Optimierungsalgorithmen auf verteilten Systemen

Das Beste zum Schluss

Rüdiger Berlich

Viele technische und ökonomische Probleme wären nicht lösbar, würden nicht Computer bei der Optimierung ihrer Parameter helfen. Inzwischen stehen in Grids und Clouds enorme Rechenressourcen zur Verfügung, mit denen sich auch hochgradig komplexe Aufgabenstellungen mit parallelisierbaren evolutionären und Schwarmalgorithmen bewältigen lassen.

In der Technik wie im täglichen Leben ist meist der Weg das Ziel. Man weiß nicht, wo das Optimum liegt – sonst würde man es direkt implementieren –, ist aber durchaus in der Lage, neue Lösungen daraufhin zu prüfen, ob sie besser oder schlechter sind als die bereits bekannten.

Dies allgemeingültig festzulegen ist allerdings hohe Kunst. Zum einen werden die wenigsten Personen genau darin übereinstimmen, was in einer bestimmten Situation das Beste sei. Zum anderen mag es mehrere miteinander konkurrierende Ziele geben, die es gegeneinander abzuwägen gilt. Wer heute etwa ein neues Auto kaufen möchte, wird üblicherweise auf einen möglichst umweltfreundlichen und verbrauchsarmen Motor achten, der aber gleichwohl möglichst stark sein soll. In der Managementtheorie füllt die Suche nach optimalen Strategien unter einer Vielzahl miteinander konkurrierender Bewertungskriterien und Randbedingungen ganze Bücher.

Die richtigen Schrauben finden

Optimieren kann also auch bedeuten, gute Kompromisse zu finden. Dem Formulieren der Bewertungskriterien kommt auch in der Technik besondere Bedeutung zu. Dabei spielt der Faktor Mensch die wichtigste Rolle. Die technische Optimierung ersetzt nicht die Erfahrung des menschlichen Protagonisten – vielmehr ist sie zwingend auf sein Wissen angewiesen, und die Arbeit mit einem Optimierungsalgorithmus kann helfen, verborgenes Wissen explizit zu machen.

Den meisten technischen Herausforderungen ist gemein, dass die sie bestimmenden Parameter auf Arten miteinander verknüpft sind, die dem Auge verborgen bleiben. Oder um es etwas salopper auszudrücken: Es ist meist auch dem Fachmann nicht klar, an welchen Schrauben er (gleichzeitig) drehen muss, um ein besseres Ergebnis zu erzielen. Vielmehr sind die auftretenden Fragestellungen so komplex, dass man nur danach streben kann, anhand der gezielten Variation bestimmter Parameter die aktuelle Situation zu verbessern.

Dabei funktioniert die Idee, einfach für jeden Parameter einige Werte auszuprobieren, nur bei einfachen Aufgabenstellungen. Man stelle sich etwa einen Autohersteller vor, der die Leistung eines seiner Motoren verbessern will. Eine wichtige Voraussetzung dafür ist eine optimale Verbrennung. Sie hängt von Parametern wie dem Einspritzdruck, den Winkeln, unter denen die Einspritzdüsen

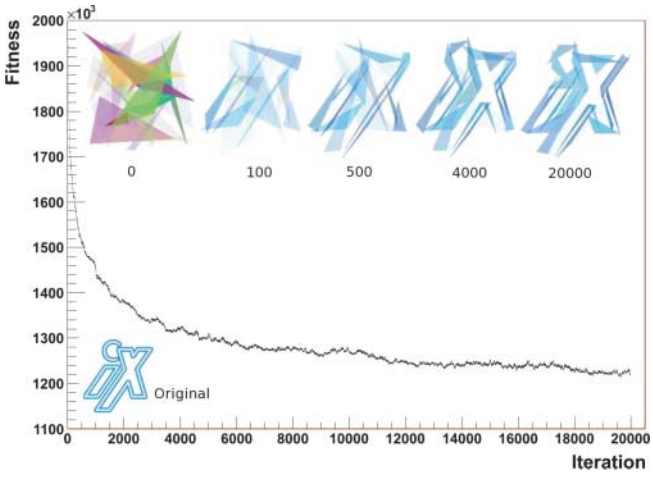
angeordnet sind, dem Zündzeitpunkt und ganz allgemein der Geometrie des Brennraums und der Zylinder ab. Die Vorgänge im Motor gestalten sich zudem unterschiedlich, je nachdem, mit welcher Drehzahl der Motor gerade läuft. Sicherlich lassen sich viele signifikante, eng miteinander korrelierte Parameter finden, bei denen selbst kleinste Änderungen zu einer dramatischen Verschlechterung der Verbrennung führen.

Um einen Parametersatz bewerten und Aussagen über seine Qualität machen zu können, führt man Simulationen im Computer durch, die die Vorgänge im Motor nachbilden. Entsprechende Berechnungen können aufgrund der Komplexität der physikalischen und chemischen Vorgänge Minuten bis Stunden dauern. Geht man beispielsweise von 100 Parametern und jeweils nur 10 Testwerten pro Parameter aus sowie von einer Berechnungszeit von einer Minute pro Wertesatz, müsste ein einzelner CPU-Kern schon mehr als 10^{94} Jahre rechnen, um aus der Menge dieser Kombinationen die beste herauszusuchen. Zum Vergleich: Das Alter des Universums schätzt man heute auf etwa 10^{10} Jahre. Und von einer dichten Abdeckung des Parameterraums ist man gleichwohl weit entfernt.

Wenn man aber nicht wahllos alle 10^{100} Kombinationen durchrechnen kann, muss man sich auf einen Bruchteil beschränken. Die Kunst liegt nun in der Auswahl der zu berechnenden Kombinationen. Wenn man im Vorfeld keine Auswahlkriterien zur Verfügung hat, kann man nur mit einer zufälligen Auswahl beginnen und sich dann den Zielvorgaben annähern. Manche Annäherungs- oder Optimierungsverfahren folgen dabei dem Vorbild der Biologie. Evolutionäre Algorithmen etwa implementieren einen Zyklus aus Mutation, Rekombination und Selektion.

Annäherung ans Optimum

Für Abbildung 1 hat ein solcher evolutionärer Algorithmus die Aufgabe bekommen, die Parameter von 20 semitransparenten, einander teilweise überlappenden Dreiecken so anzupassen, dass sie gemeinsam dem *iX*-Logo möglichst ähnlich sehen. 10 Werte definieren jedes Dreieck: sechs Koordinaten (*x*- und *y*-Werte für jede Ecke im zweidimensionalen Koordinatensystem), drei Farbwerte (RGB) und die Transparenz. Insgesamt galt es also passende Werte für 200 Parameter zu finden. Aus einem Satz Dreiecke setzte die Bewertungsfunktion jeweils ein Bild zusammen. Aus dem Vergleich der Farben jedes einzelnen Pixels mit dem Zielbild ergibt sich dann ein Maß dafür, wie gut eine mögliche Lösung das Logo wiedergibt. Grundlage für die Berechnung bildete ebenso wie in den anderen Beispielen die Bibliothek Geneva (siehe Kasten „Die Geneva-Bibliothek“) [a, b].



Ein evolutionärer Algorithmus bekam die Aufgabe, das Logo der *iX* durch 20 semitransparente Dreiecke zu approximieren. Gezeigt sind einige Zwischenschritte der Optimierung sowie die in jeder Iteration erzielte Qualität oder Fitness. Bereits nach einigen Hundert Generationen ist das Logo zu erkennen (Abb. 1).

Bei der Betrachtung des Bildes sollte man sich vergegenwärtigen, dass dem Algorithmus nichts über das Optimierungsproblem selbst bekannt ist. Insbesondere ist er nicht in der Lage, Ähnlichkeiten aus der Kenntnis des Zielbildes abzuleiten. Für ihn existiert nur die Bewertung, die sich aus dem Vergleich der einzelnen Pixel ergibt. Insofern kann er nicht gezielt Dreiecke dorthin verschieben, wo sie besser passen würden.

Aus Sicht des Optimierungsalgorithmus stellt sich dieses Beispiel sogar komplexer dar als das des Motorenbaus. Gleichwohl ist er mit etwa 4 Millionen Bewertungen ausgekommen – eine fast verschwindend geringe Zahl gegenüber der Zahl der Bewertungen für den Fall, dass jemand wahllos für jeden Parameter nur 2 Werte ausprobiert hätte (2^{200}). Die gesamte Optimierung über 20 000 Iterationen hat etwa 5 Stunden Rechenzeit auf einem Rechner mit Intels Core2-Quad-Prozessor benötigt.

Bewertung als Bewährungsprobe

Man könnte einwenden, dass 4 Millionen Bewertungen bei jeweils einer Minute Rechenzeit wie im Motorenbeispiel gleichwohl noch 7,6 Jahre Rechenzeit benötigen. Der im Bild gezeigte Verlauf der Fitness – er ist typisch für Evolutionsstrategien – zeigt jedoch, dass man bereits innerhalb weniger Iterationen signifikante Verbesserungen der Fitness beobachten kann. Nebenbei bemerkt lassen sich auch deutlich komplexere Bilder mit dieser Methode darstellen, etwa die Mona Lisa mit 300 Dreiecken [a].

Abbildung 1 verdeutlicht zudem, dass zwar noch Verbesserungen möglich sind, weitere Berechnungen jenseits der 20 000 Iterationen jedoch einen kaum vertretbaren Mehraufwand bedeutet hätten. Es hängt also von den bereitstehenden Ressourcen und der verfügbaren Zeit ab, wie viele

iX-TRACT

- Evolutionäre Algorithmen sind iterative Verfahren, die Regeln der natürlichen Evolution anwenden, um das Optimum in einem Lösungsraum zu finden. Schwarmalgorithmen beziehen dazu das Schwarmverhalten von Tieren mit ein.
- Beide eignen sich vor allem dann für die Lösungssuche, wenn die Problemstellungen zu komplex für deterministische Verfahren sind.
- Durch die enormen Rechenressourcen, die in Grids und Clouds quasi auf Abruf bereitstehen, lassen sich mit den massiv parallelisierbaren evolutionären und Schwarmalgorithmen selbst hochgradig komplexe Aufgabenstellungen mit einem vertretbaren Zeitaufwand bewältigen.

Iterationen man der Optimierung zugesteht.

Bereits hier zeigt sich, dass die Begriffe „Optimum“ und „Optimierung“ in diesem Kontext genau genommen irreführend sind. „Optimum“ heißt „das Beste“ und würde im gegebenen Beispiel die theoretisch beste oder ideale Lösung bedeuten. Eine computerbasierte Optimierung technischer oder ökonomischer Fragestellungen kann aber aufgrund des meist großen Suchraums nicht den Anspruch haben, „die theoretisch beste“ Lösung zu finden. Vielmehr muss man sich mit der besten der verfügbaren Lösungen unter den gegebenen Randbedingungen bescheiden. Dabei spielen die verfügbare Rechenkapazität und die Laufzeit des Projektes sicherlich die größte Rolle.

Wie gut sich eine einmal gefundene Lösung in der Realität bewährt, hängt entscheidend von der Qualität der Bewertungsfunktion ab. Hätte man die Bewertungsfunktion in Abbildung 1 nicht einfach nur Pixel vergleichen lassen, sondern ihr Kenntnisse über Dreiecke und das Zielbild eingebaut, wäre man womöglich mit erheblich weniger Iterationen und Bewertungen ausgekommen – zum Preis von mehr Arbeit bei der Formulierung der Bewertungsfunktion.

Aber selbst wenn die Bewertungsfunktion dieselbe bleibt, können Unsicherheiten bei anderen Vorgängen existieren. Man müsste etwa die Simulation der Verbrennungsvorgänge im Motor ihrerseits wieder an gemessenen Daten kalibrieren, und Messwerte können statistische (Rauschen) oder systematische Fehler aufweisen.

Nebenbei bemerkt ist hier weiteres Potenzial für technische Optimierungen zu finden: die Anpassung von Computersimulationen an reale Daten. Ein Optimierungsalgorithmus würde in diesem Fall vermutlich nicht den Code der Simulation selbst anpassen. Jedoch beruht eine Simulation oft auf empirisch ermittelten Konstanten. Sie sind meist relativ genau, aber vielleicht nicht genau genug. Die globale, gleichzeitige Anpassung aller dieser Parameter anhand gemessener Daten könnte der Simulation noch einmal zu einer erheblich größeren Realitätsnähe verhelfen. Ein theoretisches Beispiel wäre eine Wettersimulation, die man anhand der Aufzeichnungen der letzten hundert Jahre das Wetter der letzten Woche „vorhersagen“ lässt.

Ordnung ins Chaos bringen

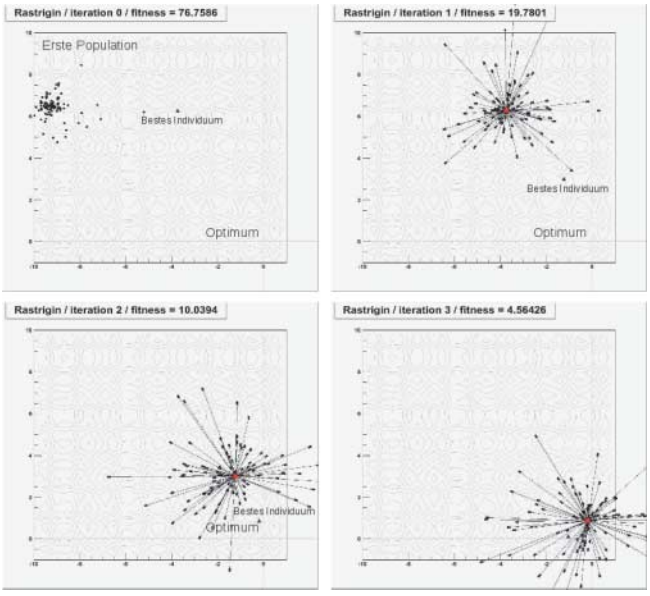
Bleibt die Frage, welche Informationen einem Optimierungsalgorithmus überhaupt zugänglich sind, um solch komplexe Aufgaben bewältigen zu können. Hier wird es wieder ganz einfach. Denn letztendlich kann man, zumindest bei Aufgaben mit einem einfachen Zielkriterium, die Bewertungsfunktion als „Black Box“ betrachten.

Vom Anwender erhält der Algorithmus Anweisungen in Form eines Computerprogramms, wie er einem gegebenen Parametersatz eine Bewertung zuordnet. Letztendlich passiert hier dasselbe wie beim Berechnen eines Rotationsparaboloiden in der Mathematik. Mehrere Eingabeparameter – in diesem Fall x und y – bekommen also einen Ausgabewert zugeordnet:

$$f(x,y)=x^2 + y^2$$

Es mögen zusätzliche Randbedingungen existieren; etwa, dass bestimmte Parameter nur Werte in einem gegebenen Bereich annehmen können. Letztendlich handelt es sich jedoch immer um eine Abbildung eines Eingaberaums auf einen Ausgaberaum. Und ob ein auf Gleitkommazahlen ausgerichteter Algorithmus nun nach dem Scheitelpunkt einer Parabel sucht oder nach der besten Konfiguration eines Automotors, ist aus seiner Sicht egal. Lediglich die benötigte Rechenzeit variiert.

Lässt man Aufgaben mit mehreren, konkurrierenden Zielfunktionen außen vor, könnte man Optimierung als die Suche nach Maxima oder Minima einer Oberfläche beschreiben, die durch eine Bewertungsfunktion definiert ist. Eingebürgert hat sich die For-



Evolutionsstrategien vollführen einen Zyklus von Duplikation/Rekombination, Mutation und Selektion. Das Beispiel der Minimumsuche der Rastrigin-Funktion visualisiert den Prozess (Abb. 2).

Die Geneva-Bibliothek

„Grid-ENabled EVolutionary Algorithms“, kurz Geneva, diente ursprünglich der Optimierung von Analysen der Elementarteilchenphysik – der Name ist also nicht ganz zufällig gewählt. Inzwischen ist die Bibliothek über diesen Einsatzzweck und die ursprüngliche Codebasis weit hinausgewachsen. Anders als der Name suggeriert, implementiert sie heute drei Optimierungsalgorithmen – die im Namen genannten evolutionären Algorithmen, Schwarmalgorithmen und Gradientenabstiege. Weitere Algorithmen, zuvorderst das „Simulated Annealing“ sollen folgen.

Alle implementierten Methoden sind darauf ausgelegt, neben dem für die Fehlersuche wichtigen seriellen Modus auch im Multithread-Betrieb sowie verteilt in Clustern, Grids und Clouds zu laufen. Da sie auf denselben Datenstrukturen basieren, kann man für die gegebene Aufgabenstellung einfach mehrere Algorithmen ausprobieren.

Die Ausführung in parallelen Umgebungen ist für den Anwender weitestgehend transparent. Im Multithread-Modus muss er lediglich gewisse Regeln für den Zugriff auf gemeinsame Datenstrukturen beachten, beispielsweise sollte er auf globale Variablen verzichten. Bei der Ausführung im Cluster muss die Serialisierbarkeit eigener Objekte sichergestellt sein. In der Praxis ist dies jedoch einfach zu erreichen. Wo es nicht geht, kann Geneva auf den Aufruf eines externen Programms für den Bewertungsschritt ausweichen, was allerdings Verzögerungen durch das wiederholte Nachladen von der Platte mit sich bringt.

Geneva ist in C++ implementiert und wird unter Linux entwickelt. Da sich ihr Zugriff auf externe Bibliotheken auf die Boost-Sammlung beschränkt, steht einer Portierung auf andere Systeme, insbesondere Windows, kaum etwas im Weg [d]. Geneva ist momentan in der Version 0.84 unter der Affero GPL v3 von www.launchpad.net/geneva frei verfügbar. Die Version 1.0, einschließlich kompletter Dokumentation, ist für Anfang 2011 geplant.

mulierung des Minimum. In dieser Konvention bedeutet eine hohe Fitness oder Qualität paradoxerweise einen niedrigen Wert der Bewertungsfunktion.

Man könnte versucht sein, ein klassisches Verfahren der Extremwertsuche zu verwenden. Dazu muss man aber das Problem in Form differenzierbarer mathematischer Funktionen ausdrücken können, etwa eines Sinus oder einer Exponentialfunktion. Die Bewertung realer Fragestellungen hingegen kann meist nur ein Computerprogramm leisten – und eine *for*-Schleife oder eine *if()*-Abfrage ist eben keine mathematische Funktion und deshalb nicht differenzierbar.

Ist eine Bewertungsfunktion gefunden, muss sich der Optimierungsalgorithmen darauf beschränken, den Parameterraum ausgehend von bekannten „guten" Lösungen mit dieser Funktion gezielt abzutasten – wie mit einem Blindenstock. Hierzu existieren unterschiedliche Strategien, die sich allerdings an mathematischen Vorbildern orientieren können.

Abstieg der Gradienten

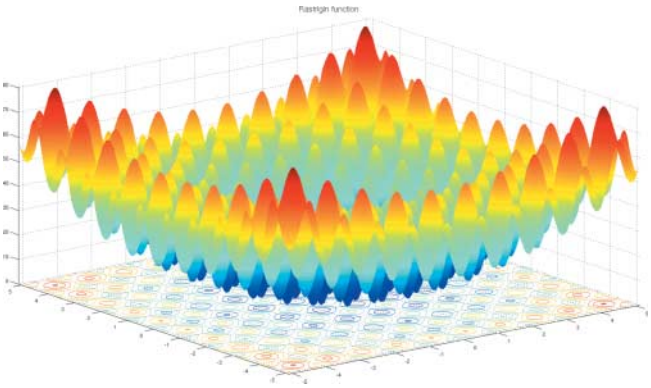
Einer der einfachsten Algorithmen ist der Gradientenabstieg. Seine Arbeitsweise lässt sich mit der Art und Weise vergleichen, mit der Wasser zu Tal fließt – immer bestrebt, dem Weg des steilsten Abfalls zu folgen. Der Algorithmus funktioniert ähnlich: Ausgehend von einem Startpunkt auf der Qualitätsoberfläche lässt sich berechnen, in welcher Richtung das „Gelände" am stärksten abfällt. Wäre die Bewertungsfunktion in mathematische Formeln fassbar, würde man einfach den Gradientenvektor der Funktion bemühen, also gewissermaßen einen unendlich kleinen Schritt in Richtung der Koordinatenachsen jedes Parameters gehen und schauen, wie weit es bergab geht.

Da im Allgemeinen jedoch keine differenzierbare Bewertungsfunktion existiert, ersetzt man den unendlich kleinen Schritt einfach durch einen besonders kleinen Schritt. Mathematisch spricht man vom Differenzenquotienten. Liegt dieser für alle Parameter vor, geht man einfach einen Schritt in die so berechnete – wegen des endlichen Schritts ungefähre – Richtung des steilsten Abfalls der Bewertungsfunktion. Am neuen Startpunkt führt man denselben Prozess durch, bis man ein zufriedenstellendes Optimum erreicht oder der Algorithmus stagniert.

Und damit stößt man schon auf die ersten Hürden: Das Bergabgehen funktioniert nur, solange man nicht in ein Tal kommt. Man spricht von lokalen Optima; gesucht ist aber das globale Optimum – die ideale oder theoretisch beste Lösung. Das Wasser würde in einem Tal – gewissermaßen einem geografischen lokalen Optimum – einen See bilden und letztendlich über die Ränder hinausfließen, bis es schließlich zum globalen Optimum, dem Meer, kommt. Der Algorithmus kann das nicht. Zwar kann er Glück haben und mit Schritten einer bestimmten Länge aus kleinen Tälern wieder herausspringen. Wahrscheinlicher ist aber, dass er endlos zwischen Punkten nahe an der Talsole hin- und herspringt – der Algorithmus stagniert.

Ein weiterer Nachteil kommt erst bei vieldimensionalen Aufgabenstellungen zum Tragen: Für jeden Schritt des Algorithmus sind *N+1* Berechnungen notwendig, wobei *N* die Zahl der Parameter ist. Man könnte also die Situation antreffen, dass man für 10 Parameter noch in akzeptabler Zeit eine Lösung erhält, für 100 aber vielleicht nicht mehr.

Unangenehm ist darüber hinaus: Da der Algorithmus ja nur die Teile der Bewertungsfunktion kennt, die er bislang berechnet hat, ist nie sicher, ob man das globale Optimum erreicht hat. Denn auch hier würde der Algorithmus konvergieren, wenn er denn so weit käme. Diese Unsicherheit ist aber allen in diesem Artikel besprochenen Optimierungsalgorithmen gemein.



Als nicht-konvexe und nicht-lineare multimodale Funktion mit einem großen Suchraum und vielen lokalen Minima eignet sich die Rastrigin-Funktion besonders als Performancetest für Optimierungsalgorithmen (Abb. 3).

Strategien der Evolution

Man stelle sich nun einen Wanderer vor, der in dunkler Nacht aus einem tiefen Tal herausfinden muss. Als passionierter MacGyver-Fan baut er sich aus seiner Thermoskanne eine Kanone und schießt wahllos um sich. Unwahrscheinlicherweise sind die Kugeln so modern, dass sie ihm nach der Landung mitteilen, wie hoch respektive tief sie liegen. Sobald er seine Kugeln verschossen hat, sucht er die Kugel, die am tiefsten Ort liegt und beginnt den Prozess von vorne.

Ganz ähnlich funktionieren die von Ingo Rechenberg gemeinsam mit Paul Schwefel bereits ab Mitte der 1960er-Jahre entwickelten Evolutionsstrategien. Sie gehen von einer Reihe bekannter Lösungen aus: den „Eltern". Hinter diesem abstrakten Begriff verbirgt sich eigentlich nur eine Menge von Zahlen, im einfachsten Fall ein Array.

Aus den Eltern erzeugt der Algorithmus mit verschiedenen Mechanismen Kandidatenlösungen: die „Kinder". Die wünschenswerte Zahl der Kindindividuen hängt dabei von der jeweiligen Problemstellung ab. In der Praxis kommt man mit 100 Individuen aber oft gut zurecht. Der Algorithmus lässt sich damit gut auf die vorhandenen Ressourcen – etwa die Zahl der Rechenknoten in einem Cluster – anpassen.

Gemeinsam bilden Eltern und Kinder eine „Population". Im einfachsten – und häufig vorkommenden – Fall erzeugt der Algorithmus die Kinder zunächst als identische Kopien eines einzelnen Elternteils, auf deren Parameter er Zufallszahlen mit einer Gauß-Verteilung addiert (Mutation). Das führt dazu, dass Kandidatenlösungen oder Kinder mit hoher Wahrscheinlichkeit in der Nähe ihrer Eltern liegen, mit niedrigerer Wahrscheinlichkeit aber auch weiter entfernt „landen". Dies würde es obigem Wanderer ermöglichen, aus breiten Tälern irgendwann hinauszuschießen, wenn er den Prozess nur lange genug wiederholt. Hierin liegt ein wichtiger Vorteil: Evolutionsstrategien neigen weit weniger dazu, in lokalen Optima hängenzubleiben, als Gradientenabstiege.

Andere, naturnähere Methoden der Kinderzeugung sind denkbar: Man könnte etwa die Feature-Vektoren zweier Elternteile überkreuzen, beispielsweise ein Kind aus der ersten Hälfte des ersten Elternteils und der zweiten Hälfte des zweiten zusammensetzen (Rekombination). Zum Schluss gilt es, die neu erzeugten Kinder zu bewerten.

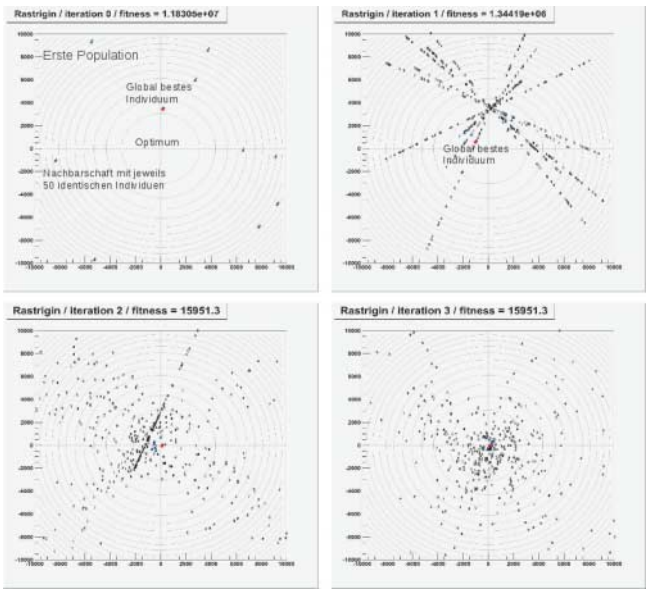
Je nach Ausprägung wählt man anhand ihrer Bewertung nur aus der Menge der Kinder oder aus der gesamten Population neue Eltern aus (Selektion). Im ersten Fall kann sich die Qualität wieder verringern – der Wanderer würde höher stehen als zu Beginn des Prozesses, kann jedoch einen weiteren Bereich erkunden. Die zweite Variante garantiert, dass die Qualität – die sogenannte Fitness – in einem Zyklus, das heißt in einer „Generation", nie sinkt. Allerdings ist sie deutlich anfälliger für lokale Optima, im Beispiel etwa für einen Graben vor einer Steilwand.

Mutationen in alle Richtungen

Wählt man die neuen Eltern nur aus der Gruppe der Kinder, könnte der Algorithmus theoretisch sogar divergieren. In der Praxis ist dies nach Erfahrung des Autors jedoch kein Nachteil, solange die Zahl der Kindindividuen groß genug bleibt. In diesem Fall verlagert sich die Optimierung immer wieder auf ein neues Optimum. In der Praxis erscheint deshalb die Wahl der Eltern aus der Menge der Kindindividuen wegen der besseren Toleranz gegenüber lokalen Optima meist als die bessere Wahl. Auch die Annäherung an das *iX*-Logo in Abbildung 1 verwendete diese Methode. In der Fitnesskurve lassen sich klar kurze Perioden erkennen, in denen sich die Fitness geringfügig verschlechterte.

Hat man neue Eltern gewählt, beginnt der Prozess von vorn und mit ihm ein neuer Zyklus aus Duplikation/Rekombination, Mutation und Selektion – ganz ähnlich wie beim Namensgeber des Verfahrens, der Evolution.

Abbildung 2 demonstriert das Verfahren anhand der Minimumsuche der Rastrigin-Funktion – eine mathematische Funktion mit besonders vielen lokalen Minima (siehe Abbildung 3). Ein Gradientenabstieg wäre kaum in der Lage, mit dieser Funktion umzugehen. Der Evolutionszyklus ist klar zu erkennen, insbesondere, wie das beste Individuum einer Generation eine neue Population begründet. Die Kindindividuen liegen in einer Art Punktwolke um das Elternteil herum. Sie sind durch das Addieren von Gauß-verteilten Zufallszahlen auf seinen Feature-Vektor – in diesem Fall zwei Gleitkommazahlen mit doppelter Genauigkeit (Double) – entstanden.



Schwarmalgorithmen ahmen das Verhalten von Tierschwärmen nach, etwa von Ameisen auf der Futtersuche. Gezeigt ist hier ebenfalls die Suche nach dem Minimum der Rastrigin-Funktion, aber in einem erheblich größeren Suchraum (-10000:10000) als bei den Evolutionsstrategien (Abb. 4).

Eine ganz ähnliche Strategie verfolgen übrigens genetische Algorithmen. Anders als Evolutionsstrategien sind ihre Feature-Vektoren jedoch aus booleschen Werten aufgebaut. Zwar hindert einen nichts daran, aus diesen eine Gleitkommazahl aufzubauen und genetische Algorithmen auf ähnliche Aufgabenstellungen wie die Evolutionsstrategien anzuwenden. Schließlich bestehen Gleitkommazahlen auch nur aus Bits. Bei genauerem Hinsehen stellt man jedoch schnell fest, dass eine kleine Änderung – der Flip eines Bits – eine große Wirkung haben kann. Im Extremfall springt man von einem Ende des erlaubten Wertebereichs zum anderen.

Typische Verhaltensweisen

Um unerwünschte Mutationen zu beschränken, muss man in genetische Algorithmen für Gleitkommazahlen eine erhebliche Menge Logik einbauen. Das führt dazu, dass Evolutionsstrategien und genetische Algorithmen auf tendenziell andere Aufgabenstellungen anwendbar sind. Umgekehrt gibt es Probleme, die sich mit Evolutionsstrategien nur schwer repräsentieren lassen. Beiden gemein ist aber der Evolutionszyklus.

Wie im echten Leben kann man mit evolutionären Algorithmen auch eine Metaoptimierung durchführen, etwa, indem man ganze Populationen miteinander konkurrieren lässt. Das Bewertungskriterium ist dann die jeweils in einer Zahl von Zyklen gefundene beste Lösung jeder Subpopulation. Auf diese Weise kann man mehrere Lösungswege parallel beschreiten und gegeneinander abwägen – zum Preis eines erheblich höheren Rechenaufwands.

Evolutionäre Algorithmen – eine Obermenge, zu der Evolutionsstrategien und genetische Algorithmen gehören – verfügen zum Leidwesen der Anwender über relativ viele Steuerparameter, deren optimale Werte stark von der jeweiligen Aufgabe abhängen. Deshalb beinhaltet Metaoptimierung auch den Versuch, diese Parameter in einem eigenen Optimierungszyklus zu bestimmen. Dessen Ziel könnte etwa darin bestehen, den Satz Steuerparameter zu finden, der in einer vorgegebenen Zahl von Zyklen das beste Optimierungsergebnis erzielt. Üblicherweise vereinfacht man für eine solche Optimierung die eigentliche Aufgabe, etwa indem man weniger Eingabedaten verwendet.

Bei den meisten Optimierungen mit evolutionären Algorithmen treten die größten Fortschritte bereits zu Beginn der Optimierung auf. Ist Rechenzeit Mangelware, kann man auf diese Art nach der altbekannten 80/20-Regel 80 % oder mehr des Ergebnisses mit 20 % der Rechenzeit erreichen. Das „80/20-Verhalten" lässt sich auch in den Beispielen der Abbildungen 1 und 2 beobachten. Für viele Aufgabenstellungen wird das ein akzeptabler Kompromiss sein.

Im Schwarm unterwegs

Ein modernerer Algorithmus mit weniger Eingabeparametern, genannt Particle Swarm Optimization, orientiert sich am Schwarmverhalten, wie es viele Tiere, etwa Ameisen, an den Tag legen. Wie im Fall der Evolutionsstrategien und Gradientenabstiege verwendet man üblicherweise Gleitkommazahlen für die Repräsentation der Parameter. Der Algorithmus geht von Gruppen von Individuen aus, die sich in „Nachbarschaften" zusam-

Onlinequellen

[a]	weitere Informationen zur Geneva-Bibliothek	www.gemfony.com
[b]	Quellcode der Geneva-Bibliothek	www.launchpad.net/geneva
[c]	das Amdahl'sche Gesetz	de.wikipedia.org/wiki/Amdahls_Gesetz
[d]	die Boost-Bibliothekssammlung	www.boost.org

menfinden. Jede Nachbarschaft besitzt ein „bestes" Individuum, zusätzlich existiert zu jedem Zeitpunkt eine global beste Lösung.

In jeder Iteration passt der Algorithmus die Positionen aller Kandidatenlösungen („Partikel") an. Hierzu berechnet er für jedes einzelne zunächst die Richtungen *G* und *L* zum global und lokal besten Partikel und multipliziert sie mit einer Zufallszahl. Ferner bestimmt er einen „Velocity"-Vektor, in den *G* und *L* der letzten Iteration und optional eine weitere Zufallskomponente eingehen. Aus der Summe dieser drei Vektoren ergibt sich der Schritt, den die Kandidatenlösung in einer Iteration vollführen muss.

Diese Methode zieht die Partikel so lange in Richtung der global und lokal besten Lösungen, bis jeweils andere diese ersetzen. Nach und nach konvergiert das ganze Ensemble an einem einzelnen Ort. Das Verfahren ähnelt damit dem, was man bei Ameisenschwärmen bei der Futtersuche beobachtet.

Kann man bei Evolutionsstrategien noch eine gewisse Nähe zu Gradientenverfahren erahnen, so ist diese Verwandtschaft bei Schwarmalgorithmen nicht mehr zu erkennen. Jedoch zeigt ihr Erfolg, dass diese Form der Optimierung trotz mangelnder Anschaulichkeit zu Recht einen wichtigen Platz einnimmt.

Im Vergleich zu Evolutionsstrategien beobachtet man zu Beginn der Optimierung manchmal eine Art „Einschwingverhalten". Bei komplexeren Problemen sucht der Algorithmus zunächst eine Weile bei kaum zunehmender Qualität, bevor er rapide auf sein Ziel hin konvergiert.

Abbildung 4 illustriert Schwarmalgorithmen erneut am Beispiel der Rastrigin-Funktion, allerdings in einem sehr viel größeren Suchraum (-10000:10000) als im Fall der Evolutionsstrategien. In den Höhenlinien der Funktion sind die lokalen Optima deshalb nicht mehr sichtbar, für den Algorithmus aber gleichwohl vorhanden.

Einen Kern für jedes Kind

Ein ausgesprochen angenehmer Aspekt besonders der evolutionären und der Schwarmalgorithmen liegt in der vergleichsweise einfachen Parallelisierbarkeit. In den meisten Aufgabenstellungen lassen sich die verschiedenen Kandidatenlösungen unabhängig voneinander bewerten. Dies erlaubt es, die Bewertungsphase und im Falle der evolutionären Algorithmen zudem die Mutation parallel durchzuführen, entweder in einzelnen Threads oder verteilt auf mehrere Rechner im Cluster, im Grid oder in der Cloud. Auch die Verwendung einer GPU (Graphics Processing Unit) ist denkbar. Die Abbildungen 2 und 4 sollten das Potenzial der Parallelisierung für diese Algorithmen noch einmal verdeutlichen.

Eine Parallelisierung auf verteilten Systemen, etwa einem Cluster, ergibt allerdings nur dann einen Sinn, wenn die Bewertungsphase im Vergleich zum eigentlichen Optimierungsalgorithmus hinreichend lange dauert. Tests mit der Geneva-Bibliothek haben ergeben, dass bei einer Aufgabe mit 1000 Parametern pro Individuum, die zusammen mit den Statusinformationen der Bibliothek jeweils etwa 100 KByte füllen, die Bewertungsfunktion rund 20 s lang rechnen muss, um bei einer Ausführung im Cluster dem theoretisch maximalen Geschwindigkeitszuwachs nahezukommen: Da für die Bewertung jeder Kandidatenlösung ein eigener Rechner zur Verfügung stand, musste die Berechnung um die Zahl der Kindindividuen – im Falle der evolutionären Algorithmen – schneller sein als auf einem einzelnen CPU-Kern. Bei der Ausführung auf Mehrkernprozessoren im Multithread-Modus tritt die Beschleunigung hingegen schon bei kurzen Bewertungsphasen ein. In engem Zusammenhang hiermit stehen die Gesetzmäßigkeiten, die das Amdahl'sche Gesetz [c] beschreibt.

Die angenommenen 7,6 Jahre Rechenzeit für 20 000 Iterationen (bei einer Minute Rechenzeit für eine Bewertung) aus dem Autobeispiel zu Beginn ließen sich enorm reduzieren, wenn man die Kindindividuen auf jeweils einem eigenen Rechenknoten evaluieren ließe. Außerdem: Da man wie gezeigt oft nach wenigen Iterationen signifikante Qualitätsverbesserungen erhält, kommt man vielleicht mit einigen Hundert Zyklen statt der 20 000 aus.

Messungen des Autors deuten übrigens darauf hin, dass der größte Overhead im Cluster im Fall der Geneva-Bibliothek nicht vom Netztransfer stammt, sondern vom Prozess der (De-)Serialisierung von Objekten auf Client- und Serverseite. Dies ist insofern angenehm, als die Programmierer nur die Lauf- und Latenzzeiten im Netz als Konstante in Kauf nehmen müssen. Den Prozess der Serialisierung hingegen können sie im Laufe der Weiterentwicklung von Geneva weiter optimieren.

Hinsichtlich der Fehlertoleranz verhalten sich Evolutionsstrategien und Schwarmalgorithmen unkritisch. Bleiben die Antworten einzelner Rechnerknoten etwa in einem weltweit verteilten Grid aus, spielt das keine große Rolle.

Zum Schluss

Mit Optimierungsalgorithmen kann man Antworten auf komplexe technische Fragestellungen finden, die selbst dem Fachmann nicht ohne Weiteres zugänglich sind. Hierbei kommen sie mit einem Bruchteil der – absolut unrealistischen – Rechenzeit aus, die man für das vollständige Durchsuchen des Parameterraums benötigen würde. Durch ihre einfache Parallelisierbarkeit auf Mehrkernebene oder in den heute verfügbaren Clustern, Grids und Clouds lässt sich die benötigte Zeit noch einmal erheblich reduzieren. Dadurch, dass durch Grids und Clouds enorme Rechenressourcen quasi auf Zuruf zur Verfügung stehen, werden bald auch Aufgabenstellungen der Optimierung zugänglich, die vormals zu komplex waren. Zudem steht mit der Geneva-Bibliothek eine Open-Source-Lösung bereit, mit der man solche Aufgaben angehen kann. Und wie im täglichen Leben kommt auch bei der Optimierung die beste Lösung immer zum Schluss. (sun)

DR. RÜDIGER BERLICH

forscht am Steinbuch Centre for Computing des Karlsruhe Institute of Technology im Bereich Grid und Cloud Computing. Er beschäftigt sich seit 2001 mit dem verteilten Rechnen und ist Hauptautor der Geneva-Bibliothek.

www.ix.de/ix1012110