liebsten hätten sie eine exakte Zukunftsvorhersage, um auf alle möglichen Alternativen vorbereitet zu sein; ersatzweise schließen sie viele Versicherungen ab, um gegen Eventualitäten gewappnet zu sein. Einige Menschen spielen aber auch gerne mit Zufall, insbesondere wenn die Vision eines Lotteriegewinns bevorsteht."

und weiter unten:

"Die Physiker waren die Ersten, die auf die Idee kamen, durch zufallsgesteuerte Systeme Naturprozesse zu simulieren. Der große Nutzen von Zufallssteuerung fand aber erst in der Informatik durch den Entwurf von zufallsgesteuerten Algorithmen statt. Was sind aber zufallsgesteuerte Algorithmen? Die Programme (Algorithmen), die wir gewohnt sind, sind deterministisch. "Deterministisch" bedeutet, dass das Programm und die Problemeingabe die Arbeit des Programms vollständig bestimmen. In jedem Augenblick ist in Abhängigkeit von den aktuellen Daten eindeutig klar, was die nächste Aktion des Programms sein wird. Zufallsgesteuerte Programme können mehrere Möglichkeiten haben, ihre Arbeit fortzusetzen, und welcher der Möglichkeiten das Programm folgt, wird zufällig entschieden. Es sieht so aus, als ob ein zufallsgesteuertes Programm von Zeit zu Zeit eine Münze wirft, und abhängig davon, ob Kopf oder Zahl gefallen ist, wählt es die entsprechende Strategie auf seiner Suche nach dem richtigen Resultat. Auf diese Weise kann ein zufallsgesteuertes Programm mehrere unterschiedliche Berechnungen für ein und dieselbe Eingabe durchlaufen. Im Unterschied zu deterministischen Programmen, wo immer eine zuverlässige Berechnung des richtigen Resultates gefordert wird, dürfen einige Berechnungen des zufallsgesteuerten Programms auch falsche Resultate liefern. Das Ziel ist jedoch, die Wahrscheinlichkeit einer falschen Berechnung klein zu halten, was in gewissem Sinne bedeutet, den proportionalen Anteil der Berechnungen mit falschem Resultat zu reduzieren."

Bei den GA sind wir glücklicherweise noch in einer etwas besseren Lage: Wir brauchen geradezu zwischendurch "falsche" Zwischenergebnisse, um die genetische Vielfalt sicherzustellen. Durch den iterativen Charakter des Algorithmus werden auch zwischenzeitliche Verschlechterungen verkraftet. Danach sind weitere Verbesserungen immer noch möglich. Weiter aus dem Artikel:

"Auf den ersten Blick sieht ein zufallsgesteuertes Programm unzuverlässig aus im Vergleich zu deterministischen Programmen, und man kann fragen, wozu es gut sein soll, gelegentlich falsche Resultate zu erhalten. Es existieren aber Aufgaben von enormer praktischer Bedeutung, bei denen der schnellste deterministische Algorithmus auf dem schnellsten Rechner mehr Zeit zur Berechnung braucht als die Zeit vom Urknall bis zum heutigen Tag, die Aufgabe also praktisch unlösbar ist.
Und da kommt ein "Wunder" - ein zufallsgesteuerter Algorithmus, der die Aufgabe in ein paar Minuten auf einem Standard-PC löst mit einer Fehlerwahrscheinlichkeit von l zu Tausend Milliarden. Warum kann man so ein Programm trotzdem als zuverlässig ansehen, obwohl es Fehler macht? Ganz einfach: Ein herkömmliches deterministisches Programm, das diese Aufgabe in einem Tag berechnet, ist viel unzuverlässiger als unser zufallsgesteuertes Programm, weil die Wahrscheinlichkeit des Auftretens eines Hardwarefehlers während des 24-stündigen Programmlaufs viel höher ist als die Wahrscheinlichkeit einer fehlerhaften Ausgabe des schnellen zufallsgesteuerten Algorithmus."

In dem angegebenen Artikel wird eine spezielle Anwendung auf ein kryptologisches Problem weiter untersucht. Am Schluss heisst es dann:

"Die Methode der häufigen Zeugen ist ein leistungsfähiges Verfahren zum Entwurf von zufallsgesteuerten Algorithmen. Der effiziente zufallsgesteuerte Primzahltest basiert auf dieser Methode und gehört damit zu den wichtigsten Anwendungen von großer praktischer Bedeutung in der Kryptographie. Die besten bekannten deterministischen Algorithmen würden für den Primzahltest großer Zahlen mehr Zeit benötigen als die Lebensdauer des Universums. ...
Aber für die Praxis ist es im Augenblick auch nicht wichtig, ob schnelle deterministische Algorithmen für die Aufgaben existieren oder nicht. Das Einzige, was zählt, ist, dass die Lösungen von Problemen durch zufallsgesteuerte Algorithmen effizient und hinreichend zuverlässig sind. Den Unterschied zwischen der Effizienz der bestmöglichen deterministischen Algorithmen und der bestmöglichen zufallsgesteuerten Algorithmen zu bestimmen, bleibt eine der zentralen

Forschungsaufgaben der theoretischen Informatik."

Gott würfelt eben doch !

Genetische Algorithmen sind ein Weg, Probleme aus dem Bereich Suchverfahren und Optimierung zu lösen, bei denen man die genetischen Prozesse der Natur abbildet. "Gott würfelt eben doch", und das sehr wirkungsvoll, wie die belebte Natur zeigt.

Inzwischen ist es in der Technik und Informatik ein vielfach gegangener Weg nachzumachen, was uns die Natur vorgemacht hat. Für diesen an vielen Stellen schon sehr erfolgreichen Gedanken hat sich inzwischen der Name Bionik festgesetzt.

Im Ausstellungkatalog zur Bionik - Ausstellung (1998) hieß es:

"Bionik - Natur als Vorbild: ... Ziel der Bionik ist die Übertragung von Problemlösungen der Natur in den Bereich der Technik, um die in Jahrmillionen entwickelten und optimierten 'Erfindungen der Natur' zu nutzen ."

Evolutionäre Algorithmen setzen nun gerade den Kernbereich der biologischen Entwicklungstrategien um, die bei der Fortpflanzung auftretende Evolution. Es ist das Zusammenspiel von Informationserhaltung und moderater Veränderung der Information mit dem Prozess der natürlichen Auslese, das die Natur so erfolgreich gemacht hat. Evolutionäre Algorithmen bilden diese Vorgänge nach, allerdings nicht in der Absicht, sie "blind" zu kopieren. Wie immer in der Informatik, verändert der Modellbildungsprozess.

Der Grundgedanke bei den Genetischen Algorithmen

Der Grundgedanke lautet:

> Wenn die Natur bei der Entwicklung optimaler Systeme so erfolgreich ist,
> dann versuchen wir ihre Prozesse zu verstehen und nachzubauen.

Vieles an dieser Kurzformulierung stimmt natürlich nicht oder nur bedingt. Sie ist hoffentlich im selben Sinne optimal wie die Natur: Sie ist nicht die beste, aber erfolgreich! Evolution ist ein sehr effektiver Optimierungsprozess! Wir werden bei den Genetischen Algorithmen also in vielen Fällen zwar nicht die beste oder auch nur eine von mehreren gleichwertigen besten Lösungen finden, aber eine gute, die meistens besser ist, als eine auf einem anderen Wege zu findende Lösung. Wenn das nicht so ist, nehme man diesen anderen Weg!

Die Genetischen Algorithmen simulieren also die natürlichen Vorgänge der Evolution. Dabei kommt es darauf an, die grundlegenden Prozesse zu erkennen, um sie nachbilden zu können.

Natürliche Organismen sind in der Regel Teile von Populationen, in denen sie mit einander konkurrieren. Sie konkurrieren dabei einmal mit Individuen derselben Spezies, aber auch mit Individuen von anderen Spezien. Mit allen konkurrieren sie um die elementaren Lebensgrundlagen wie Nahrung, Wasser und Licht, in einigen Fällen auch andere wichtige Dinge wie Wohnraum.

Zusätzlich konkurrieren sie auch noch mit den Mitgliedern der eigenen Spezies um die Chance zur Fortpflanzung. Nur die Individuen, die soweit überleben und sich so weit durchsetzen, dass sie sich fortpflanzen können, haben in diesem biologischen Sinne ihr Lebensziel erreicht. Das im evolutionären Sinne definierte Lebensziel eines Individuums ist also eine maximale Verbreitung seiner eigenen Gene.

In diesem Sinne lässt sich also eine Bewertung der Individuen durchführen -die ist von uns dann im Modellbildungsprozess nachzubilden- und auf der Grundlage dieser Bewertung erfolgen dann die Veränderungen der Populationen durch Fortpflanzung.

Dabei gilt selbstverständlich auch hier, dass dieser Prozess nicht gradlinig erfolgt. Selbstverständlich gibt es "Unfälle" und andere Zufälle in der Natur, die nicht im exakten mathematischen Sinne den Kriterien des Optimalen gerecht werden. Auch das kann Teil des zu modellierenden Prozesses sein. Dass zu gradlinige Entwicklungen z.B. durch lokale Optimierung leicht zum Verrennen in lokalen Maxima führen können, sollten wir bei den greedy Algorithmen gelernt haben.

Die wichtigen Komponenten eines GA

Die wichtigen Komponenten eines Genetischen Algorithmus sind:

- der Pool

  Es wird - in der Regel mit einem Zufallsprozess - eine größere Zahl von Individuen erzeugt, die für sich lebensfähig sind. Beim tsp müssen es also, wenn auch schlechte, aber mögliche Rundwege sein. Sie stellen in diesen Fall selbst die Gene dar.

- die Mutation

  Einige zufällig ausgewählte "Gene" werden an zufällig ausgewählten Stellen verändert. Beim tsp wird z.B. ein Stadt durch eine andere ersetzt.

- das crossing over

  Es werden zufällig je zwei Individuen ausgewählt und mit ihnen ein Kreuzen der "Gene" durchgeführt. Beim tsp wählt man z.B. zwei Positionen in den Touren aus und tauscht deren Verlauf zwischen den beiden Touren aus.

Gerade die mögliche ungezielte Veränderung von Individuen durch die beiden o.a. Schritte ist wichtig für die Lösung von Suchproblemen mit Genetischen Algorithmen, da sie uns die Möglichkeit gibt, das Problem des Hängenbleibens in lokalen Maxima zu vermeiden.

- die fitness

  Oben wurde angegeben, dass eine einfache Bewertungsfunktion für die Güte der Individuen vorliegen muss. Ihr Wert definiert die fitness. Beim tsp ist dies einfach die Länge der Tour.

- die Selektion

  Auf der Grundlage der Bewertung durch die Fitnessfunktion werden die Individuen ausgewählt. Es gibt verschiedene Modellierungen, die alle gemeinsam haben, dass Individuen mit höherer fitness eine bessere Chance haben, in der nächsten Generation aufzutreten, als schlechtere Individuen.

- die Generationenfolge

  Die ausgewählten Individuen bilden den Pool der nächsten Generation. Es wird also ein Schleifenprozess durchgeführt, bis irgendein Abbruchkriterium erfüllt ist.

der Pool

Es wird - in der Regel mit einem Zufallsprozess - eine größere Zahl von Individuen

erzeugt, die für sich lebensfähig sind. Beim tsp müssen es also, wenn auch schlechte, aber mögliche Rundwege sein. Sie stellen in diesen Fall selbst die Gene dar.

Die Überschrift dieser Seite aus biologischer Sicht wäre Population. Wollen wir genetische Prozesse in der Natur simulieren, dann müssen wir eine größere Zahl von prinzipiell lebensfähigen Individuen erzeugen, die Population. Sie müssen die Kriterien erfüllen, die ihnen die Möglichkeit geben, sich zu reproduzieren. Aus der Sicht der Gene heißt das: Die Individuen der Population müssen mit einem vollständigen Satz von Genen ausgestattet sein.

In unserem informatischen Modell heißt das entsprechend: Alle Individuen müssen das Kriterium erfüllen, zulässige Teilschritte auf dem Weg zu einer Lösung bzw. je nach Problemstellung -Optimierung oder Satisfizierung- zu einer optimalen Lösung zu sein.

Wendet man das auf das Beispiel des travelling - salesperson - problem an, ist zu fordern, dass die einzelnen Individuen überhaupt Rundwege darstellen. Beim CLP könnte eine Forderung lauten, dass kein Container überfüllt ist oder andere zwingende Rahmenbedingungen verletzt werden.

Weiterhin müssen wir fordern, dass es sich nicht nur um ein einzelnes Individuum, sondern um eine größere Zahl von Individuen handelt, da nur dann sehr vielseitige Interaktionen zwischen diesen möglich sind.

Dritte Forderung ist, dass eine ausreichende Diversifikation gilt. Sind alle Individuen der Population gleich, ist kaum mit interessanten Veränderungen zu rechnen.

Am Anfang der Entwicklung steht daher folgende Reihenfolge:

- Schreibe eine (mehrere) Funktion(en), die lebensfähige Individuen zufällig erzeugen.
- Füge diese einer Datenstruktur pool hinzu, bis eine vorgegebene Anzahl erreicht ist, dies ist die Populationsgröße der Startpopulation.

Die Mutation

Bei der Mutation wird ein Individuum ausgewählt. Davon werden einige zufällig ausgewählte "Gene" an zufällig ausgewählten Stellen verändert. Beim tsp kann man z.B. innerhalb einer Tour eine Stadt gegen eine andere austauschen. Beim CLP könnte ein Teile eines Containerinhaltes mit einem Teil aus einem anderen Container oder gegen ein Teil, das noch "am Kai steht" ausgetauscht werden.

Auch in diesem Prozess sind zwei Schritte zu berücksichtigen :

1. Was ist im Modul inhaltlich zu realisieren?
2. Wie wird das Zufallsprinzip eingebaut?

Der zweite Schritt ist einfach zu beantworten: Alle Auswahlschritte haben zufällig zu erfolgen! Im Beispiel des tsp muss die Auswahl der beiden o.a. Städte zufällig erfolgen. Genau so hat beim CLP die jeweilige Auswahl, einmal des Containers und andererseits des zu ersetzenden Teils zufällig zu erfolgen. Beim CLP ergibt sich außerdem die Möglichkeit, die Gesamtanzahl der Teile im Container dabei zu verändern, also eines hinzuzufügen oder zu entfernen. Auch die Frage, ob das erfolgt, muss zufällig beantwortet werden. Auch die Beantwortung der ersten o.a. Frage ist nicht in jedem Fall selbstverständlich.

Es geht also zuerst um die Frage: Wie kann der biologische Prozess der Mutation bei unseren informatischen Systemen modelliert werden?

Das crossing over

Es werden zufällig je zwei Individuen ausgewählt und mit ihnen ein Kreuzen der "Gene" durchgeführt. Beim tsp wählt man z.B. zwei Positionen in den Touren aus und tauscht deren Verlauf zwischen den beiden Touren aus. Beim CLP werden z.B. ganze Abschnitte der Teilelisten von zwei Containern untereinander ausgetauscht.

Gerade die mögliche ungezielte Veränderung von Individuen durch die beiden Schritte Mutation und Kreuzen ist wichtig für die Lösung von Suchproblemen mit Genetischen Algorithmen, da sie uns die Möglichkeit gibt, dass unser Lösungsverfahren lokale Maxima wieder verlassen kann.

Beim crossing over werden zwischen zwei verschiedenen Individuen Gene ausgetauscht! Daher erfolgt auch in diesem Fall an zwei Stellen eine zufällige Auswahl: Einmal müssen aus dem pool zunächst zufällig zwei Individuen ausgewählt werden, zum anderen müssen für die beiden die Bruchstellen festgelegt werden, an denen der Genabschnitt aus dem ersten Individuum durch den entsprechenden Abschnitt aus dem des zweiten Individuums ausgetauscht wird.

Der Vorgang des crossing over ist im Programmsehr viel schwieriger zu realisieren als der vorige Schritt.

Das Problem für die Lösung beim tsp ist, dass eine Tour nur zulässig ist, wenn sie jede Stadt genau einmal enthält. Die Gen - Information steckt also nicht darin, welche Städte in der Tour enthalten sind, sondern in welcher Reihenfolge.

Wenn nun also aus der einen Tour ein Tourenabschnitt in die andere übernommen wird, dann muss man davon ausgehen, dass in diesem Abschnitt sowohl Städte enthalten sind, die in dem ersetzten Abschnitt liegen, als auch Städte, die in dem nicht zu ersetzenden liegen. Und für die gilt nun, dass man sie daraus entfernen muss.

Beim CLP ist darauf zu achten, dass ein Stück nicht gleichzeitig in mehreren Containern untergebracht wird oder verloren geht. Hier wird der Austausch durch die Gruppierung der Stücke zu Containern erheblich komplizierter.

Die fitness

Oben wurde angegeben, dass eine einfache Bewertungsfunktion für die Güte der Individuen vorliegen muss. Ihr Wert definiert die fitness.

Beim tsp ist dies einfach die Länge der Tour. Als Länge wird z.B. der geometrische Abstand genommen. In einem realistischen Beispiel für Städte müsste man die Abstände z.B. aus einer Tabelle entnehmen.

Beim CLP kann es z.B. die Gesamtzahl der Stücke sein, das Gesamtgewicht oder der Gesamtwert, was auch immer für ein Optimierungswert zu dem Problem gehört.

Die Selektion

Auf der Grundlage der Bewertung durch die Fitnessfunktion werden die Individuen ausgewählt. Es gibt verschiedene Modellierungen, die alle gemeinsam haben, dass Individuen mit höherer fitness eine bessere Chance haben, in der nächsten Generation aufzutreten, als schlechtere Individuen.

Der Auswahlprozess bei den GA ist die Selektion. Nach dem Prinzip *"die guten ins Töpfchen, die schlechten ins Kröpfchen"* sorgt sie dafür, dass Individuuen mit einer höheren fitness bevorzugt in die nächste Generation übernommen werden. Durch die vorherigen Prozesse Mutation und crossing over sind in der Regel dadurch zusätzliche Individuen entstanden, dass die neu erzeugten dem pool hinzugefügt wurden.

Auch für das Selektionsverfahren gibt es wieder verschiedene Möglichkeiten. Man kann z.B. im Fall des tsp immer die ***n*** besten Touren übernehmen und alle anderen entfernen. Das führt allerdings leichter zu einem Verrennen in lokalen Minima, als wenn man noch einmal einen Zufallsprozess einsetzt: Es werden zufällig je zwei Touren aus dem vergrößerten pool entnommen und miteinander verglichen. Das Individuum mit der größeren fitness wird dann in den pool der nächsten Generation übernommen. Das macht man solange, bis die Zahl der Elemente im pool genau so groß ist wie am Beginn der Bearbeitung. Wie arbeiten also mit gleichbleibender Populationsgröße. Beim CLP kann man vergleichbar verfahren.

Die Generationenfolge

Die ausgewählten Individuen bilden den Pool der nächsten Generation. Es wird also ein Schleifenprozess durchgeführt, bis irgendein Abbruchkriterium erfüllt ist.

Durch mehrere Schleifendurchläufe unseres Verfahrens wird eine Generationenfolge für die Population simuliert. Man kann zwar in einigen Fällen Abbruchkriterien aus dem Problem heraus definieren - bei den Satisfaktionsproblemen kann man das Erreichen des Zieles feststellen. Erzeugt man mit GA z.B. magische Quadrate, dann ist man fertig --- wenn man denn eines gefunden hat. Sonst wird man beispielsweise die Zahl der Generationen vor der Bearbeitung festlegen - beachten Sie: GA gehören zu den sub - optimierenden Verfahren.

Bei einer Funktionalen Sprache wie Scheme bietet es sich an, die einzelnen Komponenten im Schritt von einer Generation zur nächsten funktional zu schachteln. Das heißt: Jeder Teilprozeß wird aufgerufen vom nächsten und gibt sein Zwischenergebnis -den veränderten pool- an diesen aufrufenden weiter.

```
(Selektion             ; S ruft K und verwendet deren Wert
 (Kreuzen              ; K ruft M und verwendet deren Wert
  (Mutation            ; M bekommt Population und Parameter
    Population
    Mutationswahrscheinlichkeit)
  crossing-over-Wahrscheinlichkeit)  ; Parameter von K
 Populations-Groesse)                ; Parameter von S
```

Dies Zwischenergebnis wird dann dem nächsten Schritt der Generationenfolge übergeben.

Eine gute Aufgabe für Gruppenarbeit in einem Software-Projekt

In der sich aus den geschachtelten Funktionsaufrufen ergebenden Unabhängigkeit der einzelnen Bearbeitungsstufen von einander liegt eine der wesentlichen Voraussetzungen, dies Problem durch modulares Arbeiten zu lösen. Teilaufgaben können problemlos an einzelne Gruppen vergeben werden, die sich untereinander nur über die Schnittstellen ihrer Module verständigen müssen.