**Particle Swarm Optimization** (**PSO**)

Der Particle Swarm Optimization Algorithmus nimmt seine Inspiration aus dem Tierreich und versucht die auftretenden Anforderungen analog zu diesem zu lösen. Der PSO orientiert sich bei seinem Verhalten an Insekten die in Schwärmen auftreten, wie etwa Bienen. Wenn diese auf der Suche nach guten Futterplätzen sind und in einem großen Gebiet nach den optimalen Plätzen suchen, stehen sie vor einem Optimierungsproblem, dass sie lösen müssen.

Dabei Schwärmen einzelne Bienen aus und stellen fest, wie an einer Stelle die Futterversorgung ist und gibt diese Information an den gesamten Schwarm weiter, sodass dieser anhand der einzelnen Rückmeldungen der einzelnen Bienen sieht, in welcher Richtung sich die vermutlich optimalen Essensplätze befinden. Anhang der Rückmeldungen machen sich nun mehrere der einzelnen Bienen in Richtung der optimalen Plätze auf und der Schwarm verdichtet sich dort.

Auf diese Weise haben die Bienen nun eine Optimierungsaufgabe erfüllt und der PSO soll dieses Verhalten simulieren. Hierzu nutzt er einzelne Partikel an Stelle der Bienen, die Logik der Verdichtung an den Stellen mit guten Rückmeldungen bleibt jedoch dieselbe. Optimierungsaufgaben sollen damit möglichst effizient und schnell gelöst werden.

Allerdings hat der PSO auch Nachteile, die bei seinem Einsatz beachtet werden müssen. Zum einen ist es möglich, das nur ein lokales Optimum gefunden wird, statt des globalen und zum anderen findet er nicht für jedes Problem das optimale Ergebnis. Daher kann auch der Aufwand der betrieben muss, um den Algorithmus an die Problemstellung anzupassen stark schwanken.

**Implementierung**

Der von uns implementierte PSO Algorithmus besteht aus 7 Klassen und einem Interface, die in Java implementiert sind:

|  |  |
| --- | --- |
| PSOConstants | Beinhalltet einen Großteil der Stellschrauben des Algorithmus: Swarm size, Max Iterationen, Problem Dimension, C1, C2, W Upperbound, W Lowerbound. |
| PSODriver | Die Main Klasse mit der, der Algorithmus gestartet werden kann. |
| PSOProcess | In der PSOProcess wird der Schwarm initialisiert und es wird eine Verbindung zu rabbitMQ aufgebaut.  Über diese Verbindung werden die Werte der einzelnen Partikel an die Simulation geschickt und die Antwort derselben ausgewertet und die Fitness der letzten Iteration überprüft. Zudem beinhaltet die Klasse eine Terminierung des Algorithmus, die sich nach der Anzahl der Max Iterationen und der Ergebnis Tolerance richtet. |
| PSOUtility | Hier wird bewertet wie nützlich der Wert eines Partikels ist im Vergleich zum Gesamtergebnis. |
| Velocity | Hier wird bestimmt wie schnell die Partikel sich im Ergebnisraum bewegen. |
| Location | Hier wird die Position eines Partikels im Ergebnisraum bestimmt. |
| Particle | Initialisieren von Velocity und Location der Partikel. |
| ProblemSet | In der Klasse ProblemSet werden die Wertebereiche der 17 Variablen definiert zudem wird die maximale Geschwindigkeit der Partikel in positiver sowie negativer Richtung bestimmt und für die Abbruchbedingung des Algorithmus die Toleranz (ERR\_TOLERANCE) festgesetzt.  Die Methode evaluate() weist die neuen Positionen der Partikel den 17 Variablen zu. |

**Ergebnisse**

Die Ergebnisse der einzelnen Testfunktionen mit den zugehörigen Einstellungen können der Datei PSO\_Auswertung entnommen werden.

Am schwierigsten war es dabei die Rastrigin function zu optimieren, da der Algorithmus bei einer Fehlerfunktion schnell in ein lokales Optimum läuft. Um dieses Problem zu lösen ist zum einen die Geschwindigkeit der Partikel entscheiden. Zum andern kann man mit mehreren Iterationen bei tendenziell stätig absteigenden Funktionen den Wertebereich immer weiter eingrenzen um dem Optimum so möglich nah zu kommen.

Ein großer Vorteil des PSO ist die Geschwindigkeit mit der er ein möglichst optimales Ergebnis findet.

In den meisten Fällen reichten weniger als 100 Iterationen um ein gutes Ergebnis zu finden. Aufbauend auf diesem Ergebnis kann der Wertebereich sowie die Geschwindigkeit der Partikel eingrenzt werden, um dem absoluten Optimum noch näher zu kommen.

Die Implementierung des Algorithmus ist für alle vier Testfunktion, mit den entsprechend optimalsten Einstellungen, in den Ordnern Projekt1 bis Projekt4 untergebracht.