Evolutionäre Optimierung dynamischer Probleme

Jürgen Branke

Institut AIFB
Universität Karlsruhe
http://www.aifb.uni-karlsruhe.de/~jbr

Viele praxisrelevante Optimierungspsrobleme sind dynamisch, d.h. sie verändern sich im Zeitverlauf. In diesen Fällen reicht es nicht mehr aus, möglichst schnell und zuverlässig ein Optimum zu finden. Es geht vielmehr darum, trotz der Dynamik des Problems fortlaufend Lösungen hoher Qualität anbieten zu können.
In der hier diskutierten Arbeit [Bra00] wird gezeigt, wie Evolutionäre Algorithmen erfolgreich an eine solche Aufgabenstellung angepasst werden können.

1 Einführung

Evolutionäre Algorithmen (EAs) sind iterative stochastische Optimierungsheuristiken, deren Funktionsweise der natürlichen Evolution nachempfunden ist: aus einer Menge potentieller Lösungen (Population) werden die besseren ausgewählt um daraus neue Varianten (Nachkommen) zu erzeugen. Dabei werden Teile der ausgewählten Lösungen neu zusammengestellt (Crossover) und leicht verändert (Mutation). Die Nachkommen können dann auf ihre Güte (Fitness) hin untersucht werden und ersetzen i.a. die schlechteren Lösungen in der Population. Der Grundzyklus (Generation) eines evolutionären Algorithmus ist in Abbildung 1 dargestellt. Eine weitergehende Darstellung Evolutionärer Algorithmen würde den Rahmen dieses Artikels sprengen, der interessierte Leser wird daher auf einschlägige Literatur verwiesen ([Gol89, Mic96, Bäc96]).
In den vergangenen Jahren wurden evolutionäre Algorithmen erfolgreich auf eine breite Vielfalt von Optimierungsproblemen angewendet. Fast alle diese Probleme sind allerdings statisch, wohingegen die Optimierungsprobleme in der Praxis oft dynamisch und stochastisch sind, d.h. sich im Zeitverlauf verändern. Beispiele wären in der Produktionplanung, wenn fortlaufend ankommende neue Aufträge in den bestehenden Produktionsplan eingebunden werden müssen, in der Tourenplanung, wenn neue Kunden integriert werden müssen, bei der Robotersteuerung, wenn die Umwelt des Roboters nicht statisch ist, oder in der Fertigung, wenn Fertigungstoleranzen und Maschinenabnutzung zu einer variierenden Qualität des Endprodukts führen.
Da die natürliche Evolution in einer dynamischen Umwelt abläuft liegt es nahe zu vermuten, dass auch evolutionäre Algorithmen in einer dynamischen Umwelt erfolgreich sein können. Dies ist jedoch für die Standard-Varianten nur sehr bedingt der Fall.
In der hier zusammengefassten Arbeit [Bra00] wird gezeigt, wie sich evolutionäre Algorithmen auf die Anforderungen dynamischer Optimierungsprobleme anpassen lassen. Dabei werden vier Fälle separat betrachtet:

1. Das Optimierungsproblem ändert sich, und es ist das Ziel, mit der Lösung dem

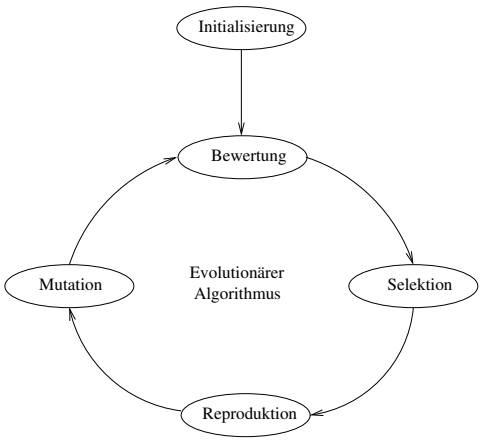

Abbildung 1: Ablaufschema eines evolutionären Algorithmus.

wandernden Optimum so gut wie möglich zu folgen.

2. Bei der Anpassung einer Lösung entstehen nicht vernachlässigbare Änderungskosten. In diesem Fall sind die Änderungskosten als zusätzliches Optimierungskriterium zu berücksichtigen.

3. Die Lösung, einmal implementiert, kann nicht mehr angepasst werden. Dann ist es wichtig, von vornherein nach einer Lösung zu suchen, die trotz zu erwartender Änderungen des Problems eine gute mittlere Lösungsqualität besitzt.

4. Wenn die Lösung auf das Problem zurückwirkt, dann wird es notwendig, nach einer Lösung zu suchen, die nicht nur gut sondern auch flexibel ist bzw. den Problemzustand so beeinflusst, dass gut auf Änderungen reagiert werden kann.

Die wichtigsten Resultate aus [Bra00] zu diesen vier Aspekten werden in den folgenden Abschnitten 2 bis 5 nacheinander dargestellt, wobei auf die letzten beiden Punkte aus Platzgründen nur sehr kurz eingegangen werden kann. Der Artikel schließt mit einer Zusammenfassung und einem Ausblick in Abschnitt 6.

2 Dem Optimum folgen

Eine Möglichkeit, nach einer Änderung des Problems das neue Optimum zu suchen, wäre, einfach den evolutionären Algorithmus neu zu starten. Das ist jedoch offensichtlich sehr rechenaufwändig, denn der Algorithmus braucht eine Weile, um in die relevante Region des Suchraums vorzustoßen. Wenn man davon ausgeht, dass die Änderungen des Problems

relativ klein sind, ist es naheliegend, zu versuchen, durch die Ausnutzung von beim Optimierungsprozess gewonnener Information über den alten Suchraum die Suche nach dem neuen Optimum zu beschleunigen.
Anstatt eines Neustarts könnte man also etwa einfach die alte Population unverändert übernehmen und nur die Qualitätsfunktion anpassen. Dies ist jedoch auch wenig hilfreich, da evolutionäre Algorithmen *konvergieren*, d.h. im Laufe der Optimierung sich auf einen kleinen Bereich um die bisher beste gefundene Lösung konzentrieren und danach kaum mehr in der Lage sind, ein neues Optimum am anderen Ende des Suchraums zu finden.
Dieses Problem wurde auch in der Literatur erkannt, und es gibt im wesentlichen drei Vorschläge, wie dies zu lösen sei:

- Nach einer Änderung explizit die Vielfalt in der Population erhöhen (z.B. [Cob90, VFJ96]). Dabei wird allerdings ein Großteil der vorhandenen Information durch zufällige ersetzt.

- Von Vornherein die Vielfalt in der Population erhalten (etwa [Gre99, MKN96]). Dies behindert allerdings durch den Fokus auf Vielfalterhaltung den eigentlichen Optimierungsprozess.

- den EA mit einem Gedächtnis ausstatten (vgl. [GS87, LHR98, Bra99]. Naheliegenderweise ist dieser Ansatz besonders dann hilfreich, wenn alte Problemzustände häufig wiederkehren, in welchem Fall der EA einfach eine alte Lösung "aus dem Gedächtnis zaubern" kann. In anderen Fällen hilft das Gedächntis wenig, und das Problem der Konvergenz wird damit ebenfalls nicht gelöst [Bra99].

In meiner Dissertion [Bra00] habe ich deswegen einen neuen Ansatz entwickelt, die "Self-Organizing Scouts"(SOS), der im folgenden näher beschrieben wird.
Die Grundidee ist relativ einfach: Zunächst einmal soll der Algorithmus ganz uneingeschränkt nach einer möglichst guten Lösung suchen. Hat er eine gefunden, dann werden die Population und der Suchraum aufgespalten (*Forking*): Einige wenige Individuen (die Scout-Population) werden in der Nachbarschaft der gefundenen Lösung abgesetzt mit dem Ziel, die Lösung zu beobachten, ihr falls notwendig zu folgen, und die Lösung evtl. noch zu verfeinern.
Der Rest der Population (die sogenannte Basis-Population) kann sich dann auf die Suche nach neuen Lösungen machen und wird explizit von der Nachbarschaft der bereits gefundenen Lösung ausgeschlossen.
Dieser Prozess des Abspaltens von Teilpopulationen und Teil-Suchräumen ist veranschaulicht in Abbildung 2. So wird eine Lösung nach der anderen entdeckt und jeweils durch eine Scout-Population "bewacht". Das Zentrum der Scout-Populations-Suchräume ist jeweils durch das beste dort vorhandene Individuum definiert. Findet die Scout-Population eine neue beste Lösung (z.B. weil sich das Optimum verschoben hat), so wird auch der Suchraum entsprechend angepasst.
Insgesamt unterhält der SOS-Algorithmus also Individuen (und damit Information) in einer ganzen Reihe vielversprechender Gebiete des Suchraums. Durch die kontinuierliche

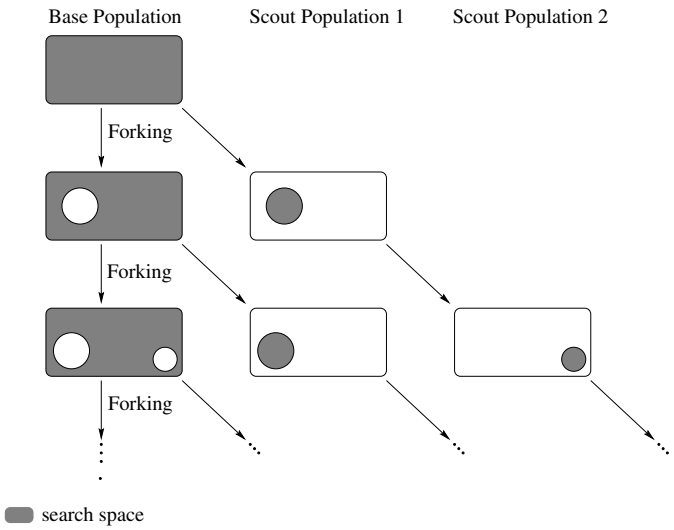

Abbildung 2: Wenn ein neues lokales Optimum gefunden wurde, wird der Suchraum aufgeteilt. Eine Scout-Population sucht speziell im Bereich des neuen lokalen Optimums, während die restlichen Individuen von diesem Teil des Suchraums ausgeschlossen werden.

Überwachung durch die Scout-Populationen wird diese Information ständig aktuell gehalten. Außerdem wird duch das Fernhalten der Basis-Population von bereits gefundenen vielversprechenden Bereichen eine ständige Suche nach neuen Optima erzwungen. Der SOS-Algorithmus kann damit sehr gut auf folgende Änderungen reagieren:

1. Das Optimum wandert langsam. In diesem Fall folgt die Scout-Population dem Optimum einfach und zieht ihren Suchraum nach sich.

2. Das Optimum springt von einer Stelle zu einer anderen Stelle, die zwar bisher auch schon gut, aber eben nicht optimal war. In diesem Fall war mit relativ hoher Wahrscheinlichkeit bereits eine andere Scout-Population auf dem bisher schon guten Bereich des Suchraums, die dort dann sofort das Verfolgen des Optimums übernehmen kann.

3. Das Optimum springt an eine ganz andere Stelle, die bisher nicht sonderlich vielversprechend war. Dies ist sicher der schwierigste Fall. Da allerdings die Basis-Population fortlaufend in neuen Bereichen nach Optima suchen muss bleibt zu hoffen, dass sie auch dieses neue Gebiet bald entdecken wird. Wenn die Problemänderungen moderat sind sollte dieser Fall auch nicht zu oft auftreten, andernfalls ist ein Neustart des Algorithmus vermutlich die beste Lösung, da offensichtlich die vom alten Suchraum übertragene Information wertlos ist.

Empirische Tests haben gezeigt ([Bra00]), dass SOS anderen getesteten Verfahren bei einer breiten Auswahl von Problemcharakteristiken (Anzahl der lokalen Optima, Ände-

rungsfrequenz, Wanderungsgeschwindigkeit der lokalen Optima, Anzahl der Dimensionen) signifikant überlegen ist.

3 Berücksichtigung von Änderungskosten

In der Praxis ist die Anpassung einer Lösung an sich ändernde Rahmenbedingungen i.a. mit Änderungskosten verbunden. In diesem Fall könnte es günstiger sein, eine nicht ganz so gute aber mit geringen Änderungskosten verbundene Lösung zu wählen, als zu versuchen, zur neuen, mit sehr großen Änderungskosten verbundenen Optimallösung zu wechseln. Änderungskosten sollten also bei der Nachoptimierung berücksichtigt werden, das Optimierungsproblem wird damit um ein zusätzliches Kriterium, die Änderungskosten, erweitert.
Die Zusammenführung der Zielkriterien zu einer gewichteten Kombination ist zwar grundsätzlich möglich, meist ist es aber sehr schwierig eine angemessene Gewichtung festzulegen bevor die Alternativen bekannt sind.
Als populationsbasierte Verfahren bieten evolutionäre Algorithmen hier eine sehr vielversprechende Alternative: sie erlauben es, eine ganze Menge Pareto-optimaler[1] Lösungen gleichzeitig zu entwickeln, aus denen der Nutzer dann anschließend auswählen kann (eine Übersicht über derartige Ansätze findet sich z.B. in [Deb01]). Die wesentliche Änderung im Vergleich zum unikriteriellen evolutionären Algorithmus ist dabei, als Fitness eine Pareto-Rangfolge zu verwenden: alle nicht dominierten[2] Individuen erhalten Rang 1, alle nur von diesen dominierten Individuen erhalten Rang 2 etc.

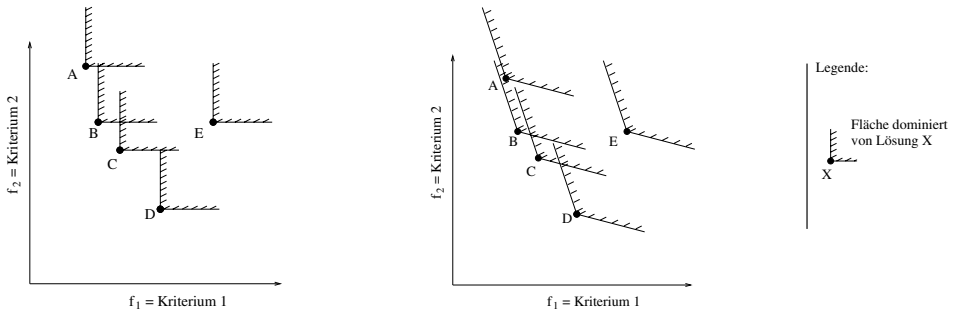

Abbildung 3: Das Konzept der Dominanz für zu minimierende Zielkriterien, im Standard-Fall (links) und bei GMOEA (rechts).

Eine solch breite Suche ist jedoch naturgemäß auch vergleichsweise aufwändig. Zudem hat ein Nutzer meist durchaus eine wenn auch vage Vorstellung seiner Präferenzen. In

[1] Eine Lösung heißt Pareto-optimal, wenn sie im Vergleich zu jeder anderen Lösung in mindestens einem Kriterium besser ist.
[2] Ein Individuum A wird von einem anderen Individuum B *dominiert*, wenn B in allen Kriterien mindestens gleich gut und in mindestens einem Kriterium besser ist als A, vgl. Abbildung 3

[Bra00] wurde deswegen ein Verfahren vorgeschlagen, diese vagen Benutzerpräferenzen zu nutzen, um die Suche so weit wie möglich auf den vom Benutzer als interessant spezifizierten Bereich einzuschränken.
Das dazu vorgeschlagene Verfahren GMOEA (für "Guided Multi-Objective Evolutionary Algorithm") erlaubt es dem Benutzer, maximale und minimale Trade-offs der Form "eine Einheit von Kriterium 1 ist maximal/minimal X Einheiten von Kriterium Y wert" zu spezifizieren. Anhand dieser Grenzen wird dann einfach der Dominanzbereich einer Lösung erweitert (siehe Abbildung 3).
Wie sich in ersten empirischen Untersuchungen ergeben hat, erlaubt das Verfahren tatsächlich, die Suche auf vom Benutzer als interessant spezifizierte Lösungsalternativen einzuschränken. Aufgrund der eingeschränkten Suche wird außerdem die Konvergenz beschleunigt. Abbildung 4 zeigt beispielhaft die Verteilung der Individuen in der Population in der 50. Generation zweier EA-Läufe mit zu minimierenden Zielkriterien, einmal mit "Guiding" und einmal ohne Guiding. Es ist deutlich zu sehen, wie Guiding die Suche auf den interessanten Bereich einschränkt und dabei auch schneller konvergiert.

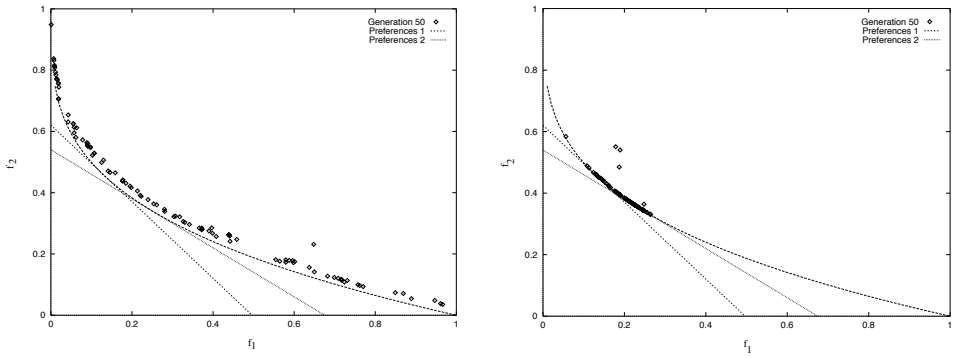

Abbildung 4: Nach 50 Generationen ist deutlich zu sehen, wie GMOEA (rechts) den ungeführten EA (links) sowohl in Bezug auf Konvergenz als auch in Bezug auf Abdeckung des interessanten Bereichs überlegen ist.

Der vorgeschlagene Algorithmus ist also eine effektive Möglichkeit, auch ungenau spezifizierte Benutzerpräferenzen in die Suche mit einzubeziehen und dadurch schneller zu einer besseren Abdeckung des interessanten Bereichs des Lösungsraums zu gelangen.

4 Robustheit

Wenn sich eine einmal gewählte Lösung nicht mehr anpassen lässt (sei es, weil die Kosten für eine ständige Anpassung zu hoch sind, oder eine Anpassung zu lange dauern würde), dann geht es darum, eine Lösung zu finden, die trotz zu erwartender Änderungen eine gute mittlere Lösungsgüte aufweist. Typisches Beispiel wären etwa Fertigungstoleranzen: Man

würde gerne eine bestimmte Lösung x produzieren, erhält aber statt dessen eine davon leicht abweichende Lösung $x + \delta$, wobei δ eine Zufallsvariable ist, deren Verteilung als bekannt angenommen wird.
Ist $f(x)$ die Qualität eines Produkts, so ist unter der Berücksichtigung von Fertigungstoleranzen nicht das Produktdesign x optimal, das $f(x)$ maximiert, sonder jenes, das die erwartete Qualität (*effektive Fitness*) maximiert. Diese könnte prinzipiell berechnet werden als:

$$f_{eff}(x) = E(f(x+\delta)) = \int_{-\infty}^{\infty} p(\delta) \cdot f(x+\delta)\, d\delta$$

Für praktische Probleme ist dieser Ausdruck jedoch i.a. nicht berechenbar, da f nicht in geschlossener Form sondern lediglich beispielsweise als Simulationsmodell vorliegt. Es müssen also andere Wege gefunden werden, f_{eff} abzuschätzen.
Die naheliegendste Möglichkeit ist sicherlich, f_{eff} als Mittelwert mehrerer Evaluierungen $f(x + \delta))$ mit zufällig gewählten δ zu berechnen (z.B. [Gre96, Ree92]). Damit wird die Fitness eine stochastische Größe. Bei einer Stichprobe der Größe n reduziert sich die Standardabweichung der geschätzten Evaluierung im Vergleich zur einfachen Evaluierung um den Faktor $\frac{1}{\sqrt{n}}$, leider erhöht sich jedoch auch der Rechenaufwand um den Faktor n.
In der hier besprochenen Dissertation [Bra00] werden einige Möglichkeiten diskutiert, die notwendige Genauigkeit bei der Fitnessbewertung auch mit weniger Rechenaufwand zu erreichen. Beispielsweise wird gezeigt,

- dass durch Verwendung von Varianzreduktionsmethoden bei der Wahl der δ-Werte signifikante Verbesserungen der Lösungsqualität erreicht werden können. Am erfolgreichsten hat sich hier das Latin Hypercube Sampling-Verfahren herausgestellt.

- dass es hilft, alle Individuen einer Generation mit den gleichen Störungen δ zu bewerten, anstatt neue δ für jedes Individuum zu ziehen.

- dass es sich lohnt, in den ersten und insbesondere in den letzten Generationen häufiger zu evaluieren, während der evolutionäre Algorithmus in der Mitte des Laufs auch mit ungenaueren Evaluierungen zurecht kommt.

- dass die Populationsgröße einen ähnlichen Effekt wie eine größere Stichprobe hat, und dass diese beiden Größen aufeinander abgestimmt sein sollten. Dies ist vermutlich auch der Grund warum das Insel-Modell, bei dem die Population in mehrere weitgehend unabhängige Subpopulationen aufgeteilt wird, bei der Suche nach robusten Lösungen vergleichsweise schlecht abschneidet.

- dass die optimale Stichprobengröße auch von der verfügbaren Zeit abhängt. Ist wenig Zeit vorhanden, sollte man die Rechenleistung lieber in eine längere Abfolge von Generationen stecken, als die Evaluierungsgenauigkeit durch große Stichproben zu erhöhen. Je mehr Zeit vorhanden ist, desto wichtiger wird die Genauigkeit.

- dass Standard-Einstellungen des evolutionären Algorithmus für deterministische Optimierungsprobleme wie Elitismus oder Steady-State-Reproduktion bei der Suche nach robusten Lösungen hinderlich sind, weil sie Bereiche des Suchraums mit hoher Varianz bevorzugen.
- dass es möglich ist, die Leistungsfähigkeit des Algorithmus zu verbessern, indem man zur Evaluierung die Fitness benachbarter, bereits evaluierter Individuen mit heranzieht. Da evolutionäre Algorithmen vielversprechende Regionen des Suchraums durch die Produktion vieler Individuen genauer untersuchen, sind in diesen Regionen meist schon viele andere Individuen evaluiert worden. Nimmt man weiter an, dass sich die effektive Fitness inerhalb einer begrenzten Entfernung nicht zu stark ändert, so stellt die Fitness benachbarter Individuen eine wichtige Informationsquelle dar.

5 Flexibilität

In vielen Anwendungen beeinflusst die implementierte Lösung die Ausgangslage für zukünftige Anpassungen. Wenn beispielsweise in der Produktionsplanung mit der Umsetzung einer Lösung begonnen wurde, dann beeinflusst diese Lösung die Ausgangslage bei der nächsten Problemänderung, etwa der Ankunft eines neuen Jobs. Das neue Optimierungsproblem besteht dann aus allen noch nicht abgearbeiteten Operationen (abhängig von der teilweise implementierten Lösung) sowie den Operationen des neuen Jobs.
In solchen Fällen ist es wichtig, neben der Lösungsqualität auch die Effekte auf den Problemzustand bei der nächsten möglichen Änderung zu berücksichtigen. Ziel muss es sein, auch weiterhin flexibel auf Änderungen reagieren zu können. Wie in [Bra00, BM00] gezeigt, wird die Flexibilität bei Job Shop Scheduling Problemen wesentlich durch Maschinenleerzeiten bestimmt. Ziel muss es also sein, Maschinenleerzeiten so lange wir möglich zu erhalten. Eine Bestrafung früher Maschinenleerzeiten (die ja bis zur nächsten Problemänderung höchstwahrscheinlich ungenutzt verstrichen wären) zusätzlich zum eigentlichen Zielkriterium führt bei dynamischen Problemen zu ganz wesentlichen Verbesserungen des eigentlichen Zielkriteriums, weil flexibler auf Änderungen reagiert werden kann.

6 Zusammenfassung und Ausblick

Dynamische, d.h. sich im Zeitverlauf ändernde Optimierungsprobleme stellen evolutionäre Algorithmen vor neue Herausforderungen.
In der hier beschriebenen Arbeit [Bra00] wurden verschiedene Möglichkeiten untersucht, evolutionäre Algorithmen für die Anwendung auf dynamische Optimierungsprobleme anzupassen. Dabei wurden vier grundlegende Aspekte betrachtet:

1. Die kontinuierliche Anpassung einer Lösung an ein sich änderndes Problem. Hier

geht es im wesentlichen darum, relevante Informationen über den Suchraum von einem Problemzustand zum nächsten zu übertragen, ohne jedoch die Suche zu stark einzuschränken. Dazu wurde der Self-Organizing-Scouts-Ansatz entwickelt, der auf jedem entdeckten lokalen Optimum eine kleine "Scout"-Population plaziert, die diesen Bereich weiter verfolgen kann, während sich die restliche Population auf die Suche nach neuen Optima machen muss. Da die Scout-Populationen in gewissen Grenzen ihrem Optimum folgen können, hat der Algorithmus quasi ein selbst-adaptives Gedächtnis über bereits entdeckte vielversprechende Bereiche des Suchraums.

2. Entstehen bei der Anpassung einer Lösung Änderungskosten, dann ist es notwendig, diese Änderungskosten bei der Suche nach einem neuen Optimum zu berücksichtigen. Hier wurde vorgeschlagen, das Problem als multikriterielles Optimierungsproblem aufzufassen und nach einer Menge Pareto-optimaler Lösungen zu suchen. Da der Nutzer jedoch meist eine gewisse Vorstellung über "sinnvolle" Lösungsalternativen hat, wurde eine EA-Variante entwickelt, die diese Vorstellungen in die Suche mit einbezieht. Dadurch ist es möglich, die Suche zu beschleunigen und den interessanten Bereich des Suchraums besser abzudecken.

3. Ist es nicht möglich, die Lösung ständig anzupassen, dann sollte man nach Lösungen suchen, die trotz möglicher Problemänderungen eine gute erwartete Qualität aufweisen. Die erwartete Qualität lässt sich grundsätzlich dadurch abschätzen, dass man für jede Lösung verschiedene Szenarien betrachtet und die mittlere Lösungsqualität über die Szenarien als Schätzer verwendet. Leider ist dieses Verfahren aber auch sehr rechenaufwändig. Es wurden deshalb in [Bra00] verschiedene Methoden entwickelt, die notwendige Anzahl an Szenarien zu verringern. Außerdem wurden einige Untersuchungen begzüglich Parametereinstellung gemacht und gezeigt, dass typische Standard-Einstellungen für EAs auf statischen Problemen nicht ohne weiteres auf dynamische Probleme übertragen werden dürfen.

4. Schließlich wurde noch der Fall angesprochen, in dem die vom EA entwickelte Lösung die weitere Entwicklung des Problems beeinflusst. Dann ist es nämlich wichtig Lösungen zu suchen, die nicht nur gut sondern auch flexibel sind, also sich bei zukünftigen Problemänderungen gut anpassen lassen. Am Beispiel von Job Shop Scheduling wurde dabei demonstriert, dass frühe Maschinenleerzeiten ein geeignetes Maß für die Nicht-Flexibilität des Systems sind, und dass durch die Vermeidung früher Maschinenleerzeiten zusätzlich zum eigentlichen Zielkriterium das eigentliche Zielkriterium im dynamischen Fall deutlich verbessert werden kann.

Insgesamt lässt sich festhalten, dass evolutionäre Algorithmen, entsprechend angepasst, in allen vier Bereichen hervorragende Ergebnisse liefern und damit eine vielversprechende Heuristik zur Lösung dynamischer Optimierungsprobleme darstellen.
Während aus Gründen der Klarheit die vier verschiedenen Aspekte in der Arbeit getrennt behandelt wurden, wäre es aus praxisrelevanter Sicht nun sicherlich interessant, die vorgeschlagenen Algorithmus-Varianten zu einem Verfahren zusammenzuführen und das Zu-

sammenspiel zu untersuchen. Desweiteren böte sich an, Vorhersage in das System mit einzubauen um die Reaktionsfähigkeit weiter zu erhöhen.

Literaturverzeichnis

[Bäc96] Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, 1996.

[BM00] Branke, J.; Mattfeld, D.: Anticipation in Dynamic Optimization: The Scheduling Case. In PPSN2000 (Schoenauer, M.; Deb, K.; Rudolph, G.; Yao, X.; Lutton, E.; Merelo, J. J.; Schwefel, H.-P., Hg.). Springer, 2000, Bd. 1917 von LNCS, S. 253–262.

[Bra99] Branke, J.: Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. In Congress on Evolutionary Computation CEC99. IEEE, 1999, Bd. 3, S. 1875–1882.

[Bra00] Branke, J.: Evolutionary Optimization in Dynamic Environments. Dissertation, Institut AIFB, Universität Karlsruhe, 76128 Karlsruhe, 2000. Published by Kluwer, 2001.

[Cob90] Cobb, H. G.: An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuouis, Time-Dependent Nonstationary Environments. Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA, 1990.

[Deb01] Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. Wiley, 2001.

[Gol89] Goldberg, D. E.: Genetic Algorithms. Addison-Wesley, 1989.

[Gre96] Greiner, H.: Robust Optical Coating Design with Evolutionary Strategies. In Applied Optics, Bd. 35 (28):(1996), S. 5477–5483.

[Gre99] Grefenstette, J. J.: Evolvability in Dynamic Fitness Landscapes: A Genetic Algorithm Approach. In Congress on Evolutionary Computation. IEEE, 1999, Bd. 3, S. 2031–2038.

[GS87] Goldberg, D. E.; Smith, R. E.: Nonstationary Function Optimization using Genetic Algorithms with Dominance and Diploidy. In Second International Conference on Genetic Algorithms (Grefenstette, J. J., Hg.). Lawrence Erlbaum Associates, 1987, S. 59–68.

[LHR98] Lewis, J.; Hart, E.; Ritchie, G.: A Comparison of Dominance Mechanisms and Simple Mutation on Non-stationary Problems. In Parallel Problem Solving from Nature (Eiben, A. E.; Bäck, T.; Schoenauer, M.; Schwefel, H.-P., Hg.). Springer, 1998, Nr. 1498 in LNCS, S. 139–148.

[Mic96] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer Verlag, 3rd Aufl., 1996.

[MKN96] Mori, N.; Kita, H.; Nishikawa, Y.: Adaptation to a Changing Environment by Means of the Thermodynamical Genetic Algorithm. In Voigt [Voi96], S. 513–522.

[Ree92] Reeves, C. R.: A Genetic Algorithm Approach to Stochastic Flowshop Sequencing. In Proceedings of the IEE Colloquium on Genetic Algorithms for Control and Systems Engineering. IEE, London, 1992, Nr. 1992/106 in Digest, S. 13/1–13/4.

[VFJ96] Vavak, F.; Fogarty, T. C.; Jukes, K.: A Genetic Algorithm with Variable Range of Local Search for Tracking Changing Environments. In Voigt [Voi96].

[Voi96] Voigt, H.-M., Hg.: Parallel Problem Solving from Nature, Nr. 1141 in LNCS. Springer Verlag Berlin, 1996.

Jürgen Branke, geboren 1969, Abitur 1988, Studium des Wirtschaftsingenieurwesens an der Universität Karlsruhe (TH) und der University of California at Berkeley Extension, 1994 Diplom in Wirtschaftsingenieurwesen, anschließend tätig als wissenschaftlicher Angestellter am Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) der Universität Karlsruhe. 2000 Promotion zum Dr. rer. pol. Üeber 25 begutachtete Publikationen in internationalen Zeitschriften, Konferenzen und Workshops in den Bereichen Evolutionäre Algorithmen und Graph-Zeichnen. Im Programmkomittee mehrerer internationaler Konferenzen, Organisator zweier Workshops zum Thema “Evolutionary Algorithms for Dynamic Optimization Problems”.