Projektgruppe: Metaheurisitiken: Neue Ideen für die Optimierung SS 2003
Lehrstuhl für Systemanalyse (LS XI)
Fachbereich Informatik
Universität Dortmund
PG-Veranstalter: Thomas Beielstein, Karlheinz Schmitt
Referat: Particle Swarm Optimization
Referent: Igor Vatolkin
Datum: 01.04.2003

GLIEDERUNG

I.Einführung

Eine Lösung für ein Problem zu finden, bedeutet in Informatik oder Mathematik oft einen Algorithmus zu finden, logisch vorzugehen, Formeln oder Beweise entwickeln. Experimentelle Lösungen gehören mehr zum Bereich der Physik. Allerdings gibt es auch Algorithmen, die nach keinen vorgegebenen Formeln arbeiten und dessen Arbeitsweise stochastisch ist und es sogar nicht immer verständlich ist, wie das Problem gelöst wird, aber trotzdem zur Lösung führen. Dieses spannende Gebiet basiert auf biologischen Erkenntnisse wie Evolution, Mutationen, Verhalten von Organismen in größeren Gruppen... Dazu gehört auch so genanntes Particle Swarm Problem, oder Particle Swarm Optimization (PSO), das das Hauptthema dieses Referats ist. Bevor es zur Definition und näheren Betrachtung von PSO kommt, werden drei Beispiele aus [9] von Problemen und Algorithmen zu dessen Lösungen betrachtet, dessen Vorgehensweise ähnlich dem PSO-Algorithmus ist.

1. Texterkennung

Das von Oliver Selfridge in den Sechzigern Jahren des letzten Jahrhunderts entwickelte Programm befasste sich damit, Textmuster zu erkennen, die zum Teil schwer lesbar oder fehlerhaft waren, indem der Computer versuchte, selbst aus den Beispielen zu „lernen“. Das Programm war in mehrere Unterprogramme aufgeteilt. Diese kleinen Unterprogramme, „Dämonen“ genannt, waren für die Erkennung einzelner Buchstaben zuständig. Wenn ein Buchstabe ähnlich wie „a“ aussah, meldete sich höchstwahrscheinlich a-Dämon, möglicherweise auch o-Dämon, weil die Buchstaben sich ähnelten. Der „Master Dämon“ sammelte die Meldungen aller kleineren Dämonen und entschied sich für einen Buchstaben. Die weitere Entwicklung des Programms bestand aus noch einfacheren Unterprogrammen, die für einfache geometrische Gestalten zuständig waren – Kreise, horizontale oder vertikale Linien. Der Kreis-Dämon könnte sich mit sehr hoher Wahrscheinlichkeit bei Buchstaben wie „a“ oder „o“ melden, der „Vertikale Linie-Dämon“ bei „p“ oder „t“. Die höhere Stufe, bestehend aus 26 Buchstaben-Dämonen, hatte nun mehr Informationen, um ihrerseits dem Hauptdämon erkannte Buchstaben vorzuschlagen. Entscheidend für erfolgreiche Arbeit dieses Algorithmus war, dass das Programm *selbst* lernte. Nach mehreren Versuchen, eine Schriftart zu erkennen und der Korrektur vom Tester war die Erkennungsquote sehr hoch. Beispielweise, hat der Kreis-Dämon gelernt, auch schiefere Ellipsen zu erkennen. Allerdings hat das Programm versagt, wenn die Versuche unterschiedliche Schriftarten enthielten.

2. Sortierung von Zahlen

Ein anderes Problem war, eine Reihenfolge von Zahlen zu sortieren. Danny Hilles entwickelte vor einigen Jahren ein Programm, das selbst lernte und versuchte, durch Ausprobieren einen guten Sortieralgorithmus zu finden. Der Entwickler vom Programm musste gar nicht selbst die Algorithmen verstehen oder entwickeln, er konnte die Arbeitsweise vom Programm beobachten und entscheiden, ob ein Algorithmus gut ist, nach der Anzahl der von dem Programm ausgeführten Schritten. Das Programm bestand wieder aus vielen kleinen Unterprogrammen, die zufällige Reihenfolgen von Befehlen konstruierten. Die könnte man als digitale Genbank beschreiben. Die Unterprogramme ließen sich etwas *mutieren* (leicht verändern) oder mit den anderen *kreuzen* (kombinieren), die erfolgreichsten Mutationen und Kreuzprodukte wurden gewählt und weiter verändert oder gekreuzt.

Es gab allerdings ein Problem – wenn man sich eine Funktion vorstellt, die *Tauglichkeit* (*Fitness*) von Algorithmen misst, könnte es durchaus sein, dass das Programm bei einem lokalen Maximum einer solchen Funktion blieb und sich nicht mehr zu einem globalen Maximum (in diesem Fall dem besten, also schnellsten Sortieralgorithmus) begeben konnte. Dafür kreierte Hillis eine andere Art von Unterprogrammen – so genannte *Raubtiere*, die die mutierten oder gekreuzten Unterprogramme störten, wenn sie sich lange an einem lokalen Maximum befanden. Die Funktionsweise war einem biologischen Subsystem sehr ähnlich.

3.Traveller Salesman Problem (TSP)

Das berühmte Traveller Salesman Problem besteht darin, einen kürzesten Weg zu finden, der alle gegebenen Städte besucht, bei vorgegebenen Positionen der Städte auf der Karte. Schon bei 15 Städten gibt es Milliarden potentieller Routen, die durch alle Städte laufen. Eine praktische Anwendung von TSP wäre zum Beispiel, den schnellsten Weg für die Daten im Internet zu finden, wo die Server die Rolle der Städte übernähmen.

1999 entwickelte Marco Dorigo eine Lösung, die auf einer einfachen Idee basierte: er ließ *Ameisen* arbeiten! Natürlich keine echten Ameisen, sondern virtuelle, die nach den Ernährungsquellen suchten. Sobald eine Ameise alle Quellen besuchte, verfolgte das Programm ihren Weg zurück und verteilte Pheromone – Lockstoffe für die anderen Ameisen. Da die Menge der Pheromone begrenzt war, waren die auf kürzeren Wegen dichter verteilt und auf längeren dünner. Nachdem sie die Wege erforschten, waren viele Ameisen mehr angelockt, die kürzeren Wege zu nehmen. Diese Idee wurde von den echten Ameisen übernommen, bei denen mehrere Experimente gezeigt haben, dass die Ameisen in der Lage waren, schnell die kürzesten Wege zu Ernährungsquellen zu finden.

4. Artificial Life als Hintergrund

Hintergrund für PSO sind Beobachtungen über das Verhalten von unterschiedlichen Organismen – Ameisen, Vogelschwärmen, Gruppen von Fischen. Der Begriff Artificial Life verbindet im Allgemeinen Rechnerprobleme und biologische Techniken, einerseits, zum Simulieren von biologischen Gruppen durch die Rechnerprogramme, und andererseits, zum Einsetzen von biologischen Techniken bei Rechnerproblemen. Die wichtige Rolle spielt hier das soziale System, das Teilen gesammelter Informationen zwischen den Organismen. Mehrere biologische Studien haben das Verhalten in einem sozialen System untersucht und aufgrund dessen man solche Systeme modellieren kann und zur Lösung von vielen Rechnerproblemen einsetzen.

5. Grobe Definition von PSO

Ein mögliches Szenario wäre eine Gruppe von Vögeln, die auf der Suche nach Futter sind. Das Futter befindet sich auf einem einzigen Platz, die Vögel suchen danach und benutzen dabei schon gesammelte Informationen – es macht Sinn, dem Vogel zu folgen, der am nahesten beim Futter ist. In PSO wird jeder „Vogel“ „Partikel“ (particle) genannt, jedem Partikel wird ein Nutzwert zugewiesen, der von der Funktion abhängt, die optimiert werden muss. Die Partikel bewegen sich in einem multidimensionalen Raum mit unterschiedlichen Geschwindigkeiten, auf der Suche nach dem Optimum und folgen dabei den im Moment erfolgreichsten Partikeln. Die Initialisierungsmenge von Partikeln wird zufällig im Raum verteilt, und nachher wird das Verhalten schrittweise simuliert, die Formeln zur Berechnung der aktuellen Geschwindigkeit von Partikeln werden nachher genau vorgestellt.

6. Vergleich mit dem sozialen System von Menschen

Ein Motiv für PSO war das Modulieren von sozialem Verhalten der Menschen. Ein wichtiger Unterschied von menschlichem Verhalten zu dem Verhalten von primitiven Organismen oder Fischen und Vögeln ist dessen Abstraktion. Vogelschwärme passen ihr Verhalten nach vorhandenen Raubtieren, Temperaturen, Futter und physikalischen Bewegungen an. Menschen müssen sich dagegen noch durch kognitive und erfahrungsbezogene Parameter beschreiben lassen. Bei der Rechnersimulation spielt außerdem noch die Kollision eine wichtige Rolle – zwei Individuen können gleiches Verhalten und Stand der Erfahrungen vorweisen, können aber nicht zum selben Zeitpunkt am selben Platz vorhanden sein. Außer dem dreidimensionalen Raum, wo die Menschen sich

bewegen, gibt es aber multidimensionale psychosoziale Räume, die abstrakt und kollisionsfrei sind, wo die Bewegungen stattfinden können.

II. Das Modell von einem Schwarm

1. Drei Prinzipien von Verhalten in einem sozialen System

Nach Eberhart und Kennedy [4] gibt es drei grundlegende Prinzipien, die das Verhalten von Organismen in einem sozialen System charakterisieren: Auswertung – Vergleich – Imitation.

Alle lebenden Organismen *werten* die momentane Situation und Anreizen *aus*. Kein Lernprozess kann stattfinden, wenn keine Analyse der Umwelt passiert, wenn schlechte Erfahrungen von den guten nicht differenziert werden.

Organismen in Schwärmen von Partikeln wie in sozialen Systemen beschäftigen sich oft damit, sich mit den anderen zu *vergleichen*. Dabei werden Standarten des Verhaltens definiert. Die Menschen messen beispielsweise ihr Intelligenzniveau durch den IQ, bei den Tieren spielt Nähe zu Futterquellen eine entscheidende Rolle, bei Partikeln bewegt der Vergleich die Suche nach dem Optimum, je nach dem zu lösenden Problem.

Zuletzt kommt das Prinzip der *Imitation*, Nachmachen des Verhaltens anderer, möglicherweise erfolgreicheren Individuen. Wenn ein Affe einfach das Verhalten eines anderen Affen nachmacht, oder ein Papagei die Worte wiederholt, ist es noch lange nicht mit der Imitation vergleichbar, die im menschlichen sozialen System stattfindet. Hier wird nicht nur das bloße Verhalten, sondern auch Ziele und Motive der anderen mitverfolgt.

Diese drei Aspekte können in Algorithmen eingesetzt werden, um komplexe soziale Modelle zu kreieren.

2. Fundamentale Prinzipen der Intelligenz von einem Schwarm

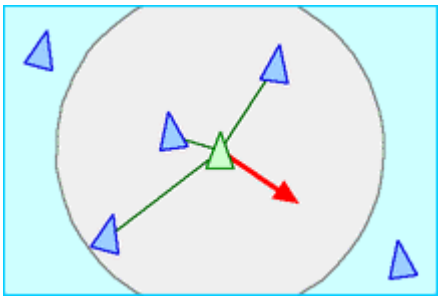
Um das Verhalten von einem Schwarm zu konkretisieren, kann man fünf folgende Prinzipien betrachten, verdeutlicht in der Veröffentlichung von Milonas [12]:

1) *Prinzip der Nachbarschaft:* Population kann einfache Bewertungen von Raum und Zeit realisieren;
2) *Prinzip der Qualität:* Population ist in der Lage, auf Qualitätsparameter der Umgebung zu reagieren.
3) *Prinzip der unterschiedlichen Reaktionen:* Population soll ihre Aktivitäten nicht nur in übertrieben engen Möglichkeitsgrenzen verüben.
4) *Prinzip der Stabilität:* Population soll nicht die Art ihres Verhaltens mit jeder Veränderung der Umgebung ändern.
5) *Prinzip der Adaptivität:* Population soll in der Lage sein, ihr Verhalten zu optimieren, wenn es sich für die Lösung des Problems lohnt.

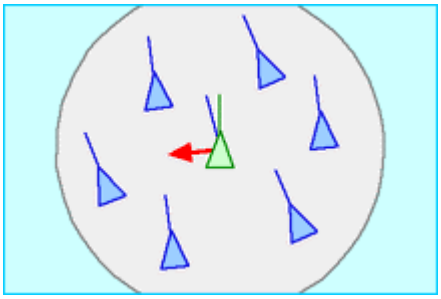
Prinzipien 4) und 5) sind gegenüber stehend, wichtig ist hier eine Balance zu finden, dass Population sich nicht zu konservativ an ihrem Verhalten festhält, aber sich auch nicht ständig verändert und sich nicht basierend auf Erfahrungen entwickeln kann.

3. Boids – ein Beispielmodell

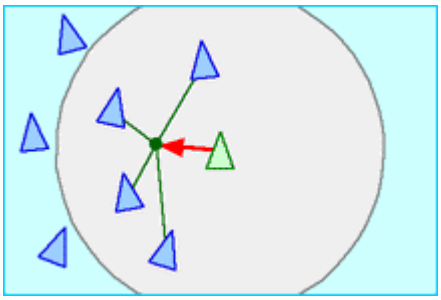
Ein bekanntes Modell sind die von Craig Reynold entwickelten Boiden. Unter [13] kann man das entsprechende Java-Applet anschauen. Wichtige Überlegungen zu dem Modell (oder den Prinzipen zu den Bewegungsarten) waren:



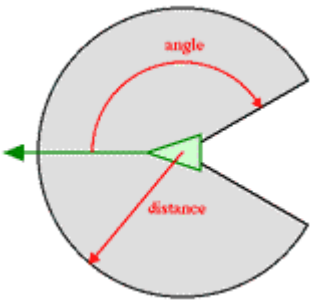
Trennung: Boide bewegen sich nach außen, um Gedränge zu vermeiden.



Anpassung: Boide werden so gesteuert, dass die Richtung der Bewegung ungefähr den Richtungen der Nachbarn entspricht.



Zusammenhalt: Boide werden zu der mittleren Poisiton einer Gruppe angezogen.



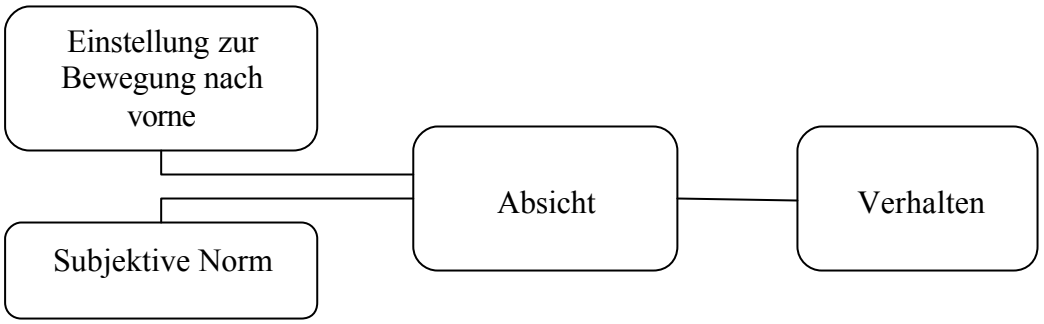
Die ***Nachbarschaft*** von einem Boid wird durch *Radius* und *Winkel* charakterisiert. Ein Boid hat zwar den Zugriff auf den gesamten definierten Raum, reagiert aber nur auf die anderen Boide, die sich in seinem Umkreis befinden.

4. Sammeln von Informationen: eigene Erfahrung und Erfahrung der Gruppe

Das Verhalten der Partikel in einem Schwarm ist von den gesammelten Erfahrungen abhängig. Um nach der optimalen Position zu suchen, kann ein Partikel auf zwei Arten von Informationen zugreifen: erstens die, die er selbst durch die vorherigen Bewegungen gesammelt hat und zweitens die von den anderen Partikeln. In dem zweiten Fall gibt es wieder zwei Möglichkeiten. Entweder ist dem Partikel die beste Position bekannt, die das erfolgreichste Partikel in der gesamten Gruppe gefunden hat (*gbest*, global best), oder die beste Position, die von seinen Nachbarn gefunden wurde (*lbest*, local best). Dabei kann man

definieren, wie groß die Nachbarschaft eines Partikels ist, für $k = 2$ werden zum Beispiel für Partikel i beste Positionen von Partikeln i-1 und i+1 bekannt und daraus kann man die beste Position von Partikeln i-1, i, i+1 ermitteln.

5. Ajzen und Fishbein's Reasoned Action Model



In dem Model von Ajzen und Fischbein (1980) hängt die Absicht der Person, eine bestimmte Bewegung durchzuführen, von zwei Werten ab. Der erste ist die Einstellung der Person selbst, die lineare Kombination vom Glauben an das Geschehen bestimmter Ergebnisse (b_i, *belief*) und Beurteilung dieser Ergebnisse (e_i, *evaluation*) ist:

$$A_0 = \sum_{i=1}^{n} b_i e_i$$

Der zweite Wert ist subjektive Norm, basiert auf sozialen Faktoren. Die ist eine lineare Kombination von dem Glauben, dass andere bestimmtes Verhalten ausüben werden (b_i, belief) und der eigenen Motivation (m_i, motivation), diesen anderen zu folgen:

$$SN_0 = \sum_{i=1}^{n} b_i m_i$$

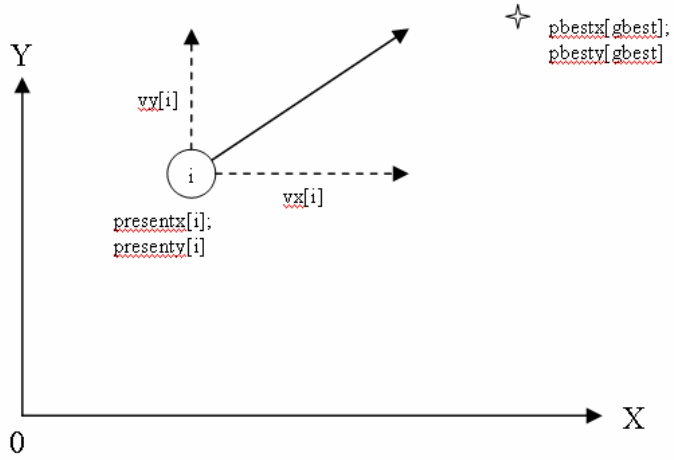
Man kann sich natürlich streiten, ob es besser wäre, statt linearer Kombination den mittleren Wert zu berechnen.

III. PSO: Algorithmus, Formel, Parameter, Änderung der Parameter

1. Erste Überlegungen zu der Formel

Bei der Simulationen der Vogelschwärme vom Heppner [8] war der Raum für Partikel zweidimensional. Der beste von den Partikeln gefundene Wert wurde im Array pbest[] gespeichert, dessen Größe gleich der Anzahl der Partikel war, die Position von diesem Wert entsprechend in pbestx[] und pbesty[]. Die Geschwindigkeiten der Partikel, die mit jedem Schritt der Simulation zu berechnen waren, wurden in vx[] und vy[] gespeichert. pbestx[gbest] war die x-Position des besten gefundenen Wertes, pbesty[gbest] die y-Position. Die Berechnung lief folgendermaßen:

```
for i = 1 to particlesNumber do
  if presentx[i] > pbestx[gbest]
    then vx[i] = vx[i] – rand()·g_incr
  if presentx[i] < pbestx[gbest]
    then vx[i] = vx[i] + rand()·g_incr
  if presenty[i] > pbesty[gbest]
    then vy[i] = vy[i] – rand()·g_incr
  if presenty[i] < pbesty[gbest]
    then vy[i] = vy[i] + rand()·g_incr
end
```



Das Bild rechts illustriert die Bewegung des Partikels. g_incr war ein Systemparameter, rand() ein stochastischer Parameter, der eingeführt wurde, damit das Verhalten von einzelnen Partikeln unterschiedlich und naturgetreu wäre. Die hohen rand()-Werte brachten die Partikel dazu, schneller neue Bereiche von dem Raum zu untersuchen, was einerseits schneller zum Finden der besten Position führen konnte, anderseits aber zufällig von dem richtigen Weg ablenkte.

2. Algorithmus

Insgesamt würde der Pseudocode für PSO so aussehen:

```
for each particle
  Initialize particle                    // Startposition initialisieren
end

do
  for each particle
    calculate fitness value              // Aktuelle Position bewerten
                                         // Beste Position des aktuellen Partikels setzen
    if fitness value is better than the best fitness value pbest in history
      set current value as the new pbest
  end
                                         // Beste Position von allen Partikeln setzen
  choose the particle with the best fitness value of all the particles as the gbest
  for each particle
                                         // Geschwindigkeit berechnen
    calculate particle velocity according equation
                                         // Neue Position von dem aktuellen Partikel setzen
    update particle position
  end
                                         // Nach bestimmter Anzahl von Schritten oder aus
                                         // einem anderen Grund abbrechen
while maximum iterations or minimum error criteria is not attained
```

3. Multidimensionalität

Das oben beschriebene Beispiel simulierte das Verhalten der Vögel in einem zweidimensionalen Raum. Da sehr viele Probleme oder Funktionen, deren Nullstellen zum Beispiel mit Hilfe vom PSO zu finden wären, mehrdimensional sind, verwalteten die Autoren im nächsten Entwicklungsschritt x-dimensionale Arrays. Die Variablen ließen sich durch $present_{id}$ und v_{id} beschreiben, wo *i* der Nummer des Partikels entsprach und *d* der Dimension.

4. Weitere Überlegungen: Beschleunigung nach dem Abstand zum Optimum und Begrenzung der Geschwindigkeit

Interessante Modifikation nachher war, die Geschwindigkeit der Partikel an die Entfernung zur Position mit dem besten Wert anzupassen. Die Formel dafür wäre:

$$v_{id} = v_{id} + rand() \cdot p\_incr \cdot (pbest_{id} - present_{id})$$

Damit die Partikel sich nicht zu stark im Raum zerstreuen würden, wurde konstanter Parameter V_{MAX} eingeführt und die berechnete Geschwindigkeit mit dem verglichen:

$$\text{if } v_{id} > V_{MAX} \text{ then } v_{id} = V_{MAX}$$
$$\text{else if } v_{id} < V_{MAX} \text{ then } v_{id} = -V_{MAX}$$

V_{MAX} müsste natürlich der Größe des Raumes entsprechen, damit die Partikel sich nicht schon bei dem ersten Schritt zu weit bewegten. Änderung der maximalen Geschwindigkeit wird weiter unten in (10) betrachtet.

5. Liste der Parameter für PSO und Initialisierung

Hier werden für PSO relevante Parameter aufgelistet, angefangen mit den Initialisierungsparametern, die zu Überlegungen für die Implementierung eine wichtige Rolle spielen:

Initialisierungsparameter

Anzahl der Partikel: nach Experimenten stellte sich heraus, dass die Anzahl der Partikel zwischen 10 und 50 gut war. Allerdings wären für manche schwierigeren Probleme mehr Partikel nötig, um sie schnell zu lösen, und so könnte die Anzahl auch 100-200 betragen.

Dimension der Partikel: ist von dem zu lösenden Problem abhängig.

Größe des Raumes: ist von dem zu lösenden Problem abhängig.

Höchstgeschwindigkeit von Partikeln: von dem Problem und der Größe des Raumes abhängig.

Startpositionen von Partikeln: Partikel werden normalerweise zufällig im Raum verteilt.

Startgeschwindigkeiten von Partikeln: auch normalerweise zufällig, zwischen $-V_{MAX}$ und V_{MAX}.

Abbruchbedingungen: maximale Anzahl von Iterationsschritten sowie maximal erlaubte Anzahl von Fehlern.

Globalität gegen Lokalität: die Balance zwischen Wirkungen von *gbest* und *pbest*: Größere Rolle von *gbest* kann zwar schneller zur optimalen Lösung führen, wäre aber bei vielen lokalen Extremastellen weniger sinnvoll, wobei alle Partikel von einem lokalen aber nicht globalen Optimum zu stark angezogen würden.

Lernfaktoren: entpricht g_incr in dem obigen Beispiel, beeinflusst die Schnelligkeit der Reaktion von einem Partikel auf die neuen Bestwerte.

Stochastische Parameter allgemein: Zufälligkeitsfaktoren, deren Änderung experimentell betrachtet werden kann.

Trägheitswert (intertia weight): wurde eingeführt von Shi und Eberhart, wird weiter unten in (9) beschrieben.

Variablen, dessen Werte sich im Laufe des Algorithmus verändern können

Geschwindigkeit: aktuelle Geschwindigkeit der Partikel

Position: aktuelle Position der Partikel.

Erfahrungen von einem Partikel: Der beste von einem Partikel gefundene Wert und **seine** Position.

Erfahrungen von allen Partikeln: Der beste von allen Partikel gefundene Wert und seine Position.

6. PSO-Formel

Nach Kennedy und Eberhart sieht die Formel zur Berechnung der aktuellen Geschwindigkeit und Position von einem Partikel folgendermaßen aus:

$$\begin{cases} \vec{v}_i(t) = \vec{v}_i(t-1) + \varphi_1(\vec{p}_i - \vec{x}_i(t-1)) + \varphi_2(\vec{p}_g - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \end{cases}$$

Aktuelle Position des Partikels ist daher eine Funktion aus der vorherigen Position, Geschwindigkeit und besten gefundenen Werten von diesem Partikel und allen Partikeln:

$$\vec{x}_i(t) = f(\vec{x}_i(t-1), \vec{v}_i(t-1), \vec{p}_i, \vec{p}_g)$$

t steht für aktuelle Zeit (genau gesagt Iterationsschritt), *t*-1 für den Schritt davor.

φ_1 und φ_2 sind zwei zufällige Variablen, die die Wirkung des bestgefundenen Wertes von dem Partikel (hier als Vektor $\vec{p}_i$ zu entsprechenden Position) bzw. des bestgefundenen Wertes von allen Partikeln ($\vec{g}_i$) festlegen. Das Effekt von φ_1 und φ_2 ist das ungleichmäßige Bewegen von dem Partikel zu dem Punkt, den man als mittleres Gewicht von den beiden besten Werten beschreiben kann:

$$\frac{\varphi_1 \vec{p}_i + \varphi_2 \vec{p}_g}{\varphi_1 + \varphi_2}$$

Aufgrund der Zufälligkeit dieser Variablen variiert die Position von diesem Punkt.

7. PSO für binäre Zahlen

Je nach Problem kann PSO binär oder reell vorgehen. Wenn eine mehrdimensionale Funktion in jeder Dimension nur den Wert 1 oder 0 einnimmt, reicht es, die Partikel im binären Raum zu betrachten. Solche Funktionen sind typisch für unser soziales Verhalten, wenn ein Individuum sich zu einem „ja" oder einem „nein" entscheidet. Die Wahrscheinlichkeit, dass ein Partikel sich in einer Dimension *d* für 1 entscheidet, hängt dann davon ab, wie die Entscheidung im vorherigen Schritt war (x_{id}(t-1)), der Neigung des Partikels, sich für 1 zu entscheiden (gleichbedeutend der Geschwindigkeit bei der Formel oben, v_{id}(t-1)), und analog zu vorher Besprochenem, dem besten gefundenen persönlichen Wert p_{id} und dem besten in der Gruppe gefundenen Wert p_{gd}:

$$P(x_{id}(t)=1) = f(x_{id}(t-1), v_{id}(t-1), p_{id}, p_{gd})$$

8. Änderung von φ.

In [10] hat Kennedy die Wirkung der Änderung von φ untersucht. Er nahm das vereinfachte System zur Betrachtung, wo die Geschwindigkeit nur von dem besten gefundenen Wert des Partikels und nicht der ganzen Gruppe abhing:

$$\begin{cases} \vec{v}_i(t) = \vec{v}_i(t-1) + \varphi(\vec{p}_i - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \end{cases}$$

Bei $\varphi = 0$ hat sich die Geschwindigkeit nicht verändert, jedes Partikel bewegte sich konstant in die Richtung, die bei der Initialisierung zufällig gewählt wurde.

Die Werte $\varphi \geq 4$ waren auch wenig sinnvoll, da die Trajektorie des Partikels sich sehr schnell vergrößerte. Bei $\varphi = 4$ war die Vergrößerung, d.h. die Differenz $x_i(t)$-$x_i(t-1)$ noch linear steigend, bei $\varphi \geq 4$ schon exponentiell.

Die besten Werte waren nach den Experimenten im Bereich [0,1], und es ist meistens angemessen, φ als ein zufälliger Wert zwischen 0 und 1 zu initialisieren. Das System der Partikel könnte man in dem Fall weniger vorhersehbar und flexibler machen. Die Kehrseite der Medaille wäre dann ein sehr zufälliges Verhalten von dem System, „Spaziergang des Betrunkenen" („drunkard's walk"), genannt.

9. Einführung und Änderung von dem Trägheitswert

Eine weitere Modifikation von PSO führte der Trägheitswert ein, der die Auswirkung der vorherigen Geschwindigkeiten auf die aktuelle Geschwindigkeit eines Partikels festlegte. Mit dem Trägheitswert *w* sah die Formel für PSO so aus:

$$\begin{cases} \vec{v}_i(t) = w \cdot \vec{v}_i(t-1) + \varphi_1(\vec{p}_i - \vec{x}_i(t-1)) + \varphi_2(\vec{p}_g - \vec{x}_i(t-1)) \\ \vec{x}_i(t) = \vec{x}_i(t-1) + \vec{v}_i(t) \end{cases}$$

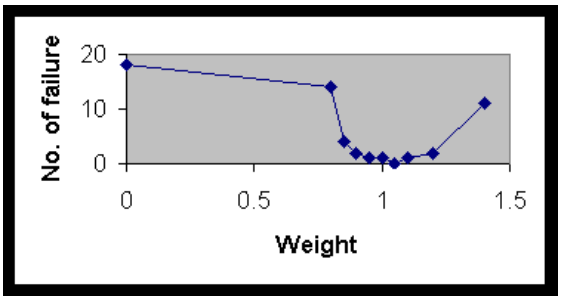
Ein größerer Wert von *w* führte dazu, dass die Partikel leichter tendierten, neue Gebiete zu erforschen, ein kleinerer Wert von *w* brachte die Partikel dazu, das aktuelle Gebiet genauer zu untersuchen. Die passende Wahl von *w* beschaffte den Ausgleich zwischen globalen und lokalen Fähigkeiten, die Gebiete zu erforschen. Experimente mit $V_{MAX} = 2$ haben gezeigt,

dass optimale Trägheitswerte aus dem Intervall [0.9, 1.2] waren. Dazu siehe auch die Grafiken unten in (10).

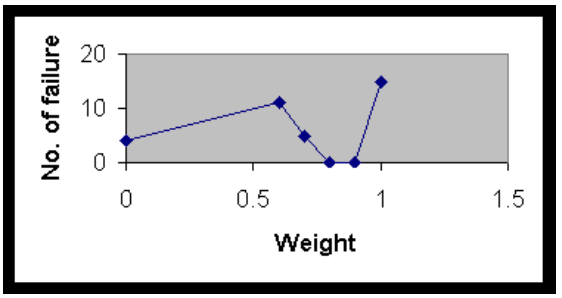
10. Änderung der maximalen Geschwindigkeit

Maximale Geschwindigkeit V_{MAX} beeinflusst die Fähigkeit der Partikel, neue Gebiete zu erforschen. Wenn V_{MAX} groß ist, gibt es mehr Möglichkeiten durch unterschiedliche Trägheitswerte diese Fähigkeit zu steuern. Da maximale Geschwindigkeit auf diese Fähigkeit indirekt wirkt und der Trägheitswert direkt, ist es besser, diese Fähigkeit nur durch den Trägheitswert zu kontrollieren. Möglich wäre, V_{MAX} abzuschaffen und nur über *w* die Partikel zu steuern. Dabei müsste man aber sehr vorsichtig sein, da große *w*-Werte die lokale Suche der Partikel verschlechtern.

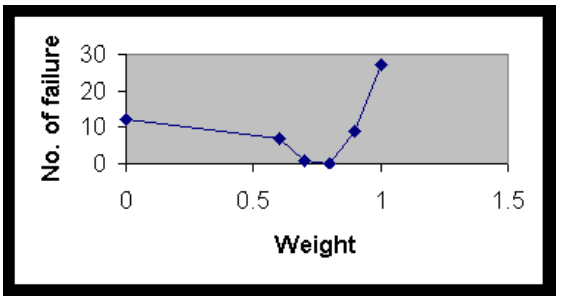
Unten sind die Grafiken der Untersuchung in [5], wo bei unterschiedlichen Werten der V_{MAX} Trägheitswert (*Weight*, X-Achse) geändert wurde. Auf der Y-Achse war die Anzahl der Fehler zu messen. Ein Fehler passierte, wenn nach einer bestimmten Anzahl der Iterationsschritte die Suche erfolglos war.



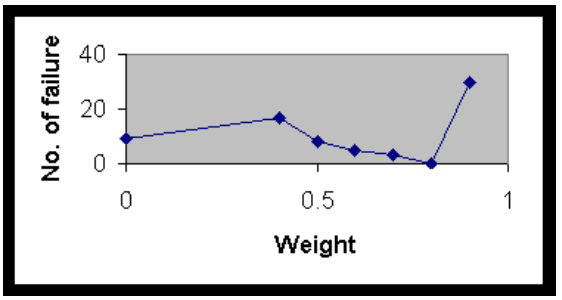
$V_{MAX} = 2$



$V_{MAX} = 5$



$V_{MAX} = 10$



$V_{MAX} = X_{MAX}$

Im letzen Versuch war die maximale Geschwindigkeit am größten und erlaubte den Partikeln, in einem Schritt von einer Grenze des Raumes zu der anderen zu gehen.

11. Änderung der Populationsmenge

Hier und in folgenden Abschnitten (12) und (13) werden die Ergebnisse der Experimente von Carlisle und Dozier [2] betrachtet. Sie benutzten eine weitere Modifikation der PSO, die ihre Leistung durch die Einführung der Konstante *K* verbesserte:

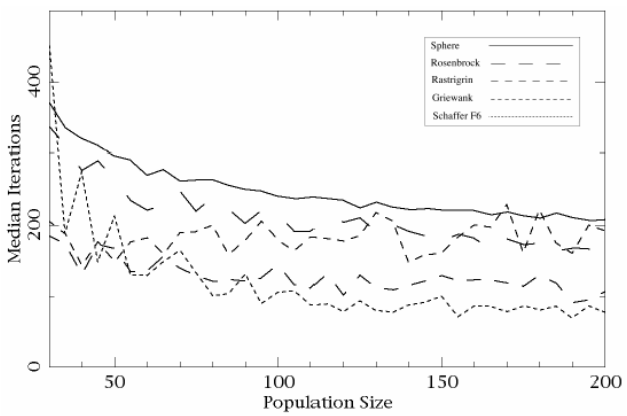
$$\vec{v}_i(t) = K(\vec{v}_i(t-1) + \varphi_1(\vec{p}_i - \vec{x}_i(t-1)) + \varphi_2(\vec{p}_g - \vec{x}_i(t-1))),$$

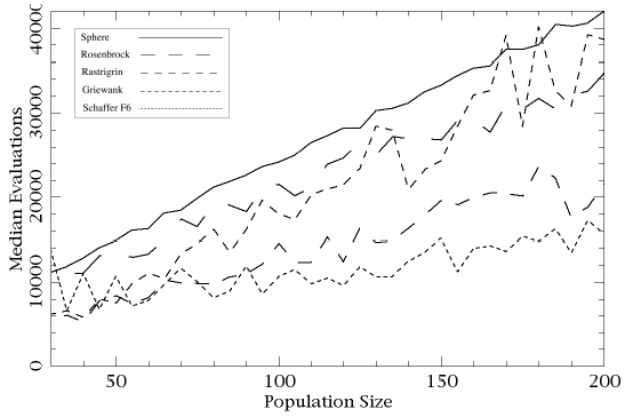
$$K = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \varphi = \varphi_1 + \varphi_2, \varphi > 4$$

Fünf unterschiedliche Funktionen wurden getestet, jeder Versuch wurde 20 mal wiederholt, und in jedem Versuch war die obere Grenze der Iterationsschritte gleich 100.000.

Mit der Vergrößerung der Population war die Anzahl der zur Lösung des Problems benötigten Schritte langsam geringer. Da aber mit der Vergrößerung der Population die Zahl der Auswertungen von jedem Partikel stark anstieg, kostete es sehr viel Rechenzeit. Die

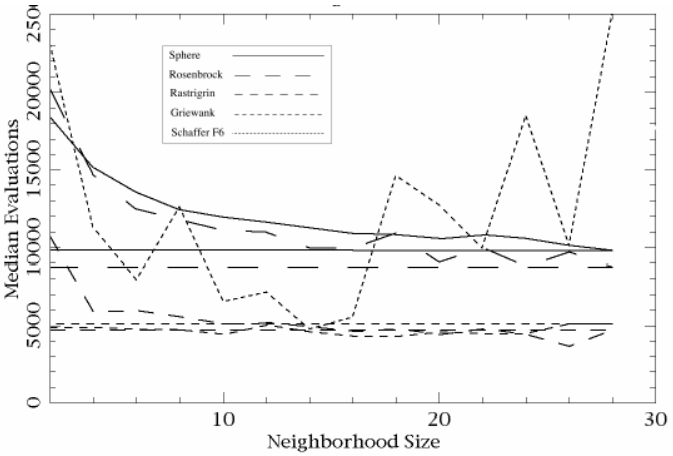
Versuche haben gezeigt, dass die goldene Mitte zwischen niedriger Anzahl der Iterationsschritte und Rechenaufwand bei der ungefähren Größe der Population gleich 30 war. Die beiden Grafiken unten illustrieren die Abhängigkeit der Anzahl der Iterationsschritte (Median Iterations) und Rechenaufwand (Median Evaluations) von der Populationsgröße. Die unterschiedlichen getesteten Funktionen sind durch unterschiedliche Linienarten dargestellt.





12. Änderung der Nachbarschaft

Nach allgemeinen Überlegungen wäre eine kleine Nachbarschaft (siehe II-4) gut für das Vermeiden lokaler Extremastellen, und globale für die schnelle Annäherung zu dem Optimum. Die Experimente haben allerdings gezeigt, dass die Vergrößerung der Nachbarschaft bis zur globalen harmlos war und der Rechenaufwand entweder ungefähr gleich blieb, oder kleiner wurde, da die Partikel auf die Erfahrungen aller anderen zugreifen konnten. Nur eine Funktion, die viele lokale Extremastellen enthielt, bereitete Probleme, und der Graph wurde mit der Vergrößerung der Nachbarschaft ziemlich schwankend.



13. Synchronisation der Aktualisierungen.

Bei lokalen Nachbarschaften ist die Berechnung von den besten gefundenen Werten *lbest* asynchron, wenn die Liste aller Partikel nacheinander durchgegangen wird. Die Berechnung von dem besten globalen Wert *gbest* geschieht dagegen zwischen den Iterationsschritten und ist deswegen synchron. Die Untersuchung hat gezeigt, dass asynchrones Aktualisieren weniger Rechenaufwand bedeutete. Die Tabelle rechts fasst die Versuchsdaten für unterschiedliche Funktionen zusammen.

	Median Iterations					
	Neighborhood of 2		Neighborhood of 14		Global Neighborhood	
	Async	Sync	Async	Sync	Async	Sync
Sphere	622.5	628.0	376.5	401.5	331.5	368.5
Rosenbrock	685.0	645.0	365.0	367.5	288.0	327.0
Rastrigrin	312.0	314.5	162.5	162.0	155.0	206.5
Griewank	152.0	181.5	154.5	148.5	173.0	171.0
Schaffer F6	624.0	993.5	310.0	499.5	255.5	522.5

14. Erfüllung der fundamentalen Prinzipen der Intelligenz von einem Schwarm

In (II-2) wurden die Prinzipien der Intelligenz von einem Schwarm nach Millonas vorgestellt. Nachdem hier der PSO-Algorithmus mit Formeln genau vorgestellt wurde, wird die Erfüllung dieser Prinzipien in dem vorgestellten Model untersucht.

1) *Prinzip der Nachbarschaft:* Die Positionen des Raumes werden bewertet, die Rolle der Zeit übernehmen die aufeinander folgenden Iterationsschritte.
2) *Prinzip der Qualität:* Population bewertet die besten gefundenen Positionen durch *pbest* und *lbest* oder *gbest*.
3) *Prinzip der unterschiedlichen Reaktionen:* Die Reaktionen auf *pbest* oder *gbest* können sich mit der Änderung von φ_1 und φ_2 variieren.
4) *Prinzip der Stabilität:* Population ändert die Art vom Verhalten nur wenn sich *gbest* ändert, ist also ohne Veränderung des Informationsstandes stabil.
5) *Prinzip der Adaptivität:* Wenn sich *gbest* ändert, ändert sich auch die Art des Verhaltens der Population, also ist die adaptiv.

Damit sind die Prinzipien erfüllt.

IV. Vergleich von PSO mit evolutionären und genetischen Algorithmen

1. Vergleich mit evolutionären Algorithmen

Um den Vergleich verständlich zu machen, wird hier kurz die Arbeitsweise der evolutionären Algorithmen (EA) vorgestellt.

Wie bei PSO, wird bei EA die Entwicklung einer Population betrachtet. Die Veränderung der Population P geschieht nach der Formel

$$P' = \mu(s(f(P))),$$

wo f die Tauglichkeitsfunktion (Fitness function) ist, die die Bewertungen von jedem Individuum liefert; s die Selektionsfunktion ist, die die Individuen mit schlechter Leistung entfernt und μ die Mutationsfunktion ist, die zufällig die Teilmenge der Individuen verändert.

Die Unterschiede zwischen EA und PSO sind:

1) Veränderung der Population ist in EA ein Produkt von drei Funktionen, bei PSO gibt es nur zwei, die Selektionsfunktion gibt es nicht (alle Partikel bleiben bis zum Ende da), die Rolle der Mutation nehmen hier die stochastischen Parameter.
2) Statt Selektionsmechanismen benutzt PSO die besten gefundenen Werte von einem Partikel, *pbest*.
3) In PSO ist das Verhalten der Partikel von *pbest* sowie *lbest*/*gbest* abhängig. Daher funktioniert PSO besser, wenn lokale Optima in der Nähe des globalen Optimums sind, EA funktionieren dagegen gut, wenn Optima gleichmäßiger verteilt sind.
4) Bei EA werden die Individuen mit schlechter Leistung entfernt, bei PSO bleibt die ganze Population bestehen, es gibt aber Begrenzung der maximalen Geschwindigkeit. Bei der Annäherung zum Optimum können sich EA-Algorithmen durch Weiterentwicklung verfeinern, Partikel in PSO können dagegen ihre Geschwindigkeit nicht so schnell anpassen.

Die Untersuchungen von Peter J. Angeline [1] haben gezeigt, dass EA bei den meisten Problemen langsamer auf der Suche nach Optima waren, dafür aber am Ende die genauer feststellen konnten. Die Partikel in PSO bewegten sich am Anfang schneller zum Ziel, waren aber nicht so präzise.

2. Vergleich mit den genetischen Algorithmen

Wie die genetischen Algorithmen (GA), benutzt PSO auch die biologischen Kulturen, das System wird mit einer zufälligen Population von Organismen initialisiert, die mit jeder Generation / jedem Schritt nach optimalen Lösungen suchen. Auch wie bei GA, werden Nutzwerte zur Bewertung der Population berechnet. Beide Algorithmen aktualisieren die

Populationen bei der Suche nach dem Optimum, beide benutzen Zufallstechnicken und beide garantieren keinen hundertprozentigen Erfolg.

Wichtige Unterschiede zu GA ist hier dagegen das *Verzichten* auf Mutationen und Crossover (Kreuzungen zwischen den Organismen). Partikel aktualisieren ihre Positionen mit der zu berechenden Geschwindigkeit und haben Speicher für die gesammelten Erfahrungen. Die werden nicht wie bei GA durch Chromosomen geteilt, wo die ganze Population sich als eine Gruppe zu dem Optimum bewegt, sondern es wird der Ein-Wege-Mechanismus eingesetzt, nur der momentan beste Wert wird allen zur Verfügung gestellt.

Die Rolle des Crossovers bei GA erfüllt bei PSO die beschleunigte Bewegung der Partikel zum Produkt von zwei Zielen, *pbest* und *gbest*. Die Anziehungskraft von diesen beiden Zielen hängt von stochastischen Faktoren φ_1 und φ_2 ab.

Mutationen beeinflussen stark den Ablauf von GA. Am Anfang können zufällige Mutationen viele Individuen schneller erfolgreicher machen. Am Ende, wenn die Tauglichkeitswerte der Individuen im Schnitt höher sind, können die Mutationen diesen Schnitt nicht mehr stark verbessern. Durch Mutationen können sogar schwächere Individuen entstehen.

Die werden bei GA wie bei EA durch Selektion aussortiert, die bei PSO nicht vorhanden ist.

Zusammenfassend:

Gemeinsamkeiten	Unterschiede
Zufällige Menge von Organismen bei der Initialisierung	Keine Mutationen, Crossover und Selektionen
Nutzwerte Bewerten die Erfolgen	Unterschiedliche Sammlung von Erfahrungen
Zufallsrolle ist wichtig	
Aktualisierung von Populationen	
Keine Garantie für den Erfolg	

V. Anwendung von PSO und ein paar weitere Überlegungen

Hier genannte Beispiele sollen weitere Ideen zur Anwendung und Erweiterung von PSO vermitteln, ohne in die genaue Beschreibung zu gehen, da sonst der Umfang des Referates explodieren kann. Die können jedoch bei der Betrachtung der PSO interessant sein.

1. Suche nach Nullstellen oder Extremastellen

Die eigentliche Anwendung von PSO ist das Feststellen der Nullstellen oder Extremastellen der komplizierten mehrdimensionalen Funktionen, die nicht linear sind, wo es keine Formeln gibt, die zu suchende Werte schnell berechnen können. Vorausgesetzt ist natürlich die Stetigkeit solcher Funktionen. Nach jedem Iterationsschritt bewerten die Partikel die Position, indem sie den Funktionswert berechnen.

2. Multimodalität

Bei einigen Problemen gibt es mehrere gleichbedeutende Lösungen so wie die Suche nach Nullstellen bei einer Funktion wie $f(x)=x^2-4$. Dies ist entsprechend zu berücksichtigen, da sonst der Algorithmus abbricht, wenn die Partikel nur eine Lösung gefunden haben.

3. Hybride PSO

Die Definition der hybriden PSO ist sehr unterschiedlich je nach Modell, hier werden zwei Beispiele betrachtet.

Zuerst, definieren Eberhart und Kennedy in ihrem Buch [4] den hybriden Schwarm als eine Menge der Partikel, die teils binäre und teils reelle Probleme lösen. Als Beispiel der Anwendung könnte man sich ein Diagnoseprogramm vorstellen, das den Patienten aufgrund mehrerer Antworten diagnostizieren kann. Einige Fragen erwarten eine binäre Antwort „ja/nein“ wie „Hatten Sie schon früher folgende Krankheit?“, die anderen erwarten eine Zahl wie Messungen von Fieber.

Løvbjerg, Rasmussen und Krink (siehe [11]) entwickelten einen hybriden Algorithmus, der auf PSO basiert und außerdem die Kreuzung benutzt, wie bei den genetischen Algorithmen. Die Geschwindigkeiten von den Kindern zweier Partikel wurden nach der neuen Formel berechnet, die die Geschwindigkeiten der Eltern berücksichtigte. Die hybride Form des Algorithmus war zwar bei einigen Versuchen von dem normalen PSO-Algorithmus übertroffen, war aber bei den anderen leistungsfähiger und schien allgemein schneller ans Ziel zu gelangen.

4. Neuronale Netzwerke

Eine weitere Anwendung von PSO sind neuronale Netzwerke. Sie werden durch mehrere Ein- und Ausgänge charakterisiert, das Netz besteht aus mehreren Gewichten, meist auf unterschiedlichen Schichten verteilt. Die Anzahl der Gewichte wäre gleich der Anzahl der Dimensionen für entsprechende PSO.

5. PSO in Physik: Kontrolle der Kernkraft- und Spannungskontrolle

In [14] setzten die Autoren PSO bei einem verteilten System ein, das die Spannung der vorgegebenen Stromanlagen aufrechterhalten sollte.

6. PSO in Medizin: Analyse vom krankhaften Zittern

In [6] wird PSO bei den Untersuchungen vom Zittern eingesetzt, um Parkinsonsche Krankheit zu diagnostizieren.

VI. Resümee

1. No Free Lunch Theorem

Um die Effektivität eines Algorithmus objektiv zu betrachten, muss man genau wissen, welche Probleme es zu lösen gibt. Interessante Überlegungen haben Wolpert und Macready gemacht. Dessen Theorem, „*The No Free Lunch Theorem“* genannt, besagt, dass kein Algorithmus besser sein kann, als irgendeiner anderer, in Bezug auf die Menge der zu berechenden Funktionen. Um das zu verstehen, stelle man sich die Aufgabe, man solle die Werte von zwei Variablen tauschen. Seien a und b die Variablen. Der erste Algorithmus führt eine neue Variable c ein und macht dann folgende Operationen:

```
c := a
a := b
b := c
```

Der andere macht folgendes:

```
a := a + b
b := a - b
a := a - b
```

Der zweite Algorithmus spart zwar Speicherplatz, benutzt aber mehr Operationen. Man könnte behaupten, der erste sei *effektiver*. Es darf aber nicht vergessen werden, dass der zweite Algorithmus aber auch die *Summe* a + b berechnet! Wenn die Aufgabenstellung entsprechend wäre „Tausche zwei Werte von den beiden Variablen und berechne die Summe“, wäre der zweite Algorithmus eindeutig effektiver, da der erste gar nicht die Aufgabenstellung erfüllt!

2. PSO

Zusammenfassend, ist PSO ein Algorithmus, der das Verhalten sozialer Systeme imitiert. Individuen interagieren miteinander durch Sammeln von Erfahrungen und bewegen sich zu den Positionen von dem Raum, die den besseren Wert haben. Der Algorithmus selbst ist sehr einfach und lässt sich mit zwei Zeilen beschreiben. Ob er effektiv ist, hängt immer davon ab, *welche* Probleme man zu lösen versucht. Erstens, können andere Algorithmen bei manchen Problemen besser sein, bei den anderen schlechter, und zweitens, wie No Free Lunch Theorem besagt, kann man die Effektivität eines Algorithmus nur sehr schwer messen, es hängt davon ab, *was* man überhaupt damit anfangen will.

VII. Literaturverzeichnis

[0]		PSO Tutorial	http://web.ics.purdue.edu/~hux/tutorials.shtml
[1]	Angeline, P. J.	Evolutionary optimization versus particle swarm optimization: philosophy and performance difference	Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, 1998
[2]	Carlisle, A. and Dozier, G.	An off-the-shelf PSO	Proceedings of the Workshop on Particle Swarm Optimization 2001, Indianapolis, IN, Apr. 2001
[3]	Eberhart R., Kennedy J.	Particle Swarm Optimization	IEEE Service Center, Piscataway, NJ, 1995
[4]	Eberhart R., Kennedy J. with Shi Y.	Swarm Intelligence	Academic Press, London, 2001
[5]	Eberhart R., Shi Y.	Parameter Selection in Particle Swarm Optimization	University Indianaplis, IN 46202, 1998
[6]	Eberhart, R. C. and Hu, X.	Human tremor analysis using particle swarm optimization	Proceedings of the IEEE Congress on evolutionary computation (CEC 1999), pp.1927-1930, Washington D.C., 1999
[7]	Eberhart, R. C. and Shi, Y.	Comparison between genetic algorithms and particle swarm optimization	Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, 1998
[8]	Heppner, F. Grenander U.	A stochastic nonlinear model for coordinated bird flocks	AAAS Publications, Washington, DC, 1990
[9]	Johnson S.	Emergence	Penguin Books, London, 2001
[10]	Kennedy J.	The Behavour Of Particles	Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, 1998
[11]	Løvbjerg, M., Rasmussen, T. K., and Krink, T.	Hybrid particle swarm optimiser with breeding and subpopulations	Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001), 2001
[12]	Millonas, M. M.	Swarms, phase transitions, and collective intelligence	Addison Wesley, Reading, MA, 1994
[13]	Reynolds C.	Boids	http://www.red3d.com/cwr/boids/
[14]	Yoshida, H., Kawata, K., Fukuyama, Y., and Nakanishi, Y.	A particle swarm optimization for reactive power and voltage control considering voltage stability	Proceedings of the International Conference on Intelligent System Application to Power System, pp.117-121, Rio de Janeiro, Brazil, 1999