Datavetenskap

Mathias Johansson, Mikael Lindberg

Partikelsvärmsoptimering

En översikt och utvärdering

Magisteruppsats

2004:08

Partikelsvärmsoptimering

En översikt och utvärdering

Mathias Johansson, Mikael Lindberg

© 2005 Författaren och Karlstads Universitet

Denna uppsats är skriven som en del av det arbete som krävs för att erhålla en magisterexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Mathias Johansson

Mikael Lindberg

Godkänd, 2004-12-17

Opponent: Stefan Alfredsson

Handledare: Thijs Holleboom

Examinator: Donald Ross

Sammanfattning

Många problem är så komplexa att det inte är möjligt att erhålla en optimal lösning inom en rimlig tidsram. Optimering är en teknik som alltid funnits och som blivit allt vanligare under de senaste decennierna för att möta den ökade efterfrågan på så nära optimala lösningar som möjligt för komplexa problem. Partikelsvärmsoptimering, PSO, är en relativt ny optimeringsmetod, som först presenterades år 1995. Sedan år 1995 har intresset för PSO ökat, mest tack vare dess påvisade effektivitet vad gäller optimering, men kanske även tack vare dess bakomliggande teorier om kollektiv intelligens. Idén med kollektiv intelligens och PSO är att låta en grupp individer samarbeta vid sökandet efter optimala lösningar till problem genom att de enskilda individerna delar med sig av sina erfarenheter till resten av gruppen. Gaussisk partikelsvärmsoptimering, GPSO, utvecklades 2003 för att tackla några funna svagheter hos tidigare PSO-metoder. GPSO visade sig vara en lovande förbättring gentemot tidigare PSO-metoder, men mer forskning ansågs behövas för att bekräfta GPSO-metodens effektivitet.

Den här uppsatsen ger en översikt över PSO och de vanligaste varianterna av PSO-algoritmer. Uppsatsen undersöker effektiviteten hos GPSO och andra vanliga PSO-metoder för ickelinjära begränsade flerdimensionella problem, och visar att GPSO inte alls medför någon förbättring överlag gentemot övriga undersökta PSO-metoder. Uppsatsen styrker även tidigare påvisad effektivitet hos PSO genom en jämförelse med en erkänd och bra optimeringsmetod som bygger på en annan teknik.

Particle swarm optimization

An overview and evaluation

Many problems are so complex that it is not possible to find an optimal solution within a reasonable time frame. Optimization is a technique that has always existed and that has become more and more common during the last decades to meet the demands of as close to optimal solutions as possible for complex problems. Particle swarm optimization, PSO, is a relatively new optimization method that was first presented in 1995. Since 1995 the interest for PSO has increased, mostly due to its shown efficiency when it comes to optimization, but possibly also due to the underlying theories of collective intelligence. The idea of collective intelligence and PSO is to let a group of individuals cooperate in the search for the optimal solutions to problems by letting each individual share their experiences with the rest of the group. Gaussian particle swarm optimization, GPSO, was developed in 2003 to address a few weaknesses found in previous PSO-methods. GPSO was shown to be a promising improvement over previous PSO-methods, but more research was said to be needed to confirm the efficiency of GPSO.

This dissertation presents an overview of PSO and the most common variants of PSO algorithms. The dissertation examines the efficiency of GPSO and other common PSO-methods for nonlinear constrained multidimensional problems, and shows that GPSO overall does not result in an improvement over other examined PSO-methods. The dissertation also verifies the previously shown efficiency of PSO by comparison to a good and recognized optimization method that is based on a different technique.

Tack

Författarna skulle framförallt vilja tacka Thijs Holleboom vid institution för informationsteknologi vid Karlstads universitet för sitt tålamod och den tid han avlagt för oss.

Författarna vill även rikta ett tack till Eva Mossberg vid instutionen för ingenjörsvetenskap, fysik och matematik vid Karlstads universitet för hennes givmildhet av tips och boklån.

Slutligen önskar författarna även tacka Andries P. Engelbrecht vid avdelningen för datavetenskap vid universitetet i Pretoria för besvarade frågor och förklaringar kring funna otydligheter i hans bok ”Computational Intelligence - An introduction”.

Innehåll

1 Inledning **1**

2 Optimeringslära **3**

2.1 Optimeringsprocessen 4

2.2 Linjär och ickelinjär optimering 5

2.3 Bivillkor 6

2.4 Diskreta vs kontinuerliga problem 7

2.5 Grundläggande komplexitetsteori 8

2.5.1 Algoritmkomplexitet 8

2.5.2 Problemkomplexitet 9

2.6 Tillämpningsområden 9

3 Kollektiv intelligens **13**

4 Svärmoptimering **15**

4.1 Fitnessfunktion 16

4.2 Partikelsvärmsoptimering 19

4.2.1 Omgivningsstrukturer 20

4.2.2 Algoritmer 22

4.3 Insektsvärmsoptimering 31

4.3.1 Kommunikation 32

4.3.2 Feromon i kontinuerliga rum 34

5 Experiment **37**

5.1 Experimentmiljö 38

5.1.1 MATLAB och FMINCON 38

5.2 Del I - återupprepning 39

5.2.1 Parameterinställningar 40

5.2.2 Testproblem 40

5.3 Del II - påbyggnad 41

5.3.1 Parameterinställningar 41

5.3.2 Testproblem 42

5.4 Testfunktioner 43

5.4.1 De Jongs Test Suite, $F_1 - F_5$ 43

5.4.2 F_6 Griewangk 48

5.4.3 F_7 Rastrigin 49

5.4.4 F_8 Schaffer 51

5.5 Resultat och diskussion 51

5.5.1 Del I - återupprepning 52

5.5.2 Del II - påbyggnad 55

5.5.3 Modellerar GPSO en partikelsvärm? 73

6 Slutsats och sammanfattande kommentarer **75**

Referenser **79**

A MATLAB kod **83**

A.1 Standard PSO 83

A.2 Adaptive PSO 86

A.3 PSO med ring-topologi 89

A.4 PSO med wheel-topologi 92
A.5 Gaussian PSO 95

Figurer

2.1 *Schematisk bild av optimeringsprocessen* 5

4.1 $f : t \longmapsto \mathbb{R}$ 16
4.2 *En-dimensionellt tillståndsrum* 17
4.3 *Star-topologi* 21
4.4 *Ring-topologi* 21
4.5 *Wheel-topologi* 22
4.6 *SPSO algoritmen med star-topologi* [15] 26
4.7 *Utökning av SPSO algoritmen till APSO* [15] 28
4.8 *Partikelförflyttning hos PSO* 29
4.9 *Utökning av SPSO algoritmen till GPSO* [3] 30
4.10 *Feromonföljande myror* 33
4.11 *Myror och delrum i ett två-dimensionellt tillståndsrum* 35

5.1 F_1 *Sphere av två dimensioner* 44
5.2 F_2 *Rosenbrock av två dimensioner* 45
5.3 F_2 *Rosenbrock av två dimensioner, närbild* 46
5.4 F_3 *Step av två dimensioner* 47
5.5 F_4 *Quartic av två dimensioner* 48
5.6 F_5 *Foxholes av två dimensioner* 49
5.7 F_6 *Griewangk av två dimensioner* 50

5.8 *F_7 Rastrigin av två dimensioner* 51
5.9 *F_8 Schaffer av två dimensioner* 52
5.10 *Illustration av GPSO applicerat på F_3* 57
5.11 *Konvergeringskurvor för F_1 Sphere* 60
5.12 *Konvergeringskurvor för F_2 Rosenbrock* 62
5.13 *Konvergeringskurvor för F_3 Step* 64
5.14 *Konvergeringskurvor för F_4 Quartic* 66
5.15 *Konvergeringskurvor för F_5 Foxholes* 68
5.16 *Konvergeringskurvor för F_6 Griewangk* 70
5.17 *Konvergeringskurvor för F_7 Rastrigrin* 72
5.18 *Konvergeringskurvor för F_8 Schaffer* 73

Tabeller

4.1 *Symbolbeskrivning för kommande matematiska beskrivningar av algoritmerna.* 23

5.1 *Parameterinställningar vid utförandet av delexperiment I.* 40
5.2 *Parameterinställningar vid utförandet av delexperiment II.* 42
5.3 *Undersökta problem i delexperiment II.* 42
5.4 *Nödvändigt antal iterationer.* 53
5.5 *Tidigare resultat av Secrest och Lamont.* [3] 54
5.6 *Resultat av upprepning.* 54
5.7 *Tabell över resultat för F_1 Sphere.* 59
5.8 *Tabell över resultat för F_2 Rosenbrock.* 61
5.9 *Tabell över resultat för F_3 Step.* 63
5.10 *Tabell över resultat för F_4 Quartic.* 65
5.11 *Tabell över resultat för F_5 Foxholes.* 67
5.12 *Tabell över resultat för F_6 Griewangk* 69
5.13 *Tabell över resultat för F_7 Rastrigrin.* 71
5.14 *Tabell över resultat för F_8 Schaffer.* 73

Kapitel 1

Inledning

Människan har alltid varit en problemlösare. Medan en del problem är väldigt vardagliga och enkla, är andra långt mer komplexa och därmed svårlösta. Trots att dagens teknik utvecklas i en rasande takt finns det än så länge ingen möjlighet att lösa de mer komplexa problemen exakt. De mer komplexa problemen kan idag endast lösas med hjälp av någon numerisk metod genom att successivt närma sig den exakta lösningen. Det går inte alltid att finna den exakta lösningen inom rimliga tidsramar, men det går ofta att komma väldigt nära. Under 1900-talet har flertalet optimeringsmetoder arbetats fram med avseendet att minimera tidsåtgången och finna allt exaktare lösningar vid lösning av komplexa problem. En av de nyare optimeringsmetoderna, kallad partikelsvärmsoptimering (Particle Swarm Optimization, PSO), utvecklades av Eberhart och Kennedy år 1995 [1]. Partikelsvärmsoptimering utnyttjar, till skillnad från tidigare metoder, den kollektiva problemlösningsförmågan hos en grupp individer [2]. Med kollektiv problemlösningsförmåga menas att en individ i en grupp av individer kan dra nytta av övriga gruppmedlemmars erfarenheter.

Inom området partikelsvärmsoptimering har en rad olika metoder utvecklats där det stora antalet bara innebär små förfiningar av originalmetoden, Standard PSO (SPSO). Barry R. Secrest och Gary B. Lamont utvecklade dock så sent som år 2003 en metod vilken innebar en ganska radikal förändring gentemot SPSO, Gaussian Particle Swarm

Optimization (GPSO) [3]. Tanken med GPSO var att tackla några svagheter hos tidigare PSO-metoder som upphovsmakarna identifierat. Upphovsmakarna till GPSO visade genom experiment att GPSO innebar en stor effektivitetsmässig förbättring gentemot SPSO hos fem icke-linjära, kontinuerliga och begränsade problem. De problem vilka användes i experimentet var dock endast av två dimensioner och den SPSO-algoritm som användes var på intet sätt optimal. Vad som menas med att SPSO-algoritmen inte var optimal är att ett antal enkla förbättringar vilka påverkar algoritmens effektivitet valts bort.

Målet med denna uppsats är att ge en överblick över området kollektiv intelligens, främst partikelsvärmsoptimering, och närmare undersöka hur GPSO står sig i jämförelse, i fråga om algoritmeffektivitet, med ett antal vanliga varianter av PSO, över ett större antal dimensioner än vad som tidigare undersökts.

Uppsatsen inleds med en introduktion till optimeringslära i kapitel 2 vilket kommer att ligga till grund för förståelsen av uppsatsen som helhet. Uppsatsen går sedan vidare med att allmänt beskriva kollektiv intelligens i kapitel 3 för att i kapitel 4 i detalj beskriva stora delar av området optimering med svärmar. I kapitel 5 beskrivs genomförandet av huvudmålet hos uppsatsen - jämförelsen mellan GPSO och andra vanliga varianter av PSO - och resultatet. Slutligen avslutas uppsatsen i kapitel 6 med slutsatser och en kort sammanfattning.

Kapitel 2

Optimeringslära

Optimeringsläran har ursprungligen sina rötter i tillämpad matematik och används idag vid olika typer av beslutsfattning. Från att i början ha använts för militära ändamål tillämpas optimeringsläran idag över ett stort antal tekniska och ekonomiska tillämpningsområden av vilka några beskrivs kortfattat i sektion 2.6. Optimeringsläran innefattar användandet av matematiska modeller och metoder med ändamålet att fatta så bra beslut som möjligt. Ett grundläggande krav vid applicering av en optimeringsmodell på ett problem är att det finns något i problemställningen som kan varieras, en så kallad variabel. Målet med optimering ur ett matematiskt perspektiv är således att söka de värden på problemets variabler vilka maximerar eller minimerar en målfunktion[1]. Målfunktionen är i många fall bara en uppskattning av det verkliga problemet som skall lösas eftersom det ofta är svårt att modellera ett problem fullständigt, och optimering bör i dessa fall endast betraktas som en uppskattning [4]. [5]

I sektion 2.1 behandlas själva optimeringsprocessen vilka samtliga modeller utgår ifrån. I sektion 2.2 beskrivs linjär och ickelinjär optimering och skillnaden mellan dessa. Sektion 2.3 och sektion 2.4 tar upp bivillkor till problem respektive skillnaden mellan diskreta

[1]Målfunktionen, i vissa fall kallad fitnessfunktion (sektion 4.1), är en modell av problemet som skall optimeras.

och kontinuerliga variabler. Sektion 2.5 tar upp grundläggande komplexitetsteori för såväl algoritmer såsom problem och kapitlet avslutas genom att i sektion 2.6 ta upp några tillämpningsområden för optimering.

2.1 Optimeringsprocessen

En stor del av denna sektion är hämtad ur boken *Linjär och ickelinjär optimering* av Jan Lundgren, Mikael Rönnqvist och Peter Värbrand [5].

Optimeringsprocessen är en generell arbetsgång som tillämpas vid modellering och lösning av optimeringsproblem. Huvudsyftet med att arbeta med en optimeringsprocess är att skapa underlag och riktlinjer vid beslut rörande det aktuella problemet. Optimeringsprocessen leder till en systematisk strukturering av problemställningen bestående av olika faser, steg, vilka illustreras i Figur 2.1.

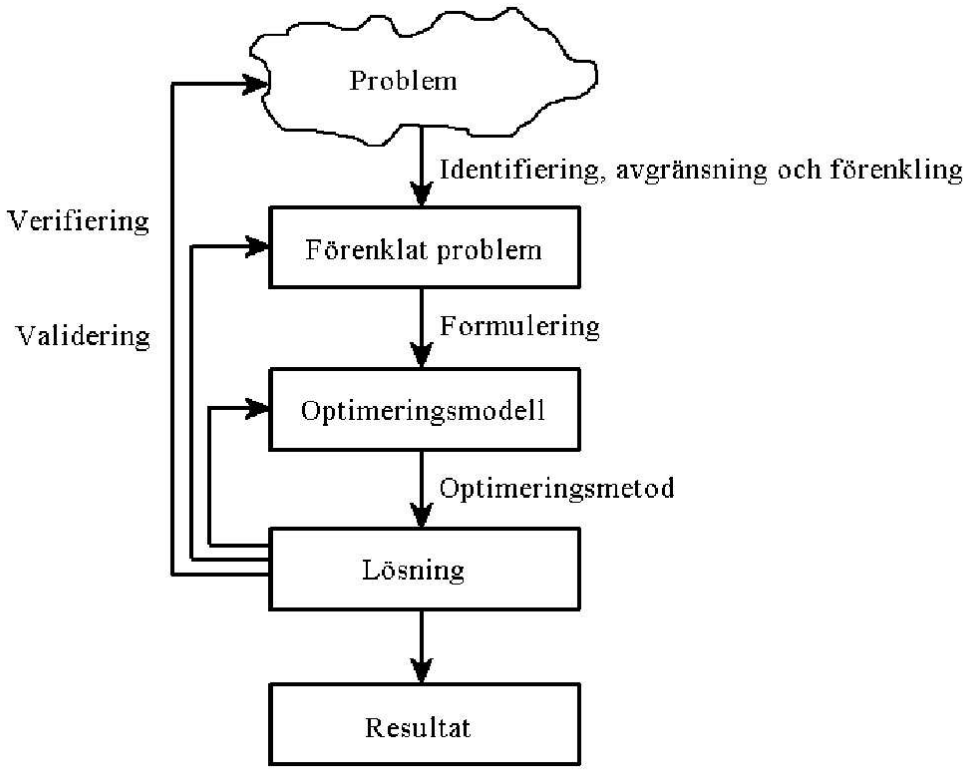
Givet ett problem som i många fall kan vara oerhört komplext är första steget att identifiera, avgränsa och förenkla problemet och göra det mer hanterbart. Första fasen kan således bestå i att minska detaljrikedomen och ta bort faktorer som inte kan inkluderas eller har mindre betydelse, för att på så sätt begränsa antalet påverkade faktorer. Den minskade detaljrikedomen i det förenklade problemet är således en avvägning mellan mot den slutgiltiga realismen och användbarheten hos resultatet.

Fas två går ut på att ta fram en matematisk modell genom en matematiskt formulering av det förenklade problemet. Den matematiska formuleringen involverar identifiering av variabler, målfunktion och eventuella bivillkor. Bivillkor tas upp i sektion 2.3.

I fas tre appliceras en optimeringsmetod på den tidigare framtagna modellen. Programvaran som implementerar optimeringsmetoden är i många fall någon kommersiell programvara - exempel är MATLAB, CPLEX och OSL - men ibland krävs framtagandet av egenutvecklad programvara för att på så sätt utnyttja kända egenskaper hos problemet.

I fas fyra gäller det att validera att modellen speglar det ursprungliga problemet på ett

tillfredsställande sätt samt att utifrån den erhållna lösningen verifiera lösningens korrekthet i förhållande till optimeringsmodellen.



Figur 2.1: *Schematisk bild av optimeringsprocessen*

2.2 Linjär och ickelinjär optimering

För att ett problem skall klassas som linjärt ska målfunktionen vara linjär men även problemets bivillkor. En funktion z är linjär om den kan skrivas som

$$z = \sum_{i=1}^{n} c_i x_i$$

då

$$\begin{aligned} \sum_{i=1}^{n} A_{ji}x_i \leq b_j,\ j &= 1,\ldots,m \\ x_i &\geq 0,\ i = 1,\ldots,n \end{aligned}$$

där c_i är koefficienten i målfunktionen för x_i, A_{ji} är bivillkorskoefficienten för variabeln x_i i bivillkor j och b_j är högerledskoefficienten i bivillkor j. Generellt för linjära problem gäller att varje lokalt optimum även är ett globalt optimum, vilket kommer utav att en linjär funktion alltid är både konvex[2] och konkav[3]. [4]

Till skillnad från linjära problem kan inte ickelinjära problem skrivas på en generell form, då utseendet på ickelinjära problem beror på hur de ickelinjära sambanden är formulerade. På grund av detta finns det heller ingen generell metod för att lösa denna klass av problem. I fallet med ickelinjära problem måste således en lösningsmetod anpassas efter det problem metoden ämnas appliceras på. [5]

Eftersom ickelinjära problem inte kan skrivas på en generell form, delas ickelinjära problem ofta vidare in i olika typer av ickelinjära problem. Ett grovt sätt att kategorisera ickelinjära problem är huruvida de är begränsade, det vill säga har bivillkor, eller obegränsade. [5]

2.3 Bivillkor

Bivillkor definierar begränsningar hos variablerna $\vec{x}$ för målfunktionen $f(\vec{x})$. Ett minimeringsproblem uttryckt med bivillkor kan då formuleras

$$min(f(\vec{x})),\ \text{begränsad av} \begin{cases} c_i(\vec{x}) = 0, & i \in \Theta \\ c_i(\vec{x}) \geq 0, & i \in \Omega \end{cases}$$

[2] $f(\vec{x})$ är en konvex funktion om det för varje val av punkter $\vec{x}_1$ och $\vec{x}_2$ som tillhör det tillåtna området och då $0 \leq \lambda \leq 1$ gäller att $f(\lambda\vec{x}_1 + (1-\lambda)\vec{x}_2) \leq \lambda f(\vec{x}_1) + (1-\lambda)f(\vec{x}_2)$

[3] $f(\vec{x})$ är en konkav funktion om det för varje val av punkter $\vec{x}_1$ och $\vec{x}_2$ som tillhör det tillåtna området och då $0 \leq \lambda \leq 1$ gäller att $f(\lambda\vec{x}_1 + (1-\lambda)\vec{x}_2) \geq \lambda f(\vec{x}_1) + (1-\lambda)f(\vec{x}_2)$

där c_i är ett specifikt bivillkor, Θ är mängden av samtliga index i där c_i representerar en likhetsbegränsning och Ω är mängden av samtliga index i där c_i representerar en olikhetsbegränsning. [5]

Alla likhetsbegränsningar samt de olikhetsbegränsningar vilka är uppfyllda med likhet i en given punkt kallas *aktiva begränsningar* i den punkten och brukar betecknas A. A_i motsvarar då en specifik aktiv begränsning. [4]

Som ett enkelt exempel antag

$$min(f(\vec{x}))\ ,\ f(\vec{x}) = 3x_1 + 5x_2\ ,\ \vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\ ,\ \text{begränsad av} \begin{cases} x_1^2 - x_2 = 0 \\ x_1 + x_2 \leq 2 \end{cases}$$

vilket ger

$$c(\vec{x}) = \begin{bmatrix} c_1(\vec{x}) \\ c_2(\vec{x}) \end{bmatrix} = \begin{bmatrix} x_1^2 - x_2 \\ -x_1 - x_2 + 2 \end{bmatrix}\ ,\ \Theta = \{1\}\ ,\ \Omega = \{2\}$$

Antag vidare att samtliga begränsningar är aktiva så ger det

$$A = \begin{bmatrix} x_1^2 - x_2 \\ x_1 + x_2 \end{bmatrix}\ ,\ \vec{b} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

2.4 Diskreta vs kontinuerliga problem

Då målfunktionens samtliga variabler $x_i \in \mathbb{R}$ sägs problemet vara kontinuerligt. På motsvarande sätt sägs problemet vara diskret i de fall då samtliga variabler $x_i \in \xi$ där ξ representerar en ändlig mängd. I de fall där $\xi = \{0, 1\}$ benämns problemet istället som ett kombinatoriskt problem och då målfunktionen består av variabler av varierande karaktär benämns problemet som ett blandat heltalsproblem, eller på engelska mixed integer problem. [6]

2.5 Grundläggande komplexitetsteori

Komplexitetsteori ämnar kategorisera problemtyper efter deras lösbarhet. Inom komplexitetsteorin skiljs det på algoritm- och problemkomplexitet, där algoritmkomplexitet avser uppskatta förändringen i beräkningstid vid förändring av problemstorlek medan problemkomplexitet ämnar kategorisera själva problemet efter komplexiteten att finna en lösning. [5]

Algoritmkomplexitet tas upp i sektion 2.5.1 och problemkomplexitet i sektion 2.5.2.

2.5.1 Algoritmkomplexitet

Vid framtagandet av en algoritm är det önskvärt med ett mått på dess effektivitet för att möjliggöra jämförelser gentemot existerande algoritmer. Algoritmkomplexitet används i syftet att ge ett mått på förändringen i en algoritms beräkningstid när problemstorleken förändras, vilket kan användas vid jämförelser mellan olika algoritmer.

En ordofunktion, O, används för att ange en övre gräns för antalet beräkningssteg[4] hos en algoritm i förhållande till storleken på det aktuella problemet som krävs. Genom att uttrycka beräkningstiden som en funktion av problemstorleken, $O(f(n))$, erhålls kunskap om algoritmens uppförande då problemstorleken växer, eftersom beräkningstiden är proportionell mot antalet beräkningssteg. Det exakta antalet beräkningssteg är oftast inte relevant så $f(n)$ förenklas vanligtvis till den mest betydelsefulla termen då $n \to \infty$. Vidare gäller att $f(n) = O(g(n))$ för två funktioner $f(n)$ och $g(n)$ om det finns en konstant c sådan att $f(n) = cg(n)$ då $n \to \infty$. [5]

Antag

$$f(n) = 6n^4 + 100n^3 + 2n$$

vilket ger

$$\lim_{n\to\infty} f(n) \approx \lim_{n\to\infty} 6n^4 \Rightarrow g(n) = n^4$$

[4] Summan av samtliga matematiska operationer som krävs vid lösandet av ett specifikt problem.

Således sägs en algoritm som kräver $f(n)$ beräkningssteg ha komplexiteten $O(n^4)$.

2.5.2 Problemkomplexitet

En stor del av denna sektion om problemkomplexitet är hämtad ur boken *Linjär och ickelinjär optimering* av Jan Lundgren, Mikael Rönnqvist och Peter Värbrand [5].

Problem kan delas upp i olika klasser efter deras komplexitet, hur svårlösta de är. Klassificeringen sker i regel genom att betrakta en transformation av problemet, ett så kallat *igenkänningsproblem*. Hur igenkänningsproblem ser ut och formuleras är inget som tas upp i denna uppsats, men det kan sägas att ett optimeringsproblem generellt sett inte är svårare att lösa än dess motsvarande igenkänningsproblem och optimeringsproblem klassificeras i regel efter hur svårlösta deras respektive igenkänningsproblem är.

Den första klassen av problem kallas lätta problem och betecknas P, där P kommer från engelskans *polynomial*. Problem av klassen P karakteriseras av att problemen kan lösas med hjälp av en algoritm av polynomisk komplexitet.

En annan klass av problem är klassen NP, där P ingår och NP står för *nondeterministic polynomial*, som består av alla optimeringsproblem vars lösningar kan verifieras med hjälp av en algoritm av polynomisk komplexitet. Att en lösning kan verifieras innebär att det är möjligt att avgöra om lösningen är bättre än en annan lösning.

De svåraste optimeringsproblemen tillhörande klassen NP kallas *NP-svåra*, eller *NP-hard*, och är problem som inte kan bevisas vara lösbara med hjälp av en algoritm av polynomisk komplexitet.

2.6 Tillämpningsområden

Optimering används frekvent i dagens teknologiska samhälle i allt från planering och logistik till att maximera en ekonomisk vinst. Eftersom antalet tillämpningsområden är stort begränsas denna sektion till att endast kort ta upp ett fåtal specifika tillämpningsområ-

den med syftet att läsaren själv ska få upp ögonen för möjliga användningsområden. Valda tillämpningsområden i denna sektion är hämtade ur boken *Linjär och ickelinjär optimering* av Jan Lundgren, Mikael Rönnqvist och Peter Värbrand [5].

- **Tillverkning av kretskort**

  Vid kretskortstillverkning används en uppsjö av olika komponenter vilka varje tillverkningsmaskin endast kan hålla att begränsat antal av vid ett tillfälle. Det är också vanligt att en och samma maskin tillverkar en rad olika kretskort under en och samma tidsperiod. På grund av att komponentbyte i en maskin är förknippat med en tidsförlust ämnar optimeringsprocessen bestämma den ordning vilken korten tillverkas i och det sätt byte av komponenter sker på som minimerar den totala tillverkningstiden.

- **Borrning av mönsterkort**

  Problemet med borrning av mönsterkort kan betraktas som problemet att hitta den kortaste vägen vilken förbinder samtliga borrpunkter på mönsterkortet. Då tiden för att borra ett hål i stort sett är konstant medför det att den totala tiden då det inte borras är proportionell mot hur lång sträcka borren behöver förflyttas mellan hålen.

- **Schemaläggning**

  Problemet med schemaläggning återkommer frekvent inom många olika områden. En av de mest uppenbara kanske är skolscheman där lektioner med tillhörande lärare, elever, salar och tider ska kombineras på ett effektivt sätt. Lärare och elever kan endast befinna sig på en plats samtidigt, salar får inte dubbelbokas och kursernas timplaner skall efterföljas. Samma typ av problem återfinns bland annat vid bemanningsplanering, planering av flygrutter samt vid ruttplanering av frakt.

- **Design av telenät**

  Vid konstruktion av fasta telenät räcker det inte att beakta den för stunden bästa lösningen utan även framtida behov bör undersökas. Efter en approximation av framtidens behov gäller det att avgöra hur nätets knutpunkter ska anslutas för att

minimera kostnaderna och samtidigt kunna klara av det framtida behovet. För luftburen kommunikation består problemet istället av utplacering av basstationer och bestämmandet av antalet sändare som behövs. Ytterligare ett problem som blir gällande i mobila telenät är fördelningen av frekvensband med målet att minimera störningar från närliggande basstationer. Med minimal störning kommer maximal genomströmmning av teletrafik att kunna erhållas.

- **Styrning av trafiksignaler**

  För att styra trafiken samt öka framkomligheten på vägarna i städer används trafiksignaler. Vid planeringen av trafiksignalerna ingår bland annat att bestämma intervallen för röd/grön-faserna och i vilken ordning trafiken från olika riktningar i en korsning ska passera. Med en god planering kan trafikanterna uppleva en så kallad "grön våg".

Kapitel 3

Kollektiv intelligens

Intelligens är ett begrepp vars betydelse och innebörd länge har varit, och fortfarande är, ett hett diskussionsämne bland forskare världen över. Uppslagsböcker beskriver ofta ordet intelligens som förmågan att tänka, resonera, förstå, analysera och kunna dra nytta av tidigare erfarenheter. Men även om det finns definitioner av intelligens i uppslagsböcker, finns det ingen vedertagen vetenskaplig betydelse av intelligens [7].

När de första datorerna kom, började omedelbart paralleler dras mellan den mänskliga hjärnan och datorprogram. Datorer kan lagra information för senare återkallande, bearbeta symbolisk information, erhålla slutsatser från premisser, och så vidare - sådant som den mänskliga hjärnan också kan göra. Således borde en dator kunna vara intelligent om en människa kunde vara intelligent, och därmed var den artificiella intelligensen född. [2]

En definition av Artificiell Intelligens (AI) vilken gavs vid en IEEE konferens angående neurala nätverk år 1996 säger att AI är forskning kring hur datorer skall utföra sådana uppgifter vilka människor är bättre på [8]. Traditionellt sett handlar AI ofta om fristående enheter vilka besitter en viss problemlösningsförmåga och som agerar helt individuellt vid problemlösning.

Det är först på senare år som kollektiv intelligens börjat utforskas på allvar. Svärmintelligens (Swarm Intelligence, SI) är ett område inom AI som bygger på just kollektiv

intelligens. Forskning inom SI bygger inte så mycket på individens egen intelligens utan är mer centraliserad kring hur en grupp, ofta kallad en svärm, individer samarbetar för att lösa en specifik uppgift. Intelligensen har då höjts en nivå från att bara ha innefattat en individ till att nu inkludera samtliga individer i en svärm och deras samarbetsförmåga. En enskild människas bedrift kan gynna andra människor och på så sätt höja kunskapsnivån hos oss som en grupp. Till exempel kommer de flesta människor aldrig att bli någon ny Einstein men många kan dra nytta av hans upptäckter. På liknande sätt ämnar en individ i en svärm inte bara använda sin egen kunskap utan även dra nytta av information frambringad av övriga svärmmedlemmar.

Då en människa och hennes sociala beteenden är svåra att simulera har en stor del av dagens forskning kring svärmar avsett insekter, fiskar och fåglar. Trots att en insekt kanske bara har några hundra hjärnceller kan de som en grupp åstadkomma en mängd komplexa uppgifter [2]. En koloni av sociala insekter kan ses som ett decentraliserat system för problemlösning, där varje individuell insekt specialiserar sig på någon eller några uppgifter. En individuell insekts uppgifter kan till exempel bestå av att hitta föda, städa boet eller angripa inkräktare, medan kolonin som helhet utför uppgifter såsom att bygga ut boet, dela upp födan mellan de individuella insekterna och försvara boet. [9]

Överlevnadschanserna för en enskild fisk ökar om den tillhör ett stim. Ett stim av fiskar har fler ögon än en ensam fisk och kan på så sätt upptäcka eventuella faror snabbare. Ett stim ger även ökad säkerhet för individuella fiskar vilka befinner sig nära stimmets centrum och denna fördel är en faktor som hjälper till att hålla ihop stimmet [2]. Detta beteende att sträva efter en central placering kan även skådas hos en flock fåglar och har inspirerat till partikelsvärmsoptimering där enskilda partiklar dras mot en bättre position inom svärmen [8].

SI kan sägas ämna simulera en mängd enkla enheter med förmågan att kommunicera för att på så vis bilda en svärm med kollektivt tänkande i syftet att lösa ett gemensamt problem.

Kapitel 4

Svärmoptimering

Optimering är inte entydigt definierad, men beskrivs generellt sett som en process vilken utifrån någon aspekt åstadkommer en förbättring. För många problem existerar i regel många tänkbara lösningar, som kan sägas lösa problemen olika bra. Optimering kan då sägas vara processen att finna en lösning på det aktuella problemet som uppfyller de ställda kraven. Att inte den absolut bästa lösningen, den optimala lösningen, eftersöks kan ha många olika orsaker. En vanlig orsak är att medan antalet påverkande faktorer ökar linjärt i ett problem tenderar antalet möjliga lösningar ofta växa exponentiellt. Problemen blir helt enkelt för stora och det skulle ta alldeles för lång tid att ta fram den optimala lösningen. Som ett exempel kan nämnas att sökrummet för Symmetric Travelling Salesman Problem[1] (STSP) med 48 städer innehåller $\frac{(48-1)!}{2} \approx 1.29 \cdot 10^{59}$ olika giltiga turer eller lösningar. Om det är möjligt att evaluera en miljon turer per sekund, skulle det då ta i storleksordnigen 10^{45} år att undersöka alla giltiga turer.

Optimering ur denna uppsats synvinkel går ut på att söka de input värden vilka maximerar eller minimerar resultatet av en funktion, den så kallade fitnessfunktionen vilken kommer att behandlas i kapitel 4.1. Vidare kommer varje lösning, giltig uppsättning inputvärden till fitnessfunktionen, för ett problem att betraktas som ett tillstånd för

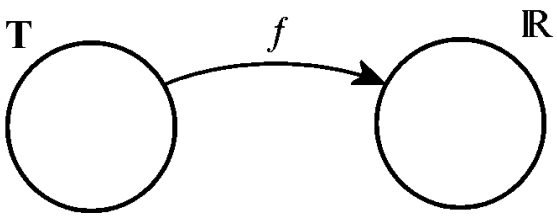
[1] Symmetric innebär här att avståndet mellan två städer är detsamma oberoende av färdriktning.

problemet. Mängden av alla tillstånd för ett problem kommer slutligen att refereras till som problemets tillståndsrum. Fitnessfunktionen är då en mappning mellan tillståndsrummet och de reella talen för att på så sätt möjliggöra utvärdering, och därmed jämförelse, av olika tillstånd.

En svärm kan definieras som en löst sammankopplad mängd av interagerande enheter, ofta refererade till som agenter [2]. Idén bakom optimering med hjälp av svärmar är att låta agenterna utbyta information och erfarenheter, och på så vis lösa det aktuella problemet gemensamt. Sättet agenterna utbyter information på, varierar beroende på vilken typ av algoritm det rör sig som. Det finns två huvudsakliga grupper av algoritmer inom optimering med svärmar: partikelsvärmsoptimering, som tas upp i kapitel 4.2, och insektsvärmsoptimering, som tas upp kortfattat i kapitel 4.3.

4.1 Fitnessfunktion

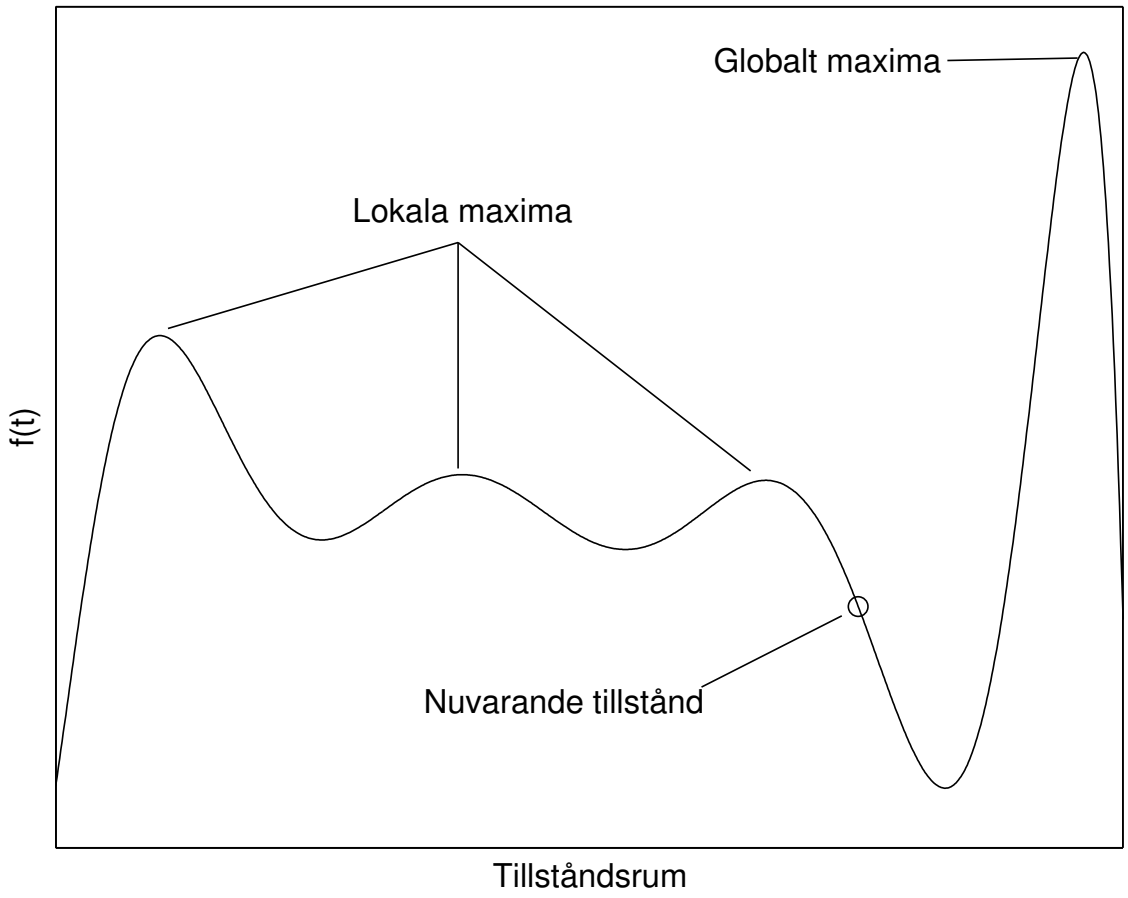
Fitnessfunktionen, $f(t)$, avbildar ett tillstånd, t, tillhörande domänen T på ett reellt tal tillhörande kodomänen, vilket visas i Figur 4.1.



Figur 4.1: $f : t \longmapsto \mathbb{R}$

Fitnessfunktionens reella värde används inom optimering för att avgöra hur bra ett specifikt tillstånd är. Optimering kan då betraktas som ett försök att finna ett tillstånd med ett så bra fitnessvärde som möjligt eller ett försök att hitta det globala maximet,

alternativt globala minimet, för fitnessfunktionen.



Figur 4.2: *En-dimensionellt tillståndsrum*

I Figur 4.2 illustreras tillståndsrummet för ett problem i horisontellt led och i vertikalt led visas tillståndens fitnessvärden. Vid optimering med partikelsvärmar, insektssvärmar eller andra lokala sökmetoder, eftersöks det globala maximet, eller i vissa fall det globala minimet, då det representerar den optimala lösningen. Tillståndsrummet i Figur 4.2 är synnerligen enkel och speglar få problem då varje tillstånd här endast har två intilliggande tillstånd[2], men exemplet fungerar bra för att illustrera principen. Då antalet dimensioner

[2]Ett intilliggande tillstånd till ett tillstånd T är ett tillstånd med samma utseende som T men med någon grundläggande förändring införd. Denna förändring är problem- och representationsspecifik.

i fitnesslandskapet motsvarar antalet påverkande faktorer i problemet, tenderar många verkliga problem att vara svårillustrerade då de i regel har ett stort antal påverkande faktorer.

Vid definition av fitnessfunktionen krävs en djup insikt om optimeringsalgoritmens arbetsätt samt en god förståelse för själva problemet. Det är vidare viktigt att funktionen är effektiv i avseende på korrekthet, men även i avseende på exekveringstid. Ett sätt att minska tiden som spenderas på evaluering av fitnessfunktionen är att använda så kallad inkomplett evaluering, där evalueringen endast är ett uppskattat eller relativt värde. Enda kravet vid inkomplett evaluering är att algoritmen forfarande kan skilja ett sämre tillstånd från ett bättre. [10]

I många fall ger fitnessfunktionen för litet utslag vid en förbättring av ett tillstånd. Flera metoder för att skala om fitnessvärdet existerar och de tre vanligaste är linjär skalning, sigma trunkering samt exponentiell skalning. [11]

Linjär skalning

$$f_i' = a * f_i + b$$

där a och b är konstanter ofta valda på så sätt att det nya fitnessvärdet speglar större förändringar vid nära optimala lösningar. Ett problem är att linjär skalning kan införa negativa fitnessvärden vilket leder till specialfall.

Sigma trunkering

Sigma trunkering är en vidareutveckling av linjär skalning där negativa värden undviks, matematiskt ger det

$$f_i' = f_i + (\overline{f} - \alpha * \sigma)$$

där α är ett litet heltal, σ är standardavvikelse för svärmens samtliga agenter och $\overline{f}$ är svärmens genomsnittliga fitnessvärde. Eventuella negativa fitnessvärden sätts

vanligtvis till noll.

Exponentiell skalning

$$f_i' = f_i^k$$

där k är ett reellt tal större än ett.

4.2 Partikelsvärmsoptimering

Idén bakom partikelsvärmsoptimering (PSO) härstammar från försök att simulera fåglars sociala beteende inom en flock, med målet att finna mönster i flockens kollektiva beteende som möjliggör synkroniserat flygande [8]. Inom PSO kallas en fågel för en partikel, och flocken refereras till som en svärm. Sett ur en matematisk och fysikalisk synvinkel saknar en svärmmedlem både massa och volym och borde således refereras till som en punkt. En punkt är däremot inte associerad till en förflyttning vilket är orsaken till att den inom PSO refereras till som en partikel [2].

Vid optimering med hjälp av partikelsvärmar rör sig svärmens partiklar genom det flerdimensionella sökrummet i sökandet efter en tillfredsställande lösning. Partiklarnas rörelsemönster påverkas inte bara av sina egna erfarenheter utan även av sina svärmmedlemmar. Att partiklarna påverkas av övriga svärmmedlemmars erfarenheter resulterar i vad som kallas för kollektiv intelligens, vilket har visat sig både effektivt och robust vid lösning av flertalet optimeringsproblem. Partiklar vilka fastnar på lokala optima kommer tack vare den kollektiva intelligensen att attraheras av övriga partiklar och på så sätt undvika att stagnera i sitt sökande. [8]

4.2.1 Omgivningsstrukturer

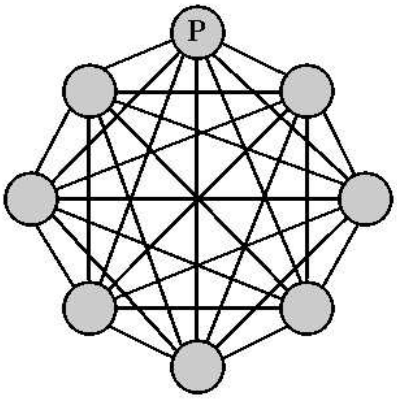
För att en individ, agent så väl som människa, ska kunna dra nytta av och kunna lära sig av en likasinnad är ett nödvändigt krav att det existerar någon form av kommunikationslänk mellan de båda. Varje enskild individ kan sägas ha en omgivning, en mängd individer, med vilka individen kan kommunicera. En individs omgivning behöver inte nödvändigtvis innefatta alla existerande individer, och en viss individ behöver inte vara i närheten av en annan individ för att tillhöra dess omgivning. En människas omgivning kan till exempel vara dess familj. Kommunikationslänkarna inom familjen fungerar ofta även om enskilda familjemedlemmar befinner sig långt ifrån varandra. Forskning har de senaste 60 åren visat att omgivningens struktur påverkar kommunikationen inom en grupp, och även gruppens prestationer [2].

Då det har visat sig att en partikel som saknar omgivning, dvs inte kan kommunicera med någon annan partikel, fastnar vid första bästa lokala optima är det lätt att inse att styrkan hos PSO ligger i svärmens kommunikationsförmåga [2][8]. Inom PSO-området har det studerats en rad enkla typer av omgivningsstrukturer, däribland star, ring samt wheels topologier.

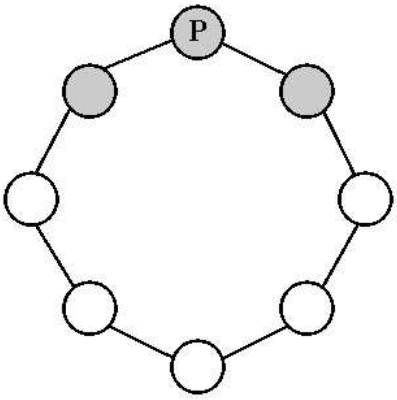
I star-topologin kan varje partikel kommunicera med alla andra partiklar i svärmen, vilket leder till ett fullt kopplat nätverk vilket illustreras i Figur 4.3. En partikels omgivning består alltså av hela övriga svärmen, och således kan varje partikel påverkas av alla andra partiklar. Star-topologin var den första omgivningsstruktur som användes inom PSO. [8]

Ring-topologin, som även går under namnet circle, innebär att en partikel endast kan kommunicera med n andra partiklar, där n är en parameter till algoritmen [8]. Den begränsade kommunikationen medför att delar av svärmen är skiljd från andra delar, vilket leder till att medan partiklarna inom vissa delar av svärmen konvergerar mot lokala optimum, så kan partiklar inom andra delar fortsätta sökandet [2]. Ring-topologin då $n = 2$ för en partikel P illustreras i Figur 4.4.

I wheel-topologin, vilken illustreras i Figur 4.5, kan endast en så kallad fokalpartikel



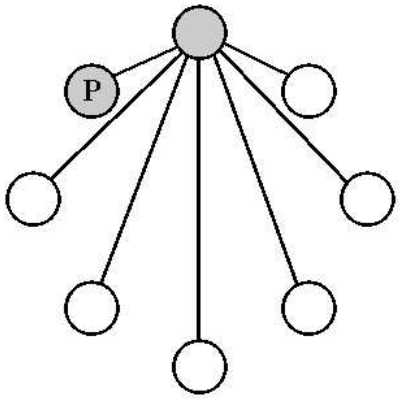
Figur 4.3: *Star-topologi*



Figur 4.4: *Ring-topologi*

kommunicera med alla partiklar i svärmen, medan övriga partiklar endast kan påverkas av fokalpartikeln. Fokalpartikeln justerar sin position utefter den bästa partikeln i svärmen, och om fokalpartikeln förflyttning resulterar i en förbättring vidarebefordras förbättring till alla andra partiklar. [8][2]

Det finns även andra topologier, till exempel där en partikels omgivning beror på avståndet till övriga partiklar i tillståndstummet. Till exempel kan omgivningen till en partikel



Figur 4.5: *Wheel-topologi*

i bestå av de partiklar som ligger inom ett begränsat avstånd till partikel i, vilket kallas geografisk topologi. Effekten av en geografisk topologi är att omgivningarna är små sett till antalet partiklar vid algoritmens början och växer allteftersom partiklarna närmar sig varandra. [8]

Beroende på valet av topologi kommer svärmen uppvisa olika typer av beteende. Det har visat sig att en star-topologi generellt sett konvergerar mot ett lokalt optimum snabbare än vad en ring-topologi skulle göra [2]. Anledningen till den snabbare konvergensen är att en svärms samtliga partiklar i ett tidigare skede blir uppmärksammade på ett nytt och bättre tillstånd. Vidare påverkar storleken av n i ring-topologin konvergeringshastigheten på så sätt att ett större n medför snabbare konvergering. Enligt Kennedy och Eberhart fungerar fokalpartikeln i wheel-topologin som en buffert vilken fördröjer konvergeringen mot lokala optima [2].

4.2.2 Algoritmer

Den första PSO algoritmen utvecklades av Eberhart och Kennedy i mitten av 90-talet[1], och sedan dess har ett flertal olika varianter utvecklats. Vi ämnar i denna sektion presentera

en rad förekommande varianter av PSO algoritmer med hjälp av pseudokod. I ett försök att minimera redundant kod krävs det av läsaren att hon eller han är väl förtrogen med hela kapitlet då efterföljande sektioner är vidareutvecklingar av föregående. För att förenkla för läsaren har vi i Tabell 4.1 gett en kortfattad beskrivning av förekommande symboler i algoritmerna.

Symbol	*Beskrivning*
$f(x)$	Fitnessvärde för partikel x
X_i	Vektor som representerar partikel i
$X_{i,j}$	Partikel i:s läge i dimension j
$V_{i,j}$	Partikel i:s hastighet i dimension j
$pbest_i$	Vektor som representerar partikel i:s bästa position hittils
$pbest_{i,j}$	Partikel i:s läge i dimension j i dess bästa position hittils
$gbest$	Indexet för den partikel med bäst $pbest$
N	Antalet partiklar i svärmen
D	Antalet dimensioner i problemet
n	Omgivningsstorlek för ring-topologi
$Rmin_j$	Minsta möjliga värde för en partikels läge i dimension j
$Rmax_j$	Största möjliga värde för en partikels läge i dimension j
$Vmax_j$	Maximal fart för en partikel i dimension j, vanligtvis $10 - 20\%$ av $(Rmax_j - Rmin_j)$. [12][13]
w	Inertia weight, $w \in [0, 1]$. [8]
c_1, c_2	Accelerationskonstanter som viktar betydelsen av en partikels egna erfarenheter gentemot omgivningens för partikelns rörelse. $c_1 + c_2 \leq 4$ där c_1 och c_2 är positiva[8].
Δf	Gräns för att en partikel ska räknas som inaktiv.
I_i	Inaktivitetsräknare för partikel i
T_c	Största tillåtna värde på I_i för alla i innan en partikel byts ut
$GRand(s)$	Returnerar slumptal med medelvärde 0 och standardavvikelse s

Tabell 4.1: *Symbolbeskrivning för kommande matematiska beskrivningar av algoritmerna.*

Samtliga algoritmer i denna sektion beskrivs med star-topologi men skulle lika gärna kunna implementeras med hjälp av någon annan topologi. Den stora skillnaden är i sådana fall att $gbest$ byts ut mot $lbest_i$, det hittils bästa tillståndet i omgivningen för partikel i.

Parametrar

- **Dimension**

  Dimensionen D av ett problem är det samma som antalet påverkande faktorer. Låt säga att det aktuella problemet går ut på att maximera värdet av funktionen g som beror på vektorn $\vec{x}$ enligt följande

  $$g(\vec{x}) = \sum_{n=1}^{D} x_i \ , \ x_i \in [Rmin_i, Rmax_i]$$

  Då gäller att D är dimensionen av problemet.

- **Hastighet**

  Skillnaden mellan fart och hastighet är att hastigheten även anger en riktning. Hastigheten när vi talar om PSO är alltså avgörande för i vilken riktning partikeln rör sig i tillståndsrummet samt med vilken fart den färdas. En partikels hastighet påverkas av förhållandet mellan det bästa tillståndet partikeln har erhållit hittils, omgivningens bästa tillstånd hittils och partikelns nuvarande tillstånd. Hastigheten i en dimension j begränsas enklast av ett maxvärde, $Vmax_j$, för att förhindra att partiklar antar för höga hastigheter. En för hög hastighet kan leda till att en partikel missar bra tillstånd, då hastigheten i grunden är ett mått för hur långa "hopp" partikeln gör i tillståndsrummet. [13]

  Då dimensionen för ett problem är större än ett krävs det också att hastigheten representeras genom en vektor av samma dimension som problemet, det vill säga en vektor av D element där D är problemets dimension.

- **Inertia weight**

  Inertia weight, eller tröghet, w, är en partikelns förmåga att bevara nuvarande riktning och hastighet. Vid beräkning av nya hastigheter används inertia weight för att vikta hur stor betydelse den gamla hastigheten ska ha på den nya. Ett större värde

på inertia weight leder till utforskning av en större del av tillståndsrummet, medan ett mindre värde leder till mer lokala utforskningar. Vanligtvis startas PSO med ett högt värde på inertia weight som sedan avtar som en funktion av tiden. Inertia weight begränsas till intervallet $[0, 1]$ för att möjliggöra konvergens. [8][14]

- **Svärmstorlek**

  Svärmstorleken N anger antalet partiklar som finns i svärmen och är problemspecifik [13]. För läsare som är bekanta med Genetiska Algoritmer (GA), kan dock sägas att svärmstorleken i PSO generellt sätts till ett mindre värde än populationsstorleken i GA för samma problem. Tidigare experiment på PSO har indikerat att svärmstorlekar mellan 10-50 partiklar oftast fungerar tillfredställande. [2]

Standard PSO

SPSO algoritmen går ut på att förflytta samtliga partiklar i svärmen ett antal gånger tills önskat resultat erhållits. En enskild partikels förflyttning beror på partikelns hastighet, där hastigheten beror på det historiskt bästa tillståndet för partikeln själv och det historiskt bästa tillståndet för alla partiklar sammantaget. Med tiden kommer varje enskild partikels historiskt bästa tillstånd hamna allt närmare svärmens historiskt bästa tillstånd, vilket leder till att partiklarna kommer röra sig med allt mindre steg och till konvergens. Varje partikel har en associerad hastighet som bestämmer hur långa steg partikeln ska ta vid varje förflyttning och i vilken riktning förflyttningen ska ske. Hastigheten för en partikel är definierad som förändringen i dess position per förflyttning, och är därmed endast indirekt baserad på tiden. Algoritmen för Standard PSO (SPSO) visas i pseudokod i Figur 4.6.

Algoritmen för SPSO initierar ett N antal partiklar av D dimensioner vilka kommer att utgöra själva svärmen (4.1-4.3). Initieringen bygger på att tilldela varje unik partikel ett starttillstånd, tillhörande problemets tillståndsrum, vilket också blir partikelns *pbest*, en flygriktning och dess fart med vilken partikeln rör sig genom tillståndsrummet. Hastighet och position i en specifik dimension för en partikel kan slumpmässigt spridas inom de

For all $(i,j) \in \{i \times j,\ i \in \{1,2,\ldots,N\},\ j \in \{1,2,\ldots,D\}\}$

$$X_{i,j} = x\ ,\ x \in [Rmin_j, Rmax_j] \tag{4.1}$$

$$pbest_{i,j} = X_{i,j} \tag{4.2}$$

$$V_{i,j} = v\ ,\ v \in [-Vmax_j, Vmax_j] \tag{4.3}$$

Repeat

For all $i \in \{1,2,\ldots,N\}$

$$pbest_i = \begin{cases} pbest_i\ , & f(X_i) > f(pbest_i) \\ X_i\ , & f(X_i) \leq f(pbest_i) \end{cases} \tag{4.4}$$

For all $i \in \{1,2,\ldots,N\}$

$$gbest = \begin{cases} gbest\ , & f(X_i) > f(pbest_{gbest}) \\ i\ , & f(X_i) \leq f(pbest_{gbest}) \end{cases} \tag{4.5}$$

For all $(i,j) \in \{i \times j,\ i \in \{1,2,\ldots,N\},\ j \in \{1,2,\ldots,D\}\}$

$$\begin{aligned} V_{i,j} = \ & w * V_{i,j} + c_1 * rand() * (pbest_{i,j} - X_{i,j}) \\ & + c_2 * rand() * (pbest_{gbest,j} - X_{i,j}) \end{aligned} \tag{4.6}$$

$$V_{i,j} = \begin{cases} V_{i,j}\ , & V_{i,j} \in [-Vmax_j, Vmax_j] \\ -Vmax_j\ , & V_{i,j} < -Vmax_j \\ Vmax_j\ , & V_{i,j} > Vmax_j \end{cases} \tag{4.7}$$

$$X_{i,j} = \begin{cases} X_{i,j} + V_{i,j}\ , & X_{i,j} + V_{i,j} \in [Rmin_j, Rmax_j] \\ Rmin_j\ , & X_{i,j} + V_{i,j} < Rmin_j \\ Rmax_j\ , & X_{i,j} + V_{i,j} > Rmax_j \end{cases} \tag{4.8}$$

Until stop condition is satisfied

Figur 4.6: *SPSO algoritmen med star-topologi* [15]

tillåtna intervallen alternativt avsiktligt fördelas enligt något förutbestämt system.

Efter initieringen går algoritmen in i en iterationsfas tills dess att ett förutbestämt stoppvillkor uppfylls, vanligtvis efter ett specifikt antal iterationer alternativt när ett tillräckligt bra tillstånd har erhållits. Vid varje iteration uppdateras varje partikels *pbest* (4.4) och svärmens *gbest* vid behov (4.5). Därefter beräknas nya hastigheter och positioner för varje partikel tillhörande svärmen (4.6-4.8).

Adaptive PSO

Trots det sociala beteendet hos PSO är det möjligt för partiklar att bli inaktiva och därmed för svärmen att med tiden stagnera. En partikel sägs vara inaktiv när den endast rör sig inom en mycket begränsad del av tillståsdsrummet och inte kan ta sig bort därifrån. En partikel blir inaktiv då $pbest_i$, $pbest_{gbest}$ och X_i alla befinner sig inom ett litet område av tillståndsrummet och när V_i är nära $\vec{0}$. Adaptive PSO (APSO) algoritmen skiljer sig gentemot SPSO i den bemärkelsen att inaktiva partiklar ersätts med nya slumpgenererade partiklar med syftet att undvika stagnering inom svärmen. [15]

I APSO förklaras en partikel inaktiv då skillnaden mellan dess fitnessvärde och svärmens globalt bästa fitnessvärde hittils ligger inom ett litet intervall Δf och då $i \neq gbest$ i ett T_c antal iterationer. Utökningen av SPSO till APSO visas i Figur 4.7 och ersätter ekvationerna 4.6-4.8.

Det som är nytt i APSO gentemot SPSO är Δf, I och T_c. Δf är ett gränsvärde som definierar inom vilket intervall skillnaden mellan en partikels fitnessvärde och fitnessvärdet för *gbest* måste ligga för att partikeln ska uppfylla kravet för inaktivitet. I är en array av heltal med ett element för varje partikel i svärmen. Varje element I_i i I motsvarar en partikel p_i och anger hur många iterationer i följd som partikel p_i uppfyllt kravet för inaktivitet. Alla element i I initieras till 0 vid algoritmens start.

I ekvation 4.9 uppdateras en partikels värde i inaktivitetsräknaren I. I ekvation 4.10 uppdateras en partikel i:s hastighet i en specifik dimension enligt tidigare formel om $I_i \leq T_c$, annars tilldelas partikeln ett helt nytt värde på hastigheten oberoende av tidigare hastighet. På liknande sätt uppdateras en partikel i:s position i ekvation 4.12. Ekvationerna 4.11 och 4.13 syftar till att kontrollera att en partikels hastighet respektive position inte faller utanför de tillåtna gränserna. Vidare sätts *pbest* för en partikel till dess nuvarande tillstånd om partikeln byttes ut av inaktivitetskontrollerna (4.14). Slutligen nollställs inaktivitetsräknaren I_i för partikel i om partikeln ersattes tidigare (4.15).

En liknande utbyggnad av SPSO är det så kallade *no-hope/re-hope* systemet där inak-

$$
\begin{aligned}
&For\ all\ i \in \{1, 2, \ldots, N\} \\
&\quad I_i = \begin{cases} I_i + 1\ , & i \neq gbest \wedge |f(X_{i,j}) - f(pbest_{gbest})| < \Delta f \\ 0\ , & i = gbest \vee |f(X_{i,j}) - f(pbest_{gbest})| \geq \Delta f \end{cases} \qquad (4.9) \\
&\quad For\ all\ j \in \{1, 2, \ldots, D\} \\
&\qquad V_{i,j} = \begin{cases} w * V_{i,j} + c_1 * rand() * (pbest_{i,j} - X_{i,j}) & \\ +c_2 * rand() * (pbest_{gbest,j} - X_{i,j})\ , & I_i \leq T_c \\ v\ ,\ v \in [-Vmax_j, Vmax_j]\ , & I_i > T_c \end{cases} \qquad (4.10) \\
&\qquad V_{i,j} = \begin{cases} V_{i,j}\ , & V_{i,j} \in [-Vmax_j, Vmax_j] \\ -Vmax_j\ , & V_{i,j} < -Vmax_j \\ Vmax_j\ , & V_{i,j} > Vmax_j \end{cases} \qquad (4.11) \\
&\qquad X_{i,j} = \begin{cases} x\ ,\ x \in [Rmin_j, Rmax_j]\ , & I_i > T_c \\ X_{i,j} + V_{i,j}\ , & I_i \leq T_c \end{cases} \qquad (4.12) \\
&\qquad X_{i,j} = \begin{cases} X_{i,j}\ , & X_{i,j} \in [Rmin_j, Rmax_j] \\ Rmin_j\ , & X_{i,j} < Rmin_j \\ Rmax_j\ , & X_{i,j} > Rmax_j \end{cases} \qquad (4.13) \\
&\qquad pbest_{i,j} = \begin{cases} pbest_{i,j}\ , & I_i \leq T_c \\ X_{i,j}\ , & I_i > T_c \end{cases} \qquad (4.14) \\
&\quad I_i = \begin{cases} I_i\ , & I_i \leq T_c \\ 0\ , & I_i > T_c \end{cases} \qquad (4.15)
\end{aligned}
$$

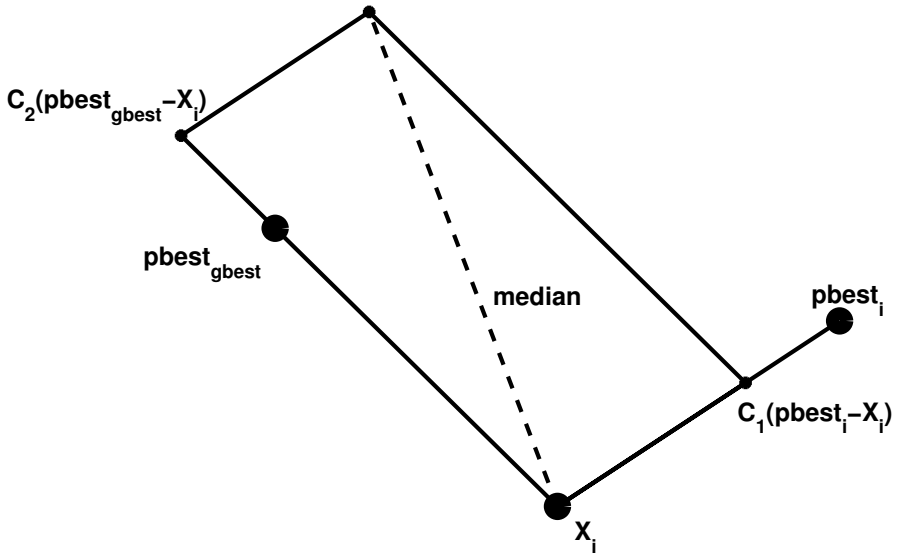
Figur 4.7: *Utökning av SPSO algoritmen till APSO* [15]

tiva partiklar flyttas till ett slumpgenererat tillstånd, men tillåts behålla sitt *pbest* tillstånd [16].

Gaussian PSO

Barry R. Secrest och Gary B. Lamont har utvecklat en variant av PSO kallad Gaussian PSO (GPSO). Genom en grafisk representation av ekvationerna 4.6-4.8, vilken visas i Figur 4.8, anser upphovsmakarna sig funnit två svagheter hos alla tidigare PSO algoritmer. Första problemet är den linjära kurva, medianen, efter vilken en partikel kommer att förflyttas. Upphovsmakarna menar på att sökrummet begränsas för en specifik förflyttning till ett

enda parallellogram konstruerad utifrån punkterna $pbest_{gbest}$, $pbest_i$ samt X_i. Det andra problemet enligt upphovsmakarna uppstår då hastigheten hos en partikel minskar med tiden. Hastighetsminskningen resulterar i att partikelns förflyttning med tiden kommer att sammanfalla med den räta linjen mellan punkterna $pbest_{gbest}$ och $pbest_i$. De här två problemen medför båda att områdena kring $pbest_i$ samt $pbest_{gbest}$ inte genomsöks i samma utsträckning. [3]



Figur 4.8: *Partikelförflyttning hos PSO*

Med GPSO är tanken att områdena kring en partikels bästa tillstånd och det globalt bästa tillståndet ska genomsökas i en större omfattning än vad som görs i andra varianter av PSO. GPSO ämnar eliminera de två nämnda svagheterna hos PSO genom att ersätta ekvationerna 4.6-4.8 i algoritmen för SPSO med ekvationerna i Figur 4.9.

I ekvation 4.16 sparas ett slumpvärde i en temporär variabel, som används i efterföl-

$$
\begin{aligned}
&For\ all\ i \in \{1, 2, \ldots, N\} \\
&\quad temp = rand && (4.16)\\
&\quad dist = \begin{cases} 1\ , & i = gbest \\ \sqrt{\sum_{j=1}^{D}(pbest_{i,j} - pbest_{gbest,j})^2}\ , & i \neq gbest \end{cases} && (4.17)\\
&\quad magn = \begin{cases} GRand((1 - c_2) * dist)\ , & temp > c_1 \\ GRand(c_2 * dist)\ , & temp \leq c_1 \end{cases} && (4.18)\\
&\quad For\ all\ j \in \{1, 2, \ldots, D\} \\
&\qquad V_{i,j} = magn * rand * 2\pi && (4.19)\\
&\qquad X_{i,j} = \begin{cases} pbest_{i,j} + V_{i,j}\ , & temp > c_1 \\ pbest_{gbest,j} + V_{i,j}\ , & temp \leq c_1 \end{cases} && (4.20)\\
&\qquad X_{i,j} = \begin{cases} X_{i,j}\ , & X_{i,j} \in [Rmin_j, Rmax_j] \\ Rmin_j\ , & X_{i,j} < Rmin_j \\ Rmax_j\ , & X_{i,j} > Rmax_j \end{cases} && (4.21)
\end{aligned}
$$

Figur 4.9: *Utökning av SPSO algoritmen till GPSO* [3]

jande beräkningar. Mellanlagringen krävs för att inte nya värden ska slumpas ut vid varje beräkning. Ekvationerna 4.17-4.18 beräknar magnituden hos den nya hastigheten. Ekvation 4.19 beräknar en partikels nya hastighet i dimensionled j vilket sedan används i ekvation 4.20 vid beräkningen av partikelns nya position i dimensionled j. Slutgiltigen kontrolleras en partikels position i dimensionled j så att den befinner sig inom rätt intervall i ekvation 4.21.

Förändringarna i GPSO gentemot PSO resulterar i att områdena kring partikelns bästa och den globalt bästa punkten utforskas i en större omfattning. Barry R. Secrest och Gary B. Lamont har påvisat GPSO:s effektivitet med avseende att erhålla ett så bra värde som möjligt under ett begränsat antal partikelförflyttningar i ett två dimensionellt rum i förhållande till en modifierad version av SPSO över fem ickelinjära begränsade funktioner. [3]

Cooperative PSO

Medan SPSO, APSO samt GPSO arbetar med *en* svärm av partiklar som rör sig i ett rum av D dimensioner, där svärmen har till uppgift att optimera alla D parametrar, arbetar Cooperative PSO (CPSO) med D antal svärmar vilka var och en arbetar i en dimension och endast optimerar en parameter vardera. Optimering inom varje svärm i CPSO kan ske med SPSO, APSO eller någon annan variant. Då varje svärm arbetar i en egen dimension används oftast CPSO på problem i vilka parametrarna är oberoende av varandra. [8]

Ett problem med CPSO är att samtliga parametrar krävs för att beräkna fitnessvärdet, och en lösning på detta problem är att lagra en vektor, $\vec{u}$, av D parametrar vilka håller *gbest* för varje svärm. Kommunikationen mellan svärmarna sköts således via anrop till fitnessfunktionen genom att varje enskild partikel byter ut just sin parameter i $\vec{u}$. [8]

Nedan följer ett matematiskt exempel på detta.

$$\begin{aligned} f(S_{i,p}) &= \begin{cases} \vec{u} = [u_1,\ \ldots,\ u_{i-1},\ u_i,\ u_{i+1},\ \ldots,\ u_n] \\ S_{i,p} = q \end{cases} \\ &= f([u_1,\ \ldots,\ u_{i-1},\ q,\ u_{i+1},\ \ldots,\ u_n]) \end{aligned}$$

där $S_{i,p}$ är partikel p i svärm i vilket motsvarar u_i i vektorn $\vec{u}$.

4.3 Insektsvärmsoptimering

Biologer med flera har länge fascinerats av insekters levnadssätt, deras sociala beteende och förmåga att sätta kolonins välmående i första rum [17]. Till exempel finns det myror vilka skördar löv hundratals meter från stacken varpå löven fraktas hem. Beroende på vilken stig som kräver minst arbetsbörda kommer fler myror att välja just denna stig, vilken efter ett tag mer och mer liknar en mindre motorväg. Vidare finns det myror som bildar kedjor av sina egna kroppar för att möjliggöra för andra arbetsmyror att frakta föda över

mindre klyftor i marken. Flera kedjor kan även länkas ihop för att öka genomströmningen ytterligare. [9]

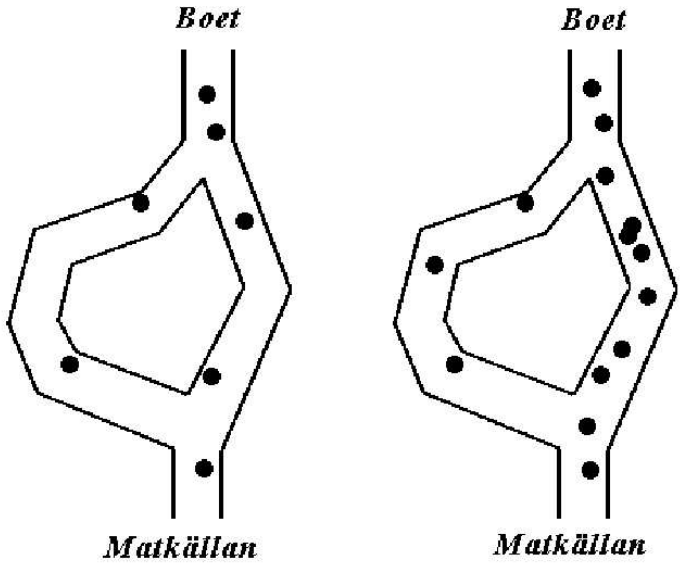
Antalet olika slags organismer på jorden har grovt uppskattats till 10^8, och av dessa är endast 2% sociala insekter [18]. Till de sociala insekterna hör alla myror och termiter samt en del getingar och bin. Av alla arter av sociala insekter är hela 50% myror [8]. Av denna anledning har merparten av forskning inom insektsvärmsoptimering berört myror, och av samma anledning kommer vi i denna sektion koncentrera oss på optimering baserat på myrors beteende.

4.3.1 Kommunikation

Kommunikation mellan insekter, i vårt fall myror, kan ske dels direkt men även indirekt. Direkt kommunikation avser kommunikation myror emellan, exempelvis visuell kontakt, utbyte av kroppsvätskor och lukt. Med indirekt kommunikation avses en modifikation av omgivningen, miljön, vilket i sin tur påverkar andra myror.

En form av indirekt kommunikation som sker hos myror är deras spårläggande och spårföljande beteende vid letandet efter föda. Myror lägger ut spår av en kemisk substans, feromon, när de rör sig mellan boet och en matkälla. Andra myror vilka letar föda följer sedan gärna detta spår av feromoner. Om det finns flera spår väljer en myra med större sannolikhet ett spår med högre koncentration av feromon. [9]

I Figur 4.10 visas feromonets påverkan på myrornas vägval. I bilden till vänster har några myror nyligen funnit en ny matkälla, och två olika vägar som leder dit. Till en början kommer ungefär lika många myror välja varje väg, men då myror vilka väljer den kortare vägen återvänder till boet snabbare kommer koncentrationen av feromoner öka snabbare på den kortare vägen [8]. Som en följd av att feromonkoncentration ökar snabbare på den kortare vägen kommer fler och fler myror välja den kortare vägen, vilket visas i bilden till höger i Figur 4.10. Det ska även påpekas att enstaka myror kommer att fortsätta välja den långa vägen, då vägvalet hela tiden är en fråga om sannolikheter. Att enstaka myror



Figur 4.10: *Feromonföljande myror*

kommer att fortsätta att välja den långa vägen är en av anledningarna till att myror är så bra på att anpassa sig till förändringar i miljön - om den kortare vägen görs otillgänglig kommer myrorna snabbt anpassa sig till den nya omgivningen genom att använda den längre vägen [9].

En kolonis decentraliserade egenskaper karakteriseras av

- Avsaknad av centraliserad styrning
- Kommunikation och koordinering bland kolonins individer sker med hjälp av en lokal miljöpåverkan
- Positiv feedback ger upphov till ökad sannolikhet att återupprepa samma beteende

Vid implementation av en myrkoloni är det just kolonins decentraliserade egenskaper det strävas efter att åstadkomma med hjälp av olika typer av matematiska modeller. [8]

4.3.2 Feromon i kontinuerliga rum

Optimeringsalgoritmer baserade på myrors beteende skapades ursprungligen som ett fler-agent system för lösning av kombinatoriska optimeringsproblem, och har i flertalet experiment visat sig vara effektivt vid lösning av till exempel TSP[3] [17][9]. På senare år har det även gjorts flertalet försök att applicera myrbaserade algoritmer på kontinuerliga funktioner [19][20][21]. Ett antal komplikationer uppstår vid försök att applicera myrbaserade algoritmer på kontinuerliga funktioner, där den huvudsakliga är hanteringen av feromoner. Det är inte helt självklart hur feromonet ska hanteras när varje parameter till ett problem kan anta oändligt många olika värden, eller hur myrorna ska röra sig i tillståndsrummet. En metod för att lösa hanteringen av feromon är att dela upp tillståndsrummet i ett antal delrum vilket motsvarar antalet myror och därefter låta myrorna röra sig med en förutbestämd steglängd bland delrummen, se Figur 4.11.

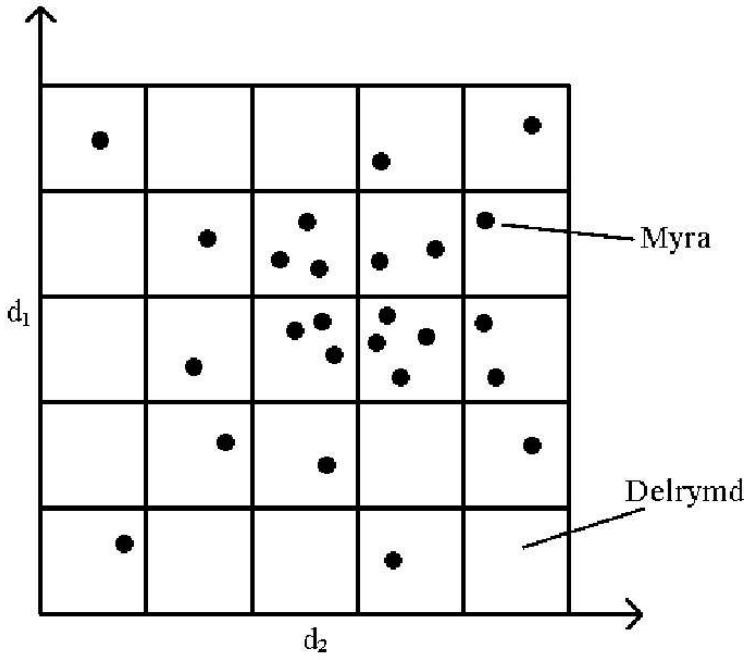
Delrummen används för att bestämma hur varje myra ska röra sig i varje enskilt dimensionsled. Rörelseriktningen för en myra p i dimension d_i bestäms av förhållandet mellan den totala feromonnivån i alla delrum till vänster om p i dimensionsledet för d_i och den totala feromonnivån i alla delrum till höger om p i dimensionsledet för d_i. Delrummen är av samma storlek och fördelas jämnt över tillståndsrummet [20][21]. Totala antalet delrum, M, i tillståndsrummet kan skrivas som

$$M = m^D$$

där m är antalet delrum längs varje dimensionsled och D är antalet dimensioner i problemet. Problemet med denna metod är således att när antalet dimensioner i problemet ökar linjärt, så ökar antalet delrum exponentiellt. I praktiken fungerar alltså denna metod endast när antalet dimensioner i problemet är relativt litet.

Då tidigare forskning inom insektsvärmsoptimering för kontinuerliga rum endast be-

[3] Travelling Salesman Problem



Figur 4.11: *Myror och delrum i ett två-dimensionellt tillståndsrum*

handlar problem av ett litet antal dimensioner [20][21] i kombination med att uppsatsens experimentdel ämnar behandla problem av ett större antal dimensioner, har experimentet begränsats till att inte inkludera insektsvärmsoptimering. Av samma anledning behandlar inte uppsatsen de olika algoritmerna för insektsvärmsoptimering.

Kapitel 5

Experiment

För att få svar på vår frågeställning, hur GPSO står sig i jämförelse med ett antal vanliga varianter av PSO vid lösning av ickelinjära begränsade flerdimensionella problem, utförs här ett experiment. Experimentet, vilket ämnar undersöka frågeställningen, är uppdelat i två delar: en återupprepningsdel, del I, och en påbyggnad, del II.

Del I ämnar styrka tidigare experiment på GPSO utförda av Barry R. Secrest och Gary B. Lamont [3] samt verifiera vår egen implementation av SPSO och GPSO.

I del II genomförs nya, icke tidigare utförda, tester för att undersöka GPSOs effektivitet vid lösning av ickelinjära begränsade flerdimensionella problem i jämförelse med ett antal utvalda PSO-algoritmer. Då GPSO och de övriga PSO-algoritmerna tillhör samma familj av metoder, jämförs de även med en referensmetod för lösning av ickelinjära begränsade problem. Utan en referensmetod att jämföra med är det omöjligt att bedöma en metods effektivitet.

I sektion 5.1 beskrivs experimentets testmiljö och den referensmetod som valts. I sektion 5.2 och sektion 5.3 beskrivs del I respektive del II av experimentet. Testfunktionerna som används i experimentet beskrivs i sektion 5.4 och slutligen återfinns resultatet från experimentet med tillhörande diskussion i sektion 5.5.

5.1 Experimentmiljö

Experimentet utfördes på en PC med en Intel Pentium 4 processor på 2.4GHz och 512Mb RAM-minne. Operativsystemet var Microsoft Windows XP Professional version 5.1.2600 och MATLAB version 6.5.1.199709 med service pack 1 användes. Vidare var MATLAB utrustad med Optimization Toolbox version 2.3 vilket enbart utnyttjades för att få tillgång till en beprövad metod, vilken används som referensmetod, för lösning av ickelinjära begränsade problem. Mer om MATLAB och den valda referensmetoden, FMINCON, återfinns under sektion 5.1.1.

5.1.1 MATLAB och FMINCON

MATLAB är en känd programvara från MathWorks Inc. som förutom symbolisk manipulering också erbjuder möjligheten att lösa matematiska problem numeriskt, med matriser som kärna vid beräkningar. MATLAB erbjuder även kraftfulla metoder för att enkelt generera grafiska illustrationer av bland annat funktioner. MATLAB kan således användas inom ett stort antal branscher däribland matematik, fysik, kemi och läkemedel. MATLABs breda användningsområde tillsammans med dess tillgänglighet på en rad olika plattformar - däribland PC, Macintosh och UNIX - har gjort MATLAB populärt såväl inom industrin som vid universitet världen över. MATLAB erbjuder vidare programmeringsmöjligheter liknande vanliga programspråk, med en C-liknande syntax, samt verktyg för skapandet av grafiska användargränssnitt vilket leder till att MATLAB kan användas som ett utvecklingsverktyg för applikationer. [22]

Det finns ett antal verktygslådor (toolboxes) tillgängliga som utökar MATLABs funktionalitet och användningsområden. En av dessa verktygslådor är Optimization Toolbox, vilket som namnet antyder erbjuder en rad funktioner som kan användas vid numerisk optimering. En av de funktioner som ingår i Optimization Toolbox är FMINCON. Namnet FMINCON kommer från att FMINCON används för att minimera (MINimize) en begrän-

sad (CONstrained) funktion f (F). FMINCON är således en funktion som används för att försöka hitta det globala minimumet till en begränsad ickelinjär funktion av flera variabler, eller dimensioner. FMINCON använder en rad olika algoritmer vid sökandet efter det globala minimumet, och vilken som används i ett specifikt fall beror på funktionen som ska optimeras samt FMINCONs inställningar. De funktioner som berörs i delexperiment II faller under den kategori av optimering som FMINCON kallar Medium-Scale, på grund av att ingen gradient[1] tillhandahålls automatiskt av funktionerna. För Medium-Scale optimering använder FMINCON en optimeringsmetod som kallas Sequential Quadratic Programming (SQP). SQP i sin tur löser kvadratiska subproblem och använder två metoder vid namn Line-search och quasi-Newton vilka dock inte är specifika för SQP. [23]

Eftersom FMINCON endast används för att ge läsaren en referenspunkt för experimentets resultat kommer inte den SQP-algoritm som används beskrivas alls. Intresserade läsare hänvisas däremot till böckerna "Linjär och ickelinjär optimering" av Lundgren, Rönnqvist och Värbrand [5] samt "Numerical Optimization" av Nocedal och Wright [6].

5.2 Del I - återupprepning

I del I av experimentet upprepas Secrests och Lamonts experiment där GPSO jämfördes med SPSO över ett antal testproblem, och de båda algoritmerna körs under ett fixt antal iterationer på varje testproblem. Varje algoritm körs tio gånger på varje testproblem enligt det ursprungliga experimentet. Resultatet presenteras i form av en tabell med statistik över algoritmernas framgångar tillsammans med en tabell över resultaten från det ursprungliga experimentet, se sektion 5.5.1.

[1] Gradienten ∇f för en funktion $f(\vec{x})$ är en kolonnvektor innehållande alla partiella förstaderivator till funktionen.

5.2.1 Parameterinställningar

Till de båda algoritmer som ingår i delexperimentet finns en rad olika parametrar. De olika parametervärdena är tagna direkt från artikeln "Visualizing Particle Swarm Optimization - Gaussian Particle Swarm Optimization" av Secrest och Lamont [3] då delexperiment I är en ren återupprepning. SPSO algoritmen som användes för delexperiment I skiljer sig gentemot algoritmen som beskrivs i Figur 4.6 på följande tre punkter

- Hastigheten hos samtliga partiklar initieras till noll.
- Avsaknad av hastighetsbegränsning.
- Avsaknad av intervallbegränsning för X_i.

Utöver ovannämna skillnader användes parameterinställningarna som visas i Tabell 5.1.

Parameter	*Värde*
Svärmstorlek, N	100
Inertia weight, w	0.8
Accelerationskonstant, c_1	2.8 för SPSO, 0.4 för GPSO
Accelerationskonstant, c_2	1.3 för SPSO, 0.6 för GPSO

Tabell 5.1: *Parameterinställningar vid utförandet av delexperiment I.*

Anledningen till att parametrarna c_1 och c_2 hos GPSO inte är enhetliga med SPSO bygger på deras något annorlunda uppgift där värden större än ett helt enkelt inte är tillåtna.

5.2.2 Testproblem

De testfunktioner vilka användes av Secrest och Lamont [3] i deras studie var F_1, F_2, F_6, F_7 och F_8. Samtliga testfunktioner var av två dimensioner där $x_1, x_2 \in [-5000, 5000]^2$ och finns beskrivna i sektion 5.4.

[2]Notera att detta inte är de ursprungliga intervallen för funktionerna vilka anges under respektive funktionsbeskrivning.

5.3 Del II - påbyggnad

I del II av experimentet undersöks förutom GPSO och SPSO även APSO, PSO med ring-topologi samt PSO med wheel-topologi. Algoritmerna körs här till skillnad från i del I under en fix cpu-tid vilken är tillräckligt hög för att tillåta alla algoritmer att konvergera. Varje algoritm körs 25 gånger på varje testproblem för att erhålla tillförlitliga resultat. Resultatet presenteras i form av en tabell med statistik över algoritmernas framgångar samt konvergeringskurvor för varje testfunktion. En känd SQP- implementation från MathWorks, FMINCON, har även körts på alla testproblem för att ge något konkret att jämföra PSO-algoritmernas resultat med. SQP har valts för jämförelsen då det är en välkänd och välanvänd algoritm för lösandet av ickelinjära begränsade problem.

5.3.1 Parameterinställningar

Till samtliga algoritmer som ingår i delexperimentet finns en rad olika parametrar. Att för varje algoritm, och för den delen också problem, finna den optimala kombinationen av parametervärden är i sig en uppsats. Vidare skulle ett sökande efter de optimala parametervärden bidra till att experimentets resultat överlag blir mer problemspecifikt och därmed också medföra ett mindre generellt resultat. Av ovanstående anledning bestäms värdet hos parametrarna i experimentet utifrån vad som i tidigare experiment visats ge goda resultat. Tabell 5.2 visar grundinställningarna för parametrarna vid experimentet med undantag från att GPSO använder sig utav $c_1 = 0.4$ och $c_2 = 0.6$ [3]. Anledningen till att parametrarna c_1 och c_2 hos GPSO inte är enhetliga med övriga algoritmer är detsamma som tidigare nämnts i sektion 5.2.1.

Eftersom inaktivitetsparametern Δf är problemspecifik har vi valt att ge Δf en generell definition i vårt experiment. Vidare skulle ett fixt Δf omöjliggöra fortsatt konvergens nära noll, något som vår definition möjliggör. Till skillnad från delexperiment I använder delexperiment II de i uppsatsen tidigare presenterade algoritmer, se sektion 4.2.2.

Parameter	*Värde*
Svärmstorlek, N	$(10+50)/2 = 30$ [2]
Omgivningsstorlek, n	$(10\% + 20\%)/2 = 15\% \Longrightarrow 0.15N$ [2]
Inertia weight, w	$w(t) = 0.9 - 0.5t,\ t : 0 \rightarrow 1$ [2][12][13]
Hastighetstak, $Vmax$	$\forall j \in \{1, 2, \ldots, D\}, Vmax_j = 0.15(Rmax_j - Rmin_j)$ [12][13]
Accelerationskonstanter, c_1 och c_2	2 [12][13][15][14]
Inaktivitetsparameter, T_c	3 [15]
Inaktivitetsparameter, Δf	$\frac{f(pbest_{gbest})}{100}$

Tabell 5.2: *Parameterinställningar vid utförandet av delexperiment II.*

5.3.2 Testproblem

I påbyggnadsdelen har antalet funktioner utökats gentemot del I till att inkludera hela De Jongs Test Suite, det vill säga $F_1 - F_5$. Varje funktion förutom F_8 undersöktes för ett större antal dimensioner gentemot del I, vilket visas i Tabell 5.3.

Funktion	*Dimensioner*
F_1	$2, 5, 10, 20, 30$
F_2	$2, 5, 10, 20, 30$
F_3	$2, 5, 10, 20, 30$
F_4	$2, 5, 10, 20, 30$
F_5	$2, 3, 4, 5$
F_6	$2, 5, 10, 20, 30$
F_7	$2, 5, 10, 20, 30$
F_8	2

Tabell 5.3: *Undersökta problem i delexperiment II.*

5.4 Testfunktioner

I sektion 5.4.1-5.4.4 ges kortfattade beskrivningar över de olika testfunktionerna, F_1 - F_8, och deras matematiska definitioner samt grafiska illustrationer. Generellt gäller att det globala minimet är 0 för samtliga funktioner förutom för Foxholes, F_5, där det globala minimet är 0.998.

5.4.1 De Jongs Test Suite, $F_1 - F_5$

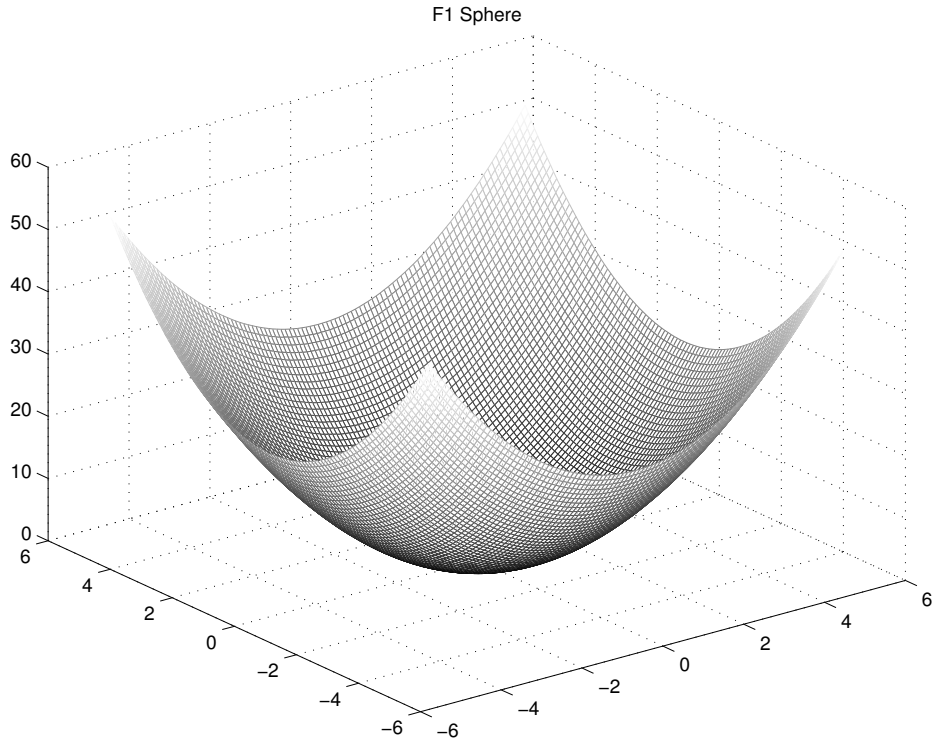
När De Jong skrev sin avhandling om en klass av genetiska adaptiva system [24] använde De Jong sig vid sina undersökningar av en test suite med fem olika matematiska funktioner. De fem funktionerna valdes för att spegla ett antal svårigheter av olika karaktär som kan uppstå vid optimering. Test suiten har senare uppkallats efter De Jong, och delar av eller hela test suiten har använts inom optimeringsområdet genom åren. De fem funktionerna vilka ingår i De Jongs test suite är Sphere (F_1), Rosenbrock (F_2), Step (F_3), Quartic (F_4) samt Foxholes (F_5).

F_1 Sphere

Sphere är en enkel ickelinjär, konvex (skålformad) funktion för vilken det är relativt enkelt att hitta det globala minimumet. På grund av att det är så pass enkelt att hitta det globala minimumet används Sphere ofta som ett mått på en algoritms effektivitet överlag. Sphere illustreras för två dimensioner i Figur 5.1 och är definierat som följer

$$F_1(\vec{x}) = \sum_{j=1}^{D} x_j^2 \ , \ -5.12 \leq x_j \leq 5.12$$

De Jong använde ursprungligen Sphere med tre dimensioner. [24]



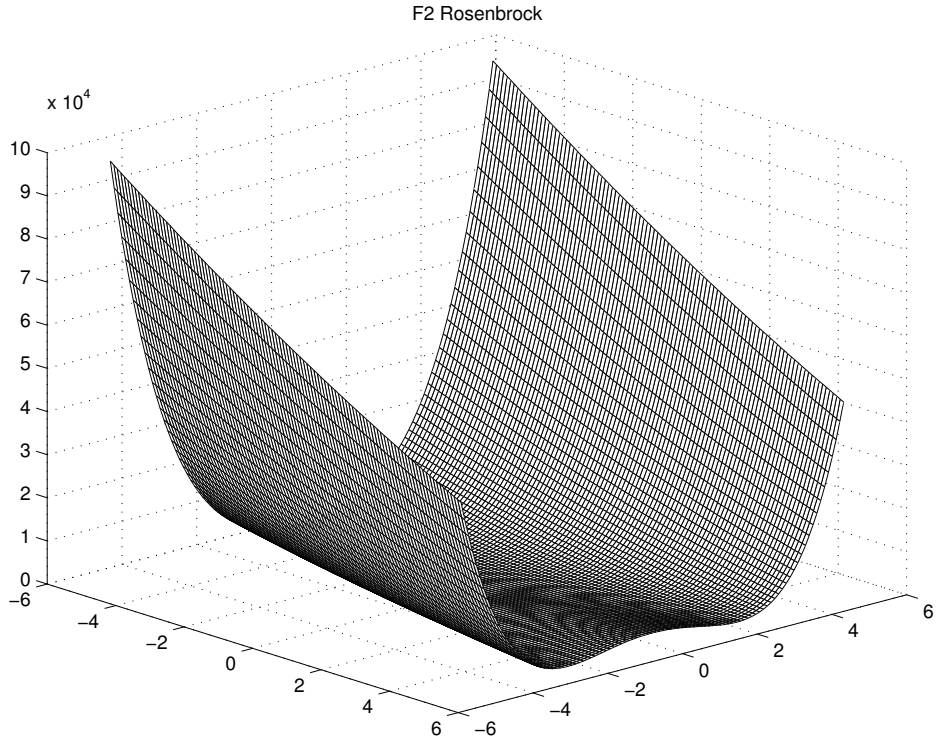
Figur 5.1: F_1 *Sphere av två dimensioner*

F_2 Rosenbrock

Rosenbrock är en funktion för vilken det är mycket svårt och därmed tidskrävande att hitta det globala minimumet, till skillnad från Sphere. Besvärligheten ligger i det globala minimumet återfinns i en både smal och djup "ravin" vilken löper längs en parabol. Rosenbrocks funktion illustreras i Figur 5.2 samt Figur 5.3 och är definierad enligt följande

$$F_2(\vec{x}) = \sum_{j=1}^{D-1} (100(x_{j+1} - x_j^2)^2 + (x_j - 1)^2) \ , \ -5.12 \leq x_j \leq 5.12$$

De Jong använde sig ursprungligen av Rosenbrocks funktion av två dimensioner. [24]



Figur 5.2: F_2 *Rosenbrock av två dimensioner*

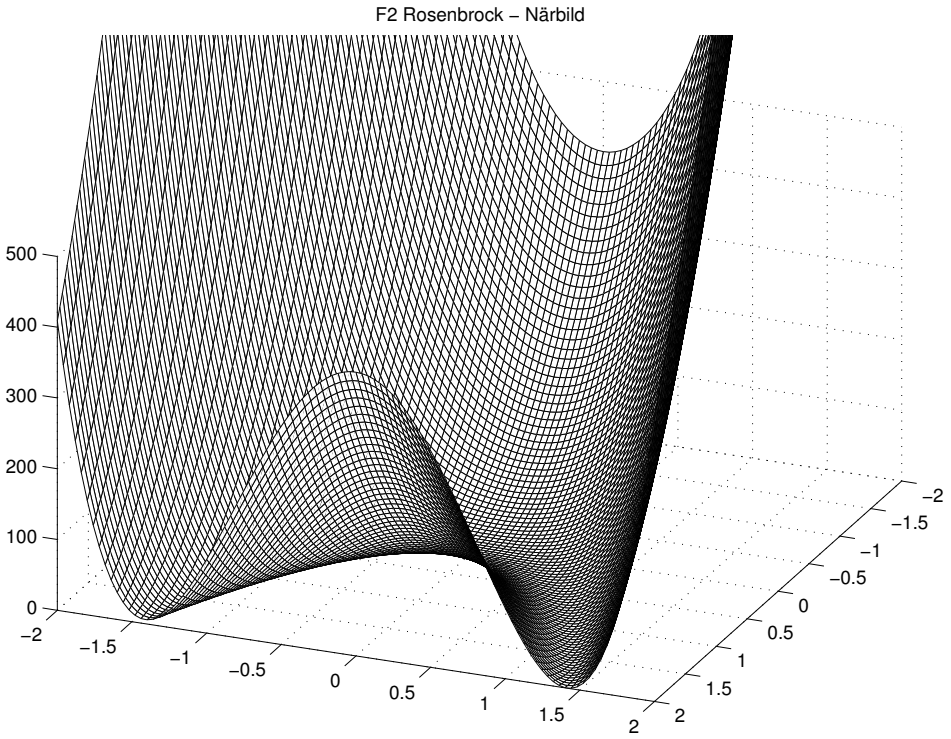
F_3 Step

Step är en funktion vald för att spegla algoritmernas effektivitet vid icke kontinuerliga och icke konvexa problem. I Figur 5.4 illustreras en step funktion av två dimensioner och ges av den matematiska definitionen

$$F_3(\vec{x}) = 5 * D + \sum_{j=1}^{D} int(x_j) \ , \ -5.12 \leq x_j \leq 5.12$$

där $int(x_j)$ ger heltalsdelen av x_j.

De Jongs ursprungliga version av F_3 baserades på fem dimensioner. [24]



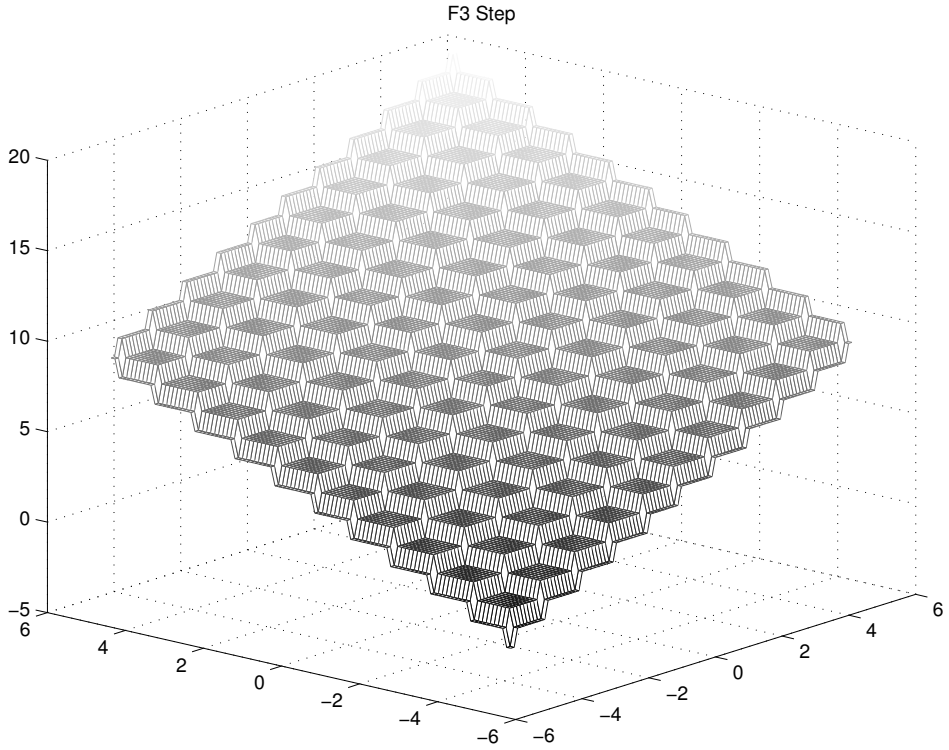
Figur 5.3: F_2 *Rosenbrock av två dimensioner, närbild*

F_4 Quartic

Quartic är en kontinuerlig och konvex funktion med brus, *gaussian noise*. Med brus menas att funktionsvärdet för samma $\vec{x}$ med väldigt stor sannolikhet medför två skilda resultat vid olika tidpunkter. Brus är vanligt bland problem hämtade från verkligheten, till exempel är det inte säkert att två mätningar alltid ger exakt samma svar utan att de kan variera något. F_4 illustreras i Figur 5.5 och ges av

$$F_4(\vec{x}) = \sum_{j=1}^{D} (jx_j^4 + gauss(0,1)) \ , \ -1.28 \leq x_j \leq 1.28$$

där $gauss(0,1)$ resulterar i ett slumgenererat tal mellan noll och ett. De Jong använde i sin forskning en quartic funktion av 30 dimensioner. [24]



Figur 5.4: *F_3 Step av två dimensioner*

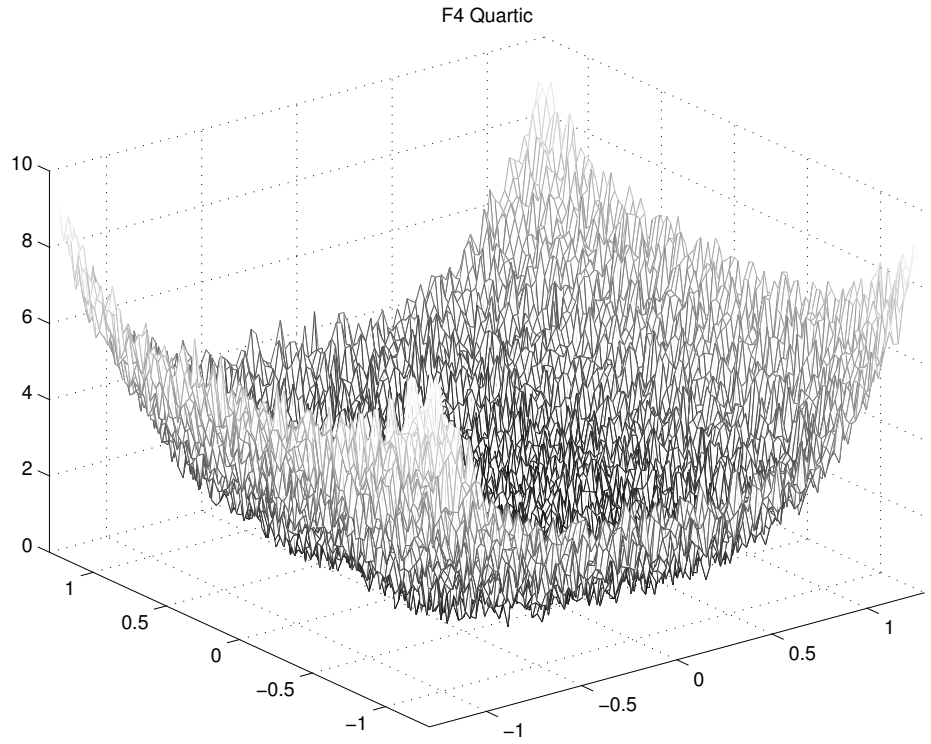
F_5 Foxholes

Foxholes är en funktion med många lokala optima, så kallade foxholes, vilka är utplacerade med explicit angivna koordinater och omgivna av en platå med derivatan noll. Funktionen är definierad som följer

$$F_5(\vec{x}) = \frac{1}{\frac{1}{5^D+1} + \sum_{j=1}^{5^D}(\frac{1}{j+\sum_{i=1}^{D}(x_i-B_{ij})^6})}\ ,\ -65.536 \leq x_i \leq 65.536$$

där D är antalet dimensioner i problemet och B är en matris av storleken $D \times 5^D$ där varje kolonn motsvarar ett specifikt foxholes koordinater, vilket implicerar att antalet foxholes är lika med 5^D. Foxholes illustreras i Figur 5.6.

Eftersom varje foxholes är så pass djupt och därmed har såpass mycket lägre funk-



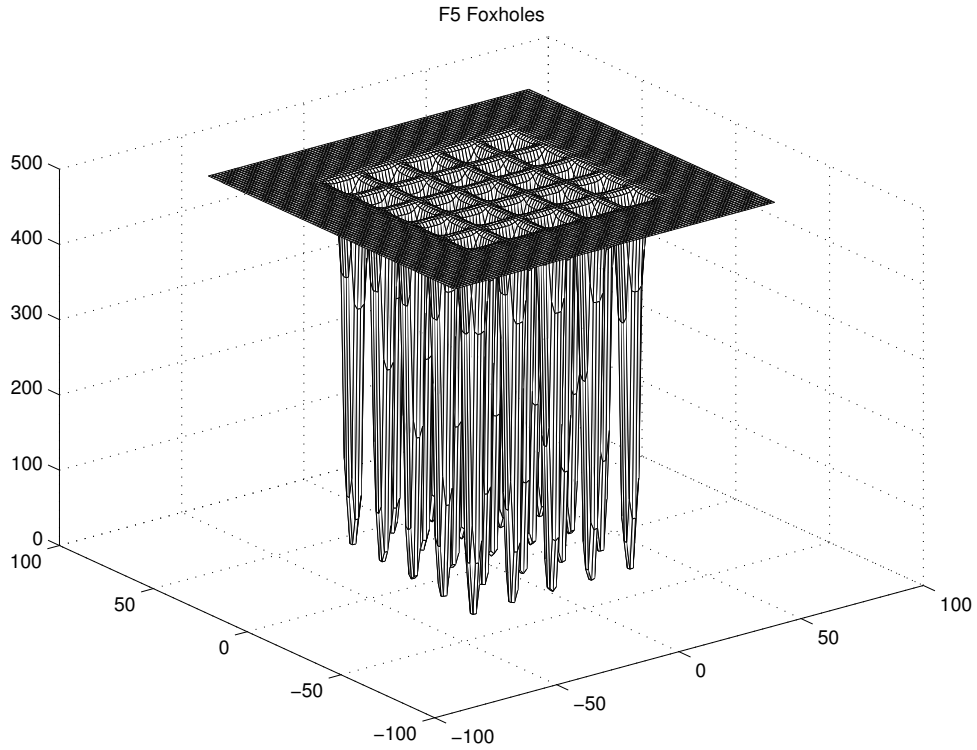
Figur 5.5: F_4 *Quartic av två dimensioner*

tionsvärde än den omgivande platån, är det lätt för algoritmer att gå ner i det första foxholet de hittar och sedan fastna där. En algoritm som inte presterar bra på foxholes är inte bra på att undfly lokala optima. Om algoritmen istället fastnar på platån utan att hitta något foxhole så implicerar det att algoritmen kommer att prestera dåligt även på F_3 Step. De Jong använde sig utav F_5 av två dimensioner och därmed 25 foxholes. [24]

5.4.2 F_6 Griewangk

Griewangks funktion är en flerdimensionell icke konvex funktion med många lokala optima, både maxima och minima. Griewangks funktion illustreras i Figur 5.7 och ges av

$$F_6(\vec{x}) = \frac{1}{4000}\sum_{j=1}^{D} x_j^2 - \prod_{j=1}^{D} \cos\frac{x_j}{\sqrt{j}} + 1 \ , \ -600 \leq x_j \leq 600$$



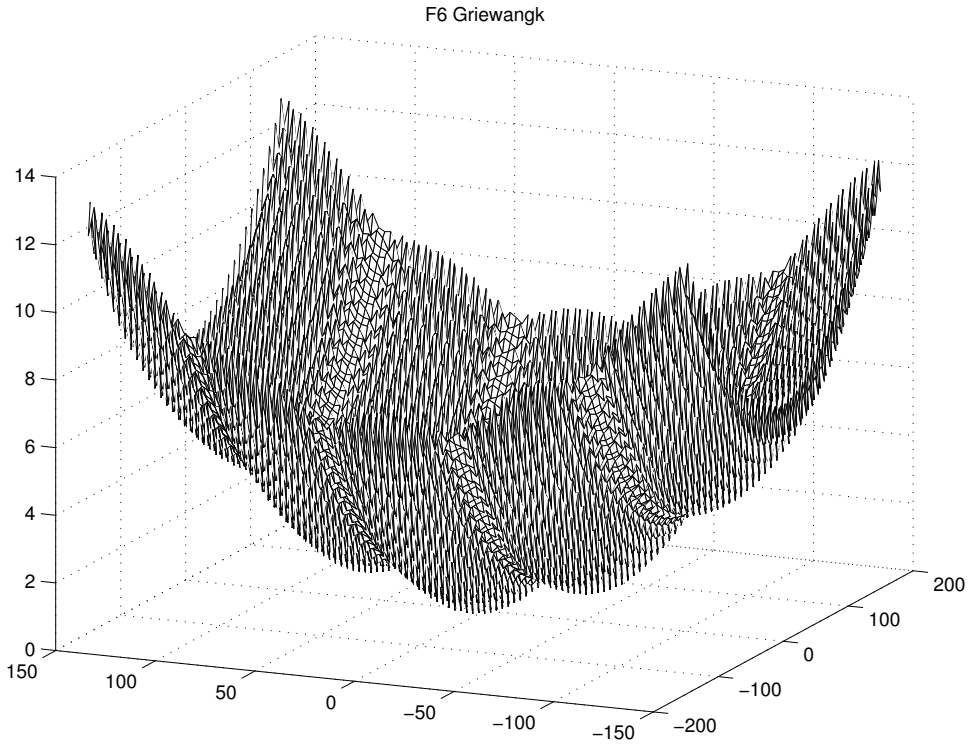
Figur 5.6: F_5 *Foxholes av två dimensioner*

där D är antalet dimensioner. Antalet lokala optima ökar med antalet dimensioner och redan då $D = 10$ har fler än 1000 lokala optimum rapporterats. [25]

Lösningsmetoder vilka arbetar med en variabel i taget har vanligtvis problem med Griewangk's funktion då dess dimensioner är direkt beroende av varandra genom produkttermen.

5.4.3 F_7 Rastrigin

Ursprungligen konstruerades Rastrigins funktion för två dimensioner men generaliserades senare av Mühlenbein för ett godtyckligt antal dimensioner. Det är den generaliserade versionen av Rastrigins funktion vi presenterar i denna sektion. Svårigheten hos Rastrigins funktion ligger i det stora antalet lokala minima. Rastrigins funktion är icke-linjär och

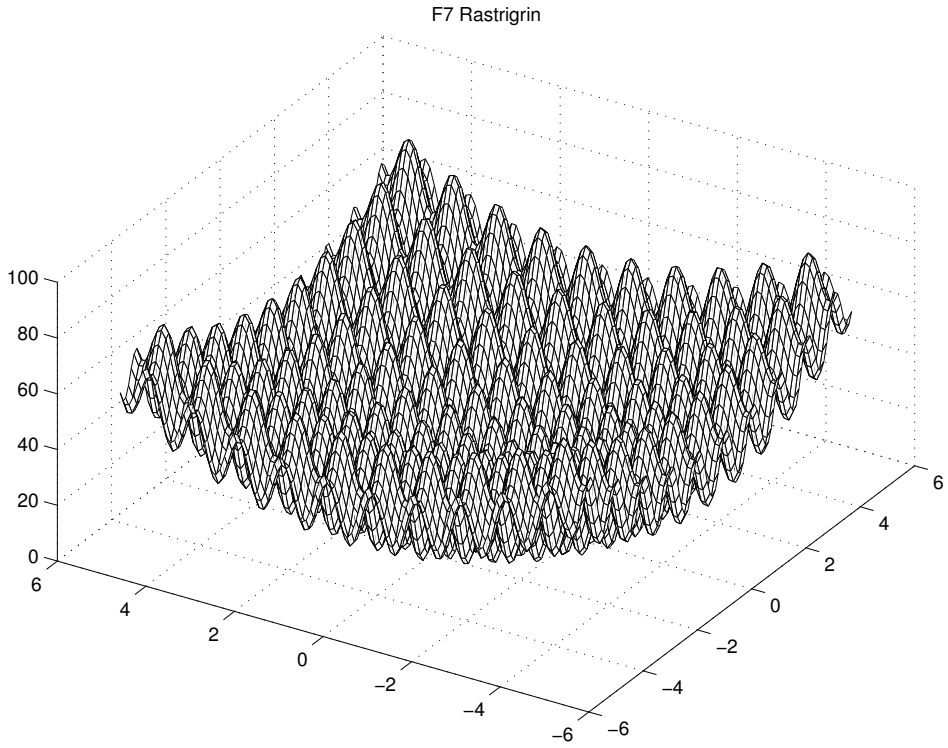


Figur 5.7: F_6 *Griewangk av två dimensioner*

funktionsvärdet för de lokala optimumen ökar med avståndet från det globala minimet i punkten $\vec{x} = \vec{0}$. Rastrigins funktion illustreras i Figur 5.8 och ges av

$$F_7(\vec{x}) = \sum_{j=1}^{D} (x_j^2 - A\cos(\omega x_j) + A) \ , \ -5.12 \leq x_j \leq 5.12$$

där D är antalet dimensioner, A är amplituden och ω är frekvensen. Större värde på ω resulterar i ett ökat antal lokala optima inom samma intervall för x_i. Vanliga värden på konstanterna är $A = 10$ och $\omega = 2\pi$. [26]



Figur 5.8: F_7 *Rastrigin av två dimensioner*

5.4.4 F_8 Schaffer

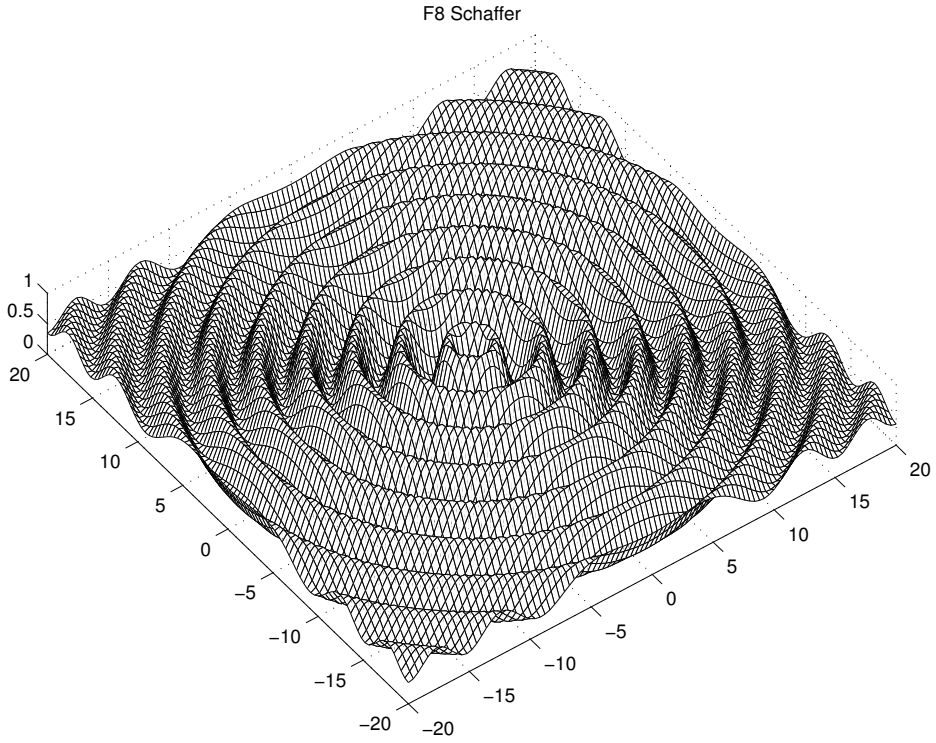
Schaffers funktion är en icke-linjär tvådimensionell funktion med oändligt många lokala optima och ges av

$$F_8(\vec{x}) = 0.5 + \frac{\sin^2 \sqrt{x_1^2 + x_2^2} - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2}$$

där $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

5.5 Resultat och diskussion

Resultaten från delexperiment I och delexperiment II presenteras i sektion 5.5.1 respektive sektion 5.5.2 med tillhörande diskussion.



Figur 5.9: F_8 *Schaffer av två dimensioner*

5.5.1 Del I - återupprepning

Mätvärdena erhållna från återupprepningsförsöket presenteras i Tabell 5.6 och överstämmer väl med tidigare mätningar utförda av Secrest och Lamont vilka presenteras i Tabell 5.5. När vi i fortsättningen nämner tidigare mätningar är det med hänvisning till Secrests och Lamonts resultat presenterade i artikeln "Visualizing Particle Swarm Optimization - Gaussian Particle Swarm Optimization" [3].

Secrest och Lamont nämner inte i sin artikel [3] hur många iterationer som GPSO och SPSO körts för att uppnå det resultat de presenterar, utan nämner enbart att både GPSO och SPSO "tillåtits att konvergera". På grund av utelämnandet av antalet iterationer i Secrests och Lamonts artikel [3] var vi tvungna att testa oss fram för att erhålla lämpliga antal iterationer. Ett lämpligt antal iterationer betyder att $\forall m_u,\ m_u \in [m_t - \sigma_t,\ m_t + \sigma_t]$

där m_u är erhållet medelvärde av en viss algoritm för en viss funktion (se Tabell 5.6), m_t är motsvarande medelvärde vid tidigare mätning och σ_t är motsvarande standardavvikelse (se Tabell 5.5). Båda algoritmerna måste dessutom köra samma antal iterationer för samma funktion. Antalet nödvändiga iterationer för att erhålla önskat resultat utföll enligt Tabell 5.4.

Funktion	*Antal iterationer*
F_1	62
F_2	50
F_6	33
F_7	40
F_8	40

Tabell 5.4: *Nödvändigt antal iterationer.*

Samtliga erhållna medelvärden efter antalet iterationer presenterade i Tabell 5.4, förutom medelvärden för F_8, faller inom intervallet $[m_t - \sigma_t,\ m_t + \sigma_t]$.

Om Secrest och Lamont verkligen givit algoritmerna tillräckligt med iterationer så att de ”tillåtits att konvergera” kan diskuteras. Anledningen till vårt tvivel bygger främst på resultatet för F_1, som har ett globalt minimum på 0 och saknar lokala minima förutom det globala, men gäller även andra funktioner som F_2 och F_7 vars resultat är långt från bra. Huruvida algoritmerna givits tid för att konvergera låter vi dock vara osagt då uppsatsens syfte inte direkt berörs av frågan. Från vår synvinkel nöjer vi oss med att konstatera att våra mätningar, i det stora hela, överensstämmer med de tidigare utförda mätningarna vilket därmed bekräftar vår implementations korrekthet.

Förutom erhållet resultat noterades även att SPSO i genomsnitt var 79% snabbare än GPSO medan de tidigare mätningarna visade på 10%.

Mätvärden

Funktion	*GPSO*			*SPSO*		
	Bäst	*Medel*	σ	*Bäst*	*Medel*	σ
F_1	3.30e-12	1.57e-9	2.45e-9	5.1	1020	1725
F_2	1.49e-1	4156	4143	67.1	1.10e+7	2.20e+7
F_6	3.10e-5	1.99e-2	1.75e-2	4.39e-1	1.04	4.36e-1
F_7	5.51e-8	6.26e-1	4.85e-1	89.8	1143	1325
F_8	3.85e-3	1.19e-2	9.09e-3	1.90e-1	4.41e-1	9.34e-2

Tabell 5.5: *Tidigare resultat av Secrest och Lamont.* [3]

Funktion	*GPSO*			*SPSO*		
	Bäst	*Medel*	σ	*Bäst*	*Medel*	σ
F_1	8.32e-11	3.25e-9	6.03e-9	1.88	38.5	49.4
F_2	2.00e-1	1324	1712	197.9	168721	418807
F_6	3.30e-3	3.21e-2	2.51e-2	1.54e-1	6.33e-1	3.06e-1
F_7	2.79e-2	9.66e-1	7.72e-1	28.9	139.8	162.8
F_8	9.72e-3	1.87e-2	1.30e-2	1.47e-2	2.47e-1	1.29e-1

Tabell 5.6: *Resultat av upprepning.*

5.5.2 Del II - påbyggnad

Mätvärdena erhållna från påbyggnadsförsöket presenteras i Tabell 5.7-5.14 och i Figur 5.11-5.18 visas konvergeringskurvor för de olika testproblemen. Konvergeringskurvorna visar hur algoritmerna konvergerar som funktioner av tiden, och syftar till att visa på skillnader i konvergeringshastighet mellan olika algoritmer. Konvergeringskurvorna syftar *inte* till att visa algoritmernas slutresultat, utan det är tabellernas uppgift.

Följande slutsatser kan dras utifrån erhållna mätvärden

- SPSO erhåller i alla fall förutom vid Rosenbrock, av två dimensioner, och Step, av samtliga dimensionsantal, bättre värden än GPSO. Genom våra mätresultat har vi således visat att GPSO inte medför någon förbättring gentemot SPSO vid ickelinjära begränsade flerdimensionella problem förutom i enstaka fall. Mätresultaten visar även att GPSO inte medför någon förbättring gentemot resterande PSO-algoritmer i de flesta fallen.

- SPSO erhåller i alla fall förutom vid Rosenbrock, av samtliga dimensionsantal, och Griewangk, av 20 och 30 dimensioner, ett bättre medelvärde än FMINCON. I flertalet problem - F_3, F_4, F_5, F_7 och F_8 - visar samtliga PSO-algoritmerna ett bättre medelvärde än FMINCON. PSO-algoritmernas goda resultat gentemot en så välkänd och välanvänd algoritm för lösandet av ickelinjära begränsade problem visar att PSO-algoritmerna inte är att förkasta och att de i många fall förefaller kunna prestera bättre resultat än SQP-metoder.

- GPSO gör överlag ett färre antal funktionsevalueringar gentemot andra PSO-algoritmer. Anledningen bakom det färre antalet funktionsevalueringar är att algoritmen för GPSO är mer beräkningskrävande vilket också noterades i delexperiment I. Att GPSO är mer beräkningskrävande gör sig mest tydligt i de fall där problemfunktionen är enkel att beräkna, jämför f-eval kolumnen i exempelvis Tabell 5.11 och Tabell 5.7. Ett färre antal funktionsevalueringar gentemot ett högre antal för att erhålla ett

specifikt resultat är att föredra och blir viktigare ju mer beräkningskrävande varje funktionsevaluering är.

- GPSO presterar enastående för Step, F_3, där optimal lösning erhölls vid samtliga körningar oavsett antalet dimensioner, se Tabell 5.9. Konvergeringskurvorna i Figur 5.13 styrker GPSOs effektivitet där vi även kan notera att optimal lösning erhålls vid ett tidigt skede, långt innan exekveringen upphör. Anledningen till varför GSPO presterar så bra för Step är att GPSO sprider partiklarna i svärmen runt *gbest* och varje partikels *pbest* samt att Step består av ett antal regelbundna platåer, se Figur 5.4. Partiklarna i GPSO sprids runt omkring antingen *gbest* eller sina *pbest* med ett medelavstånd r. Om r är större än halva längden av en platå, är sannolikheten för en partikel att hamna på en lägre nivå ungefär

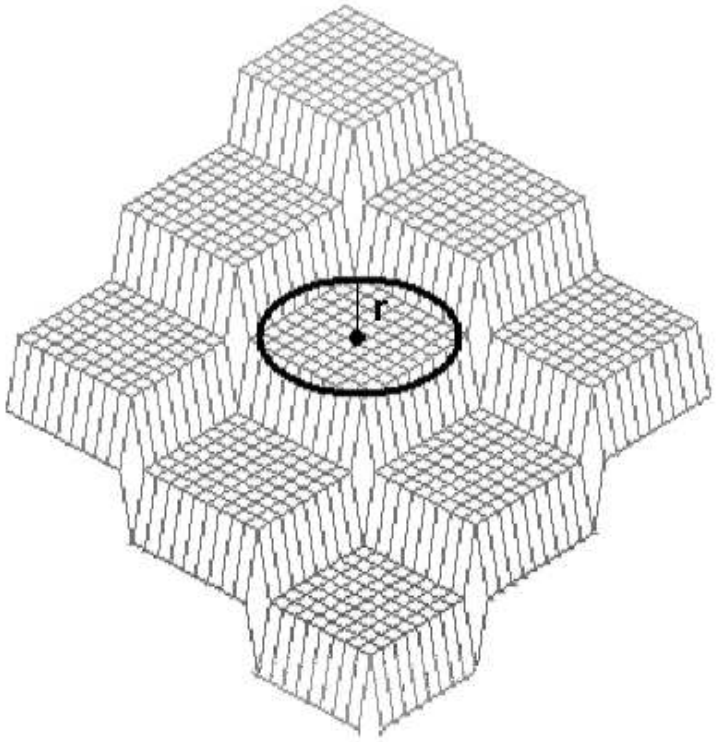
$$\frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$$

där $\frac{1}{2}$ kommer från att det är 50% chans för en partikel att hamna längre bort från *gbest* eller *pbest* än r och $\frac{1}{3}$ kommer från att ungefär $\frac{1}{3}$ av alla omkringliggande platåer är lägre, se Figur 5.10. För 30 partiklar är då sannolikheten att minst en partikel hamnar på en lägre nivå än dess *pbest*

$$1 - (1 - \frac{1}{6})^{30} \approx 0.996$$

vid varje iteration i GPSO. Det är då lätt att inse att GPSO kommer att konvergera snabbt för Step oavsett antal dimensioner.

- Eftersom en del av påbyggnadsdelen av experimentet avser funktioner av två dimensioner, är det lätt att jämföra med återupprepningsdelen. Av alla testproblem av två dimensioner var SPSO bättre än GPSO i alla fall utom två, vilket är i direkt motsats till resultatet från upprepningsdelen. Det finns två anledningar bakom GPSOs här



Figur 5.10: *Illustration av GPSO applicerat på* F_3

sämre resultat. Den första anledningen är att SPSO här har, till skillnad från del I, fått lika lång cpu-tid på sig som GPSO. Den andra anledningen är de skillnader mellan SPSO-algoritmerna som finns, se sektion 5.2.1 och sektion 5.3.1.

- Utifrån konvergeringskurvorna kan slutsatsen att GPSO överlag konvergerar långsammare än övriga PSO-algoritmer, speciellt SPSO, dras. För F_3 Step konvergerar dock GPSO oerhört snabbt och hittar i slutändan även de bästa lösningarna. GPSO konvergerar snabbt också för F_6 Griewangk men finner i slutändan inte lika bra lösningar som övriga PSO-algoritmer eftersom konvergeringshastigheten planar ut vilket tyder på att GPSO har svårt att undfly lokala optima. Ett långsamt konvergeringsförfarande behöver dock inte alltid vara en nackdel, en långsam konvergens kan tyda pa att tillståndsrummet genomsöks i en större omfattning vilket på sikt kan bidra till

att bättre lösningar erhålles.

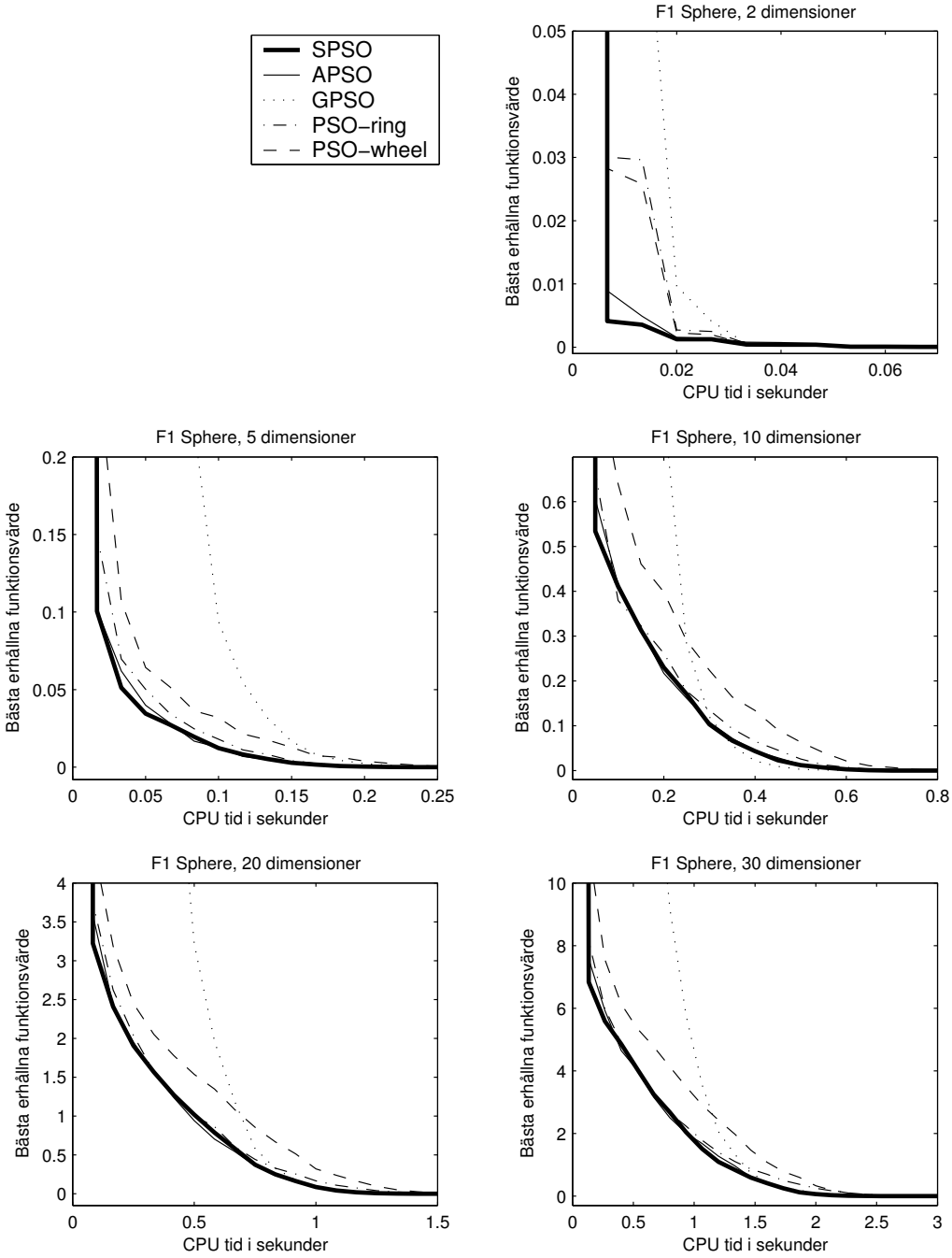
Det ska påpekas, vilket också nämnts tidigare, att parameterinställningarna för PSO-algoritmerna på intet sätt är optimala. De olika parametrarna såsom svärmstorleken kan anpassas efter vilket problem som ska optimeras, vilket skulle leda till förbättrade resultat. Med andra ord skulle samtliga PSO-algoritmer ha kunnat prestera bättre än de redovisade resultaten.

Mätvärden

F_1 Sphere

Dim./tid	*Metod*	*Bäst*	*Medel*	σ	*f-eval*	*Anm.*
30/4s	SPSO	6.3685e-019	1.1363e-016	2.4198e-016	78804	
	APSO	2.4549e-019	8.8861e-017	1.8541e-016	77989	Replace = 0
	GPSO	1.4946e-005	0.00010111	7.7239e-005	55346	
	PSO-ring	2.414e-009	2.1171e-008	3.0795e-008	73523	
	PSO-wheel	1.5546e-011	2.6708e-008	7.0238e-008	79046	
	FMINCON	1.032e-016	2.7371e-016	9.5637e-017	11917	
20/2.5s	SPSO	1.8652e-023	4.0973e-021	5.5708e-021	54595	
	APSO	4.2968e-024	3.3899e-020	6.9226e-020	53665	Replace = 0
	GPSO	4.6113e-007	3.5818e-006	2.5543e-006	35855	
	PSO-ring	1.595e-012	2.4087e-011	2.874e-011	51092	
	PSO-wheel	9.6184e-014	1.2276e-010	4.5038e-010	54607	
	FMINCON	9.6208e-017	2.1842e-016	1.1038e-016	7815	
10/1.5s	SPSO	6.2095e-035	1.125e-032	1.7066e-032	37009	
	APSO	1.2188e-034	5.2466e-031	1.4577e-030	35717	Replace = 0
	GPSO	6.6698e-013	5.0033e-011	9.5992e-011	22211	
	PSO-ring	1.375e-021	2.1602e-019	2.3037e-019	34135	
	PSO-wheel	2.2965e-021	8.699e-018	2.2075e-017	36930	
	FMINCON	7.0682e-017	1.2792e-016	7.2247e-017	4397	
5/0.5s	SPSO	4.2304e-026	1.9356e-023	4.618e-023	13291	
	APSO	1.1351e-024	4.3036e-021	1.5493e-020	12901	Replace = 0
	GPSO	4.9533e-013	5.1575e-010	1.0192e-009	7588	
	PSO-ring	5.9168e-018	1.06e-015	2.0668e-015	12144	
	PSO-wheel	1.7883e-019	8.7144e-012	4.3472e-011	13093	
	FMINCON	5.0749e-018	6.2846e-017	5.1572e-017	1041	
2/0.2s	SPSO	7.4425e-022	8.8794e-018	2.2995e-017	5326	
	APSO	2.2891e-021	1.1849e-016	4.0989e-016	5282	Replace = 0
	GPSO	7.9673e-023	6.5679e-016	1.6218e-015	2939	
	PSO-ring	1.6102e-017	1.1597e-012	5.7393e-012	4858	
	PSO-wheel	1.1163e-019	8.0604e-010	4.0024e-009	5315	
	FMINCON	3.3819e-018	5.4589e-017	3.0165e-017	504	

Tabell 5.7: *Tabell över resultat för F_1 Sphere.*

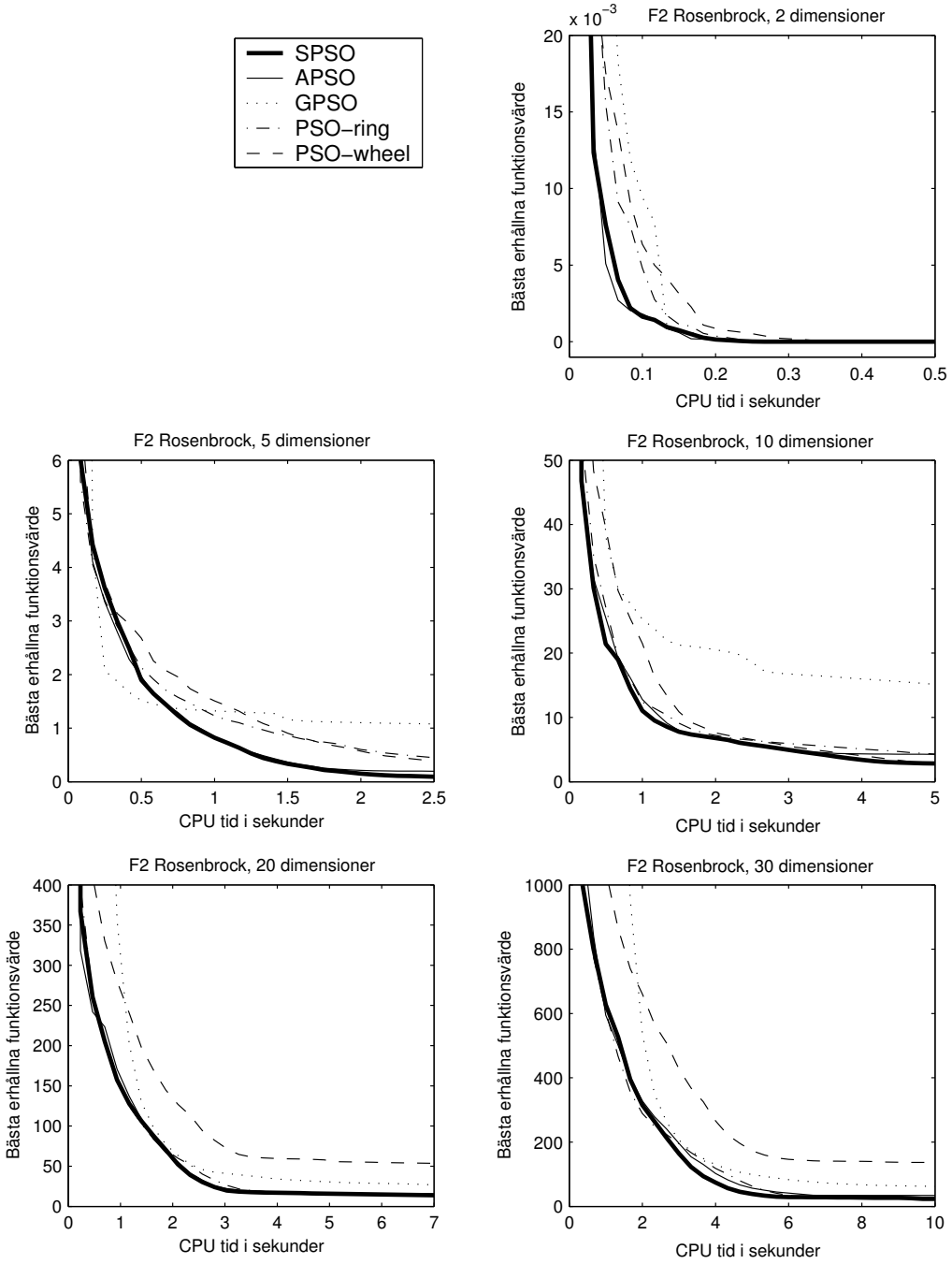


Figur 5.11: *Konvergeringskurvor för F_1 Sphere*

F_2 **Rosenbrock**

Dim./tid	*Metod*	*Bäst*	*Medel*	σ	*f-eval*	*Anm.*
30/10s	SPSO	12.5672	23.1267	3.2804	126104	
	APSO	16.7697	33.8942	19.5338	122608	Replace = 29
	GPSO	24.5652	62.4614	48.3111	97631	
	PSO-ring	24.245	26.2463	3.3196	117862	
	PSO-wheel	23.5561	136.4922	523.8324	123592	
	FMINCON	5.5702e-011	1.2757	1.898	30187	
20/7s	SPSO	12.0588	13.8551	1.6572	106164	
	APSO	10.3815	15.4749	1.3695	103819	Replace = 28
	GPSO	13.4347	26.7332	22.8829	77494	
	PSO-ring	14.1677	15.0599	0.83959	100573	
	PSO-wheel	11.3363	53.6503	137.2654	105587	
	FMINCON	4.8155e-011	1.4352	1.953	24317	
10/5s	SPSO	1.5804	2.8445	0.56002	97763	
	APSO	1.7354	4.2505	0.78316	95717	Replace = 16
	GPSO	2.2911	15.1382	26.1795	64051	
	PSO-ring	3.4761	4.3122	0.49601	91690	
	PSO-wheel	0.22201	2.7199	0.59482	97792	
	FMINCON	3.9383e-011	1.2757	1.898	15911	
5/2.5s	SPSO	0.0016165	0.094011	0.034923	57115	
	APSO	0.066101	0.194	0.071858	55882	Replace = 11
	GPSO	0.0021988	1.0787	1.8155	34781	
	PSO-ring	0.24082	0.45007	0.14493	53239	
	PSO-wheel	0.036998	0.39219	0.81424	57313	
	FMINCON	1.7604e-011	0.78617	1.6048	5770	
2/0.5s	SPSO	4.3972e-014	3.3784e-010	1.3784e-009	13056	
	APSO	1.2481e-014	1.6254e-010	3.5418e-010	12521	Replace = 0
	GPSO	2.8078e-016	4.6738e-011	1.3586e-010	7447	
	PSO-ring	1.2131e-011	4.9376e-008	1.1093e-007	12086	
	PSO-wheel	2.6096e-012	7.7412e-007	2.0057e-006	12968	
	FMINCON	6.5733e-013	1.6965e-011	5.8093e-012	776	

Tabell 5.8: *Tabell över resultat för F_2 Rosenbrock.*

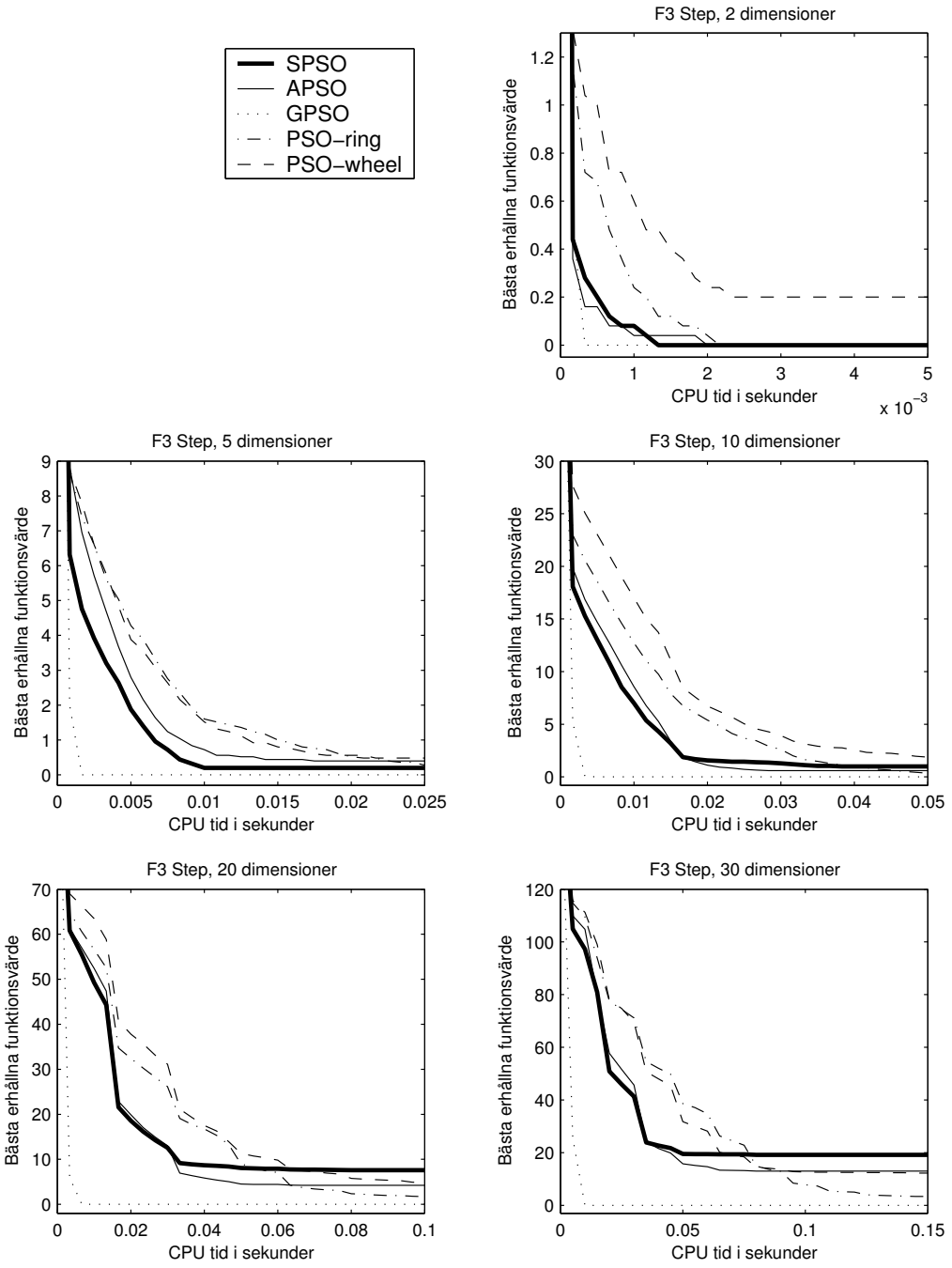


Figur 5.12: *Konvergeringskurvor för F_2 Rosenbrock*

F_3 **Step**

Dim./tid	*Metod*	*Bäst*	*Medel*	σ	*f-eval*	*Anm.*
30/0.15s	SPSO	0	19.2	12.8841	2540	
	APSO	0	13	10	2485	Replace = 23
	GPSO	0	0	0	1891	
	PSO-ring	0	3.4	5.5377	2316	
	PSO-wheel	0	12.28	8.6579	2497	
	FMINCON	115.6	140.92	16.8985	534	
20/0.1s	SPSO	0	7.6	7.2342	1880	
	APSO	0	4.2	5.2678	1852	Replace = 13
	GPSO	0	0	0	1370	
	PSO-ring	0	1.72	4.2575	1765	
	PSO-wheel	0	4.8	6.8981	1883	
	FMINCON	64.4	96.28	19.3247	348	
10/0.05s	SPSO	0	1	2.5	1218	
	APSO	0	0.6	2.1985	1210	Replace = 3
	GPSO	0	0	0	1032	
	PSO-ring	0	0.36	1.036	1142	
	PSO-wheel	0	1.88	3.3581	1172	
	FMINCON	33.2	46.84	7.7614	146	
5/0.025s	SPSO	0	0.2	1	1070	
	APSO	0	0.4	1.3844	1012	Replace = 3
	GPSO	0	0	0	1024	
	PSO-ring	0	0.28	0.79162	1040	
	PSO-wheel	0	0.48	1.3577	1072	
	FMINCON	18.6	24.04	3.8846	50	
2/0.005s	SPSO	0	0	0	1096	
	APSO	0	0	0	1096	Replace = 0
	GPSO	0	0	0	1018	
	PSO-ring	0	0	0	1031	
	PSO-wheel	0	0.2	0.57735	1075	
	FMINCON	2.24	8.68	4.0628	15	

Tabell 5.9: *Tabell över resultat för F_3 Step.*

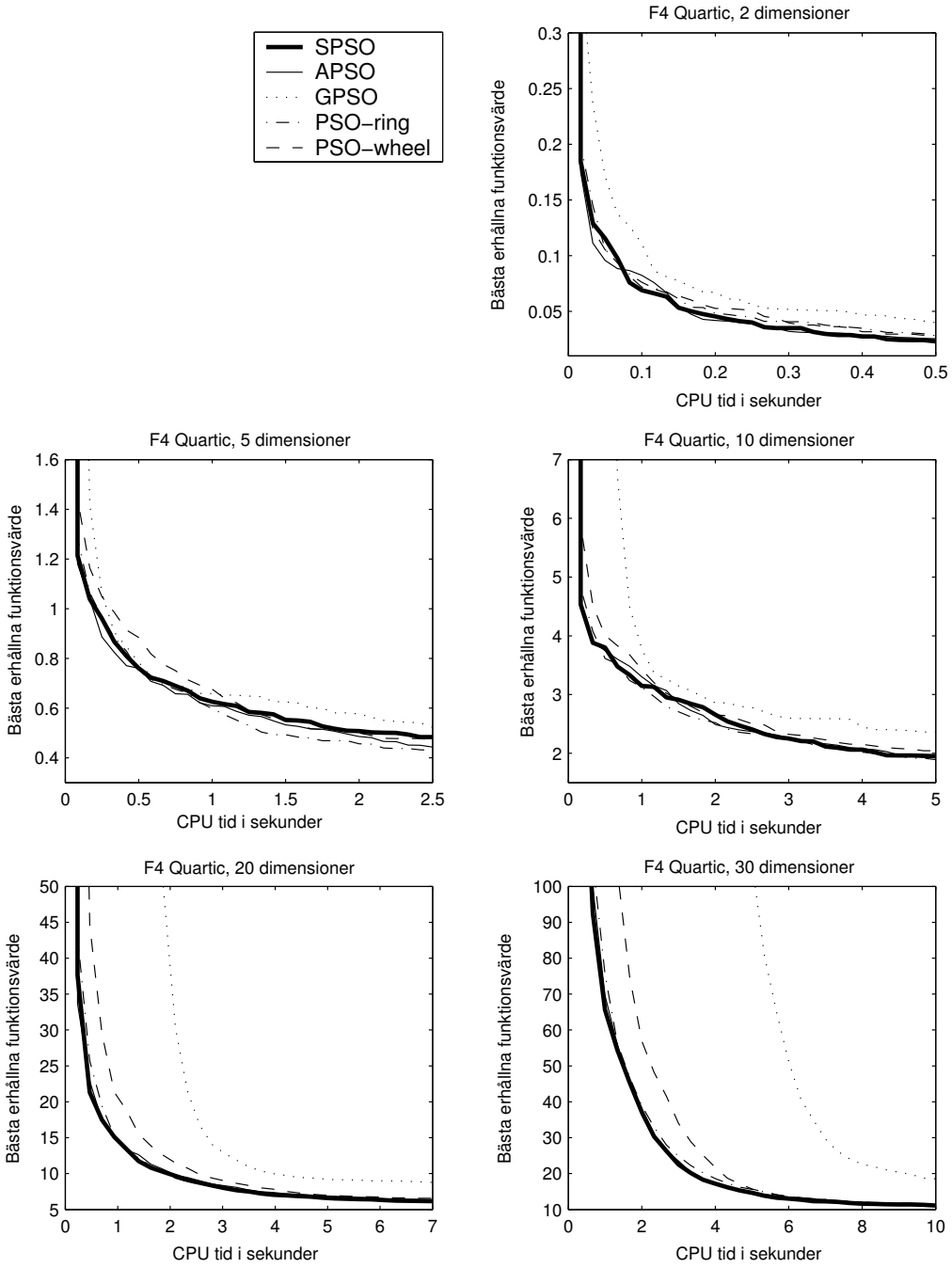


Figur 5.13: *Konvergeringskurvor för F_3 Step*

F_4 **Quartic**

Dim./tid	*Metod*	*Bäst*	*Medel*	σ	*f-eval*	*Anm.*
30/10s	SPSO	8.8234	11.0974	0.9033	32303	
	APSO	10.2903	11.3285	0.5731	32195	Replace = 0
	GPSO	13.1373	18.5402	2.5312	30094	
	PSO-ring	10.1019	11.2784	0.6515	32040	
	PSO-wheel	9.7000	11.3645	0.8660	38554	
	FMINCON	8.9027	32.4443	26.8087	7429	
20/7s	SPSO	4.5477	6.1603	0.5754	32777	
	APSO	5.5406	6.3392	0.4053	32809	Replace = 0
	GPSO	6.6995	8.7745	0.9453	29676	
	PSO-ring	5.5234	6.1728	0.3742	32382	
	PSO-wheel	5.9397	6.5503	0.3661	32872	
	FMINCON	5.5702	19.0793	16.913	9148	
10/5s	SPSO	1.3225	1.9523	0.26434	37087	
	APSO	1.4456	1.9327	0.20428	39233	Replace = 0
	GPSO	1.7744	2.3610	0.32219	30412	
	PSO-ring	1.4609	1.8935	0.23773	38786	
	PSO-wheel	1.5157	2.0426	0.25840	39148	
	FMINCON	1.6989	5.6738	4.6915	9006	
5/2.5s	SPSO	0.27385	0.48316	0.088039	33041	
	APSO	0.33295	0.44328	0.076439	32606	Replace = 0
	GPSO	0.31813	0.52242	0.1239	25117	
	PSO-ring	0.21991	0.43039	0.086944	30804	
	PSO-wheel	0.22292	0.47209	0.10826	30200	
	FMINCON	0.22286	1.5537	1.2504	5741	
2/0.5s	SPSO	0.0044971	0.022995	0.0099534	8585	
	APSO	0.0063724	0.025461	0.011695	8443	Replace = 0
	GPSO	0.013168	0.039886	0.019283	5747	
	PSO-ring	0.007604	0.028132	0.012602	8137	
	PSO-wheel	0.010419	0.026372	0.012849	8474	
	FMINCON	0.0239	0.64784	1.242	1157	

Tabell 5.10: *Tabell över resultat för F_4 Quartic.*

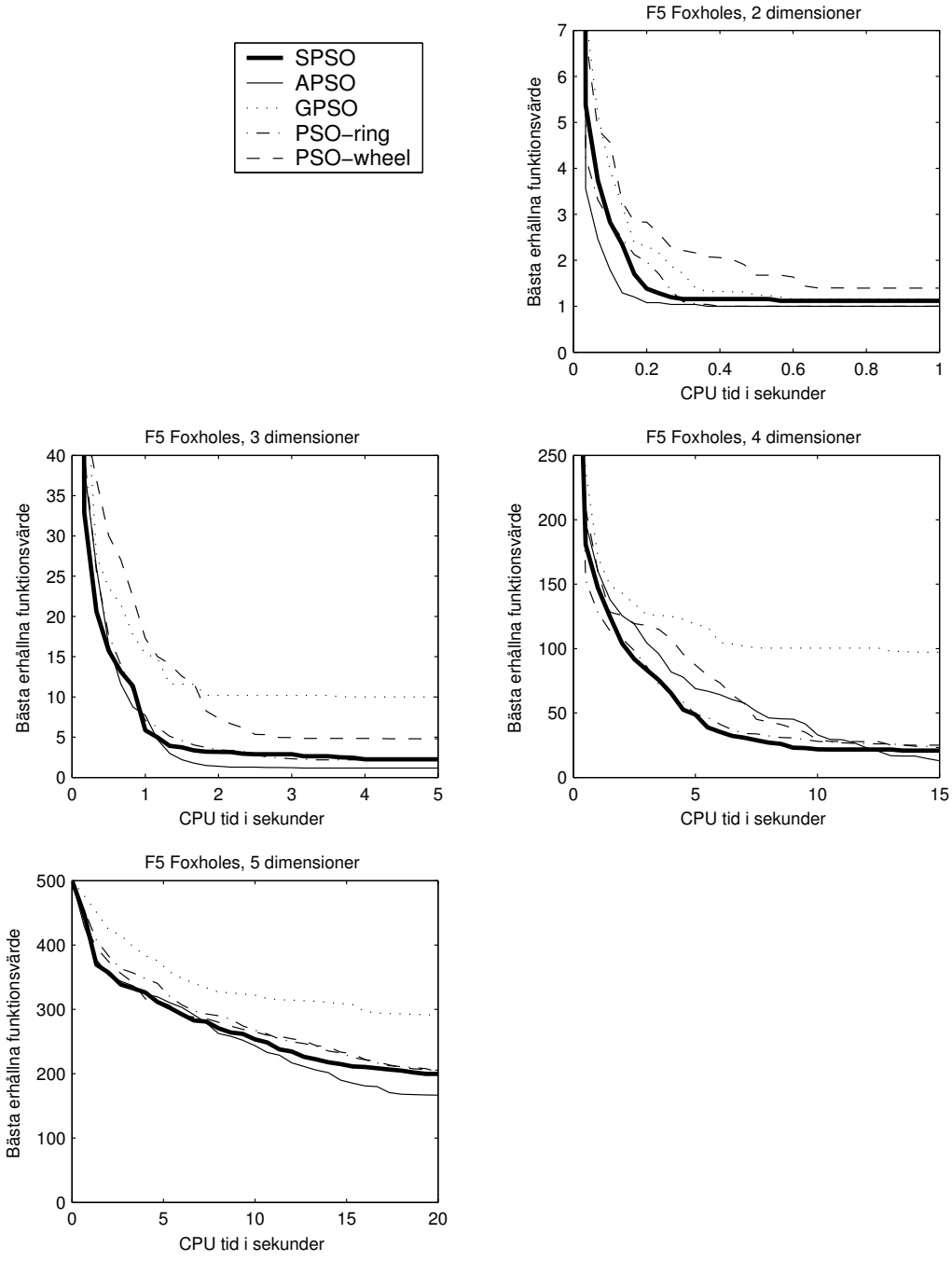


Figur 5.14: *Konvergeringskurvor för F_4 Quartic*

F_5 **Foxholes**

Dim./tid	*Metod*	*Bäst*	*Medel*	σ	*f-eval*	*Anm.*
5/20s	SPSO	59.8481	199.6517	95.7970	3359	
	APSO	18.3037	166.8027	94.4622	3391	Replace = 9
	GPSO	14.1531	291.5004	119.5967	3368	
	PSO-ring	92.4122	202.5823	85.2328	3397	
	PSO-wheel	36.1743	205.3424	112.5472	3398	
	FMINCON	65.9655	355.2172	107.0863	4613	
4/15s	SPSO	0.9980	20.7529	31.9617	13336	
	APSO	0.9980	12.9774	20.6105	13326	Replace = 28
	GPSO	6.9033	97.1814	65.6701	12985	
	PSO-ring	0.9980	23.0100	37.7200	13286	
	PSO-wheel	1.9920	25.0696	25.3365	13333	
	FMINCON	18.304	230.6059	133.2786	12938	
3/5s	SPSO	0.9980	2.2517	3.0264	22411	
	APSO	0.9980	1.1570	0.3719	22376	Replace = 29
	GPSO	0.9980	9.9995	10.5382	19874	
	PSO-ring	0.9980	2.1850	1.6613	22188	
	PSO-wheel	0.9980	4.7924	4.5403	22456	
	FMINCON	21.0726	59.2275	22.9545	7325	
2/1s	SPSO	0.9980	1.1168	0.5940	12096	
	APSO	0.9980	0.9980	0.0000	12052	Replace = 29
	GPSO	0.9980	1.1570	0.3719	9038	
	PSO-ring	0.9980	0.9980	0.0000	11759	
	PSO-wheel	0.9980	1.3951	0.7018	12132	
	FMINCON	0.998	14.9735	7.7105	1553	

Tabell 5.11: *Tabell över resultat för F_5 Foxholes.*

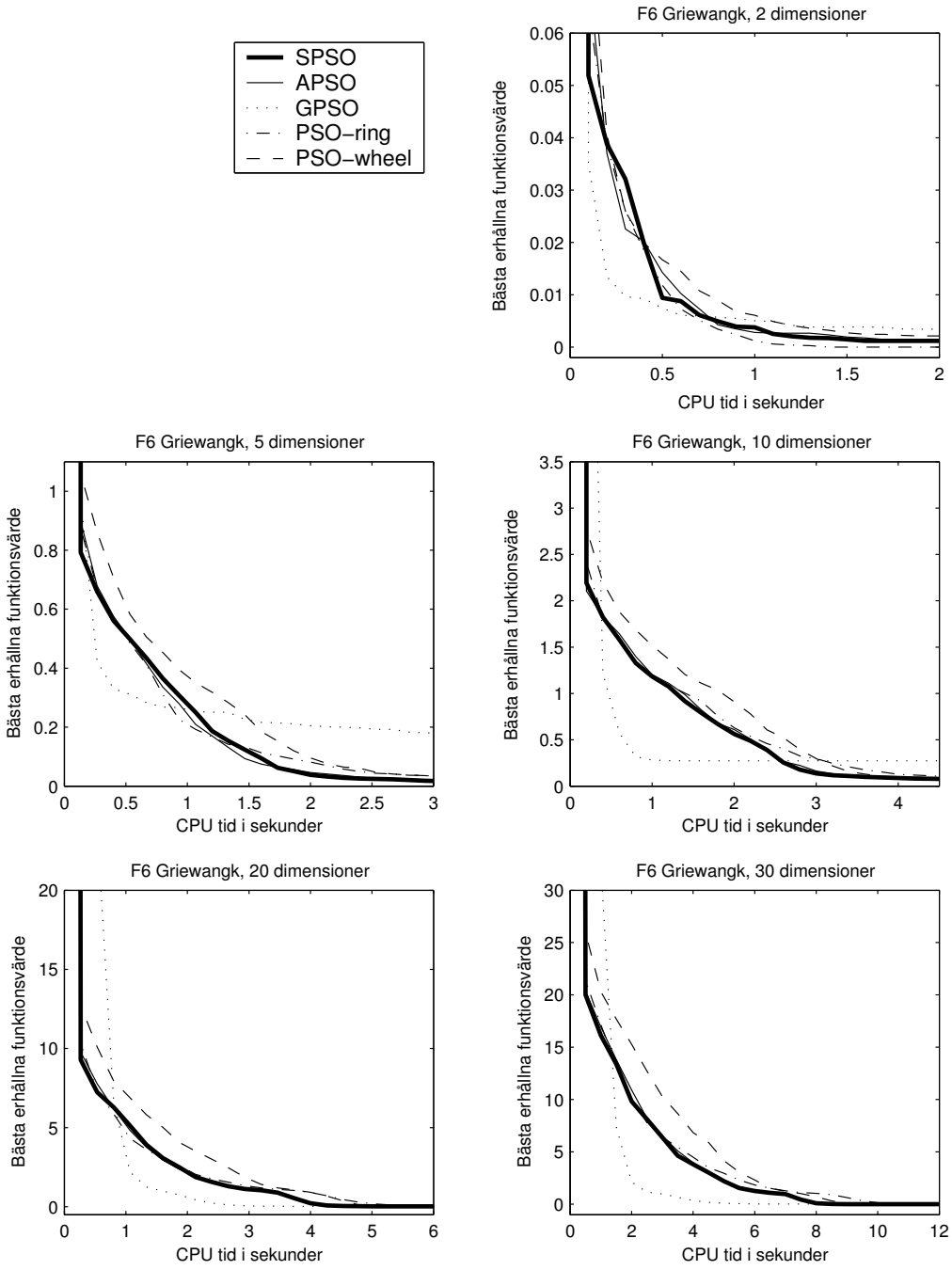


Figur 5.15: *Konvergeringskurvor för F_5 Foxholes*

F_6 **Griewangk**

Dim./tid	*Metod*	*Bäst*	*Medel*	σ	*f-eval*	*Anm.*
30/15s	SPSO	0	0.013862	0.014259	230231	
	APSO	0	0.011147	0.0092103	221737	Replace = 23
	GPSO	7.5763e-009	0.014071	0.012534	174588	
	PSO-ring	0	0.0095721	0.010251	216071	
	PSO-wheel	0	0.015359	0.011458	228335	
	FMINCON	0	1.1102e-016	1.0135e-016	13641	
20/8s	SPSO	0	0.028608	0.024585	128509	
	APSO	0	0.03529	0.027019	124708	Replace = 24
	GPSO	6.3674e-010	0.024481	0.024739	93667	
	PSO-ring	0	0.017626	0.018504	122506	
	PSO-wheel	0	0.023093	0.020019	129004	
	FMINCON	0	0.00029584	0.0014792	15965	
10/6s	SPSO	0.029547	0.070351	0.028075	120517	
	APSO	0.024594	0.064354	0.020376	118888	Replace = 26
	GPSO	0.056543	0.2734	0.1737	79471	
	PSO-ring	0.021546	0.07958	0.041236	113755	
	PSO-wheel	0.027066	0.072388	0.039553	120986	
	FMINCON	0	0.52377	0.80645	18495	
5/4s	SPSO	0	0.013697	0.0074242	83965	
	APSO	0	0.015169	0.0091999	82882	Replace = 23
	GPSO	0.059119	0.17885	0.0965	52963	
	PSO-ring	0.0012199	0.023361	0.011206	79282	
	PSO-wheel	0	0.02357	0.015546	84204	
	FMINCON	0.12326	23.423	39.9278	10120	
2/3s	SPSO	0	0.00059168	0.0020479	69294	
	APSO	0	0.00088753	0.002453	68270	Replace = 6
	GPSO	0	0.0031559	0.003996	41618	
	PSO-ring	0	3.153e-016	1.5765e-015	65460	
	PSO-wheel	0	0.0010939	0.0023479	69302	
	FMINCON	1.7234	55.407	35.8813	5263	

Tabell 5.12: *Tabell över resultat för F_6 Griewangk .*

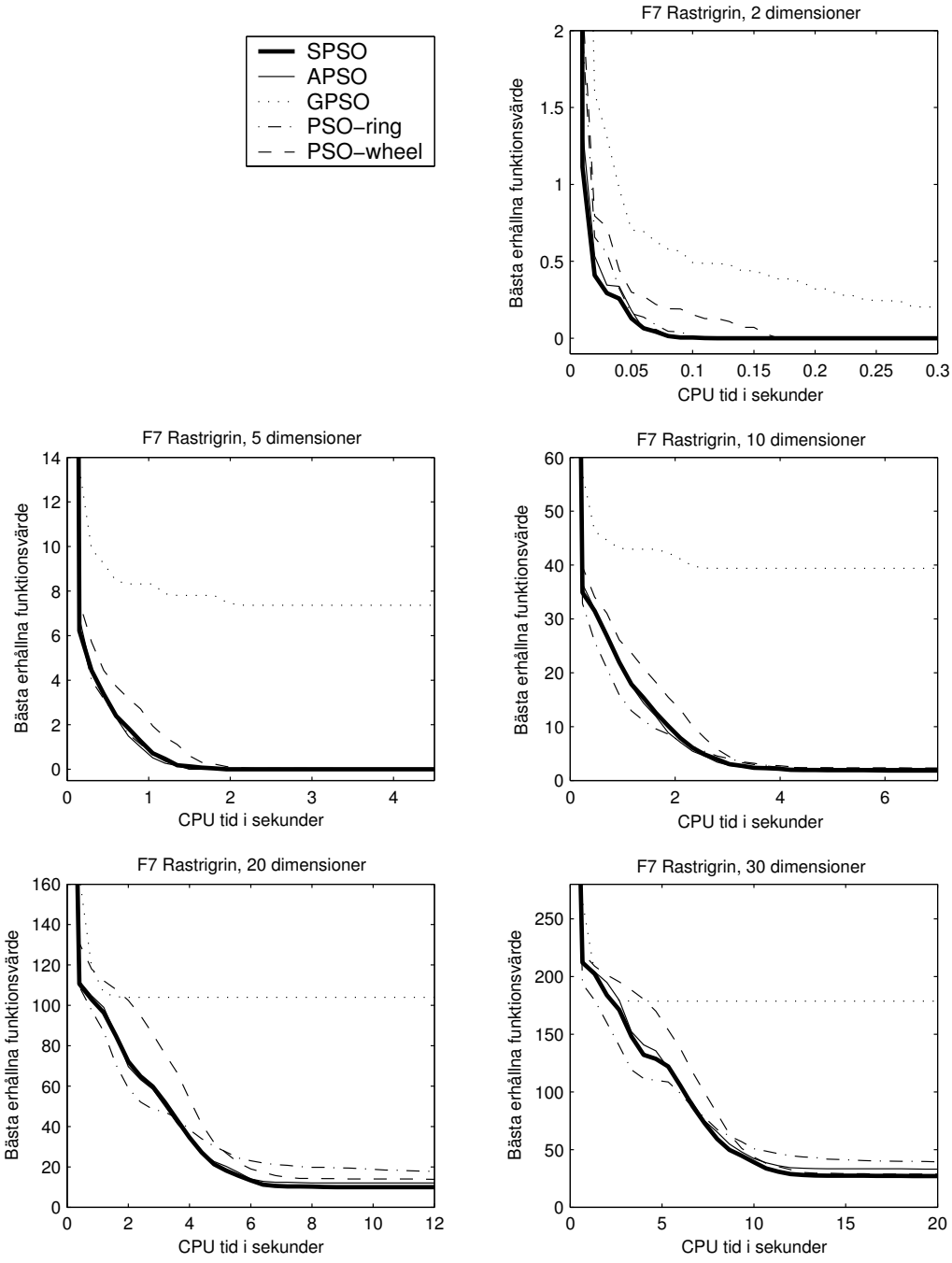


Figur 5.16: *Konvergeringskurvor för F_6 Griewangk*

F_7 **Rastrigrin**

Dim./tid	*Metod*	*Bäst*	*Medel*	σ	*f-eval*	*Anm.*
30/20s	SPSO	14.9244	27.1027	8.2811	326324	
	APSO	12.0205	33.0945	11.3055	302477	Replace = 29
	GPSO	120.3896	178.6141	26.7484	258385	
	PSO-ring	23.6748	39.4263	7.783	310894	
	PSO-wheel	11.9395	28.8225	12.9687	351340	
	FMINCON	119.3944	197.8365	41.0306	59004	
20/12s	SPSO	4.9748	9.9496	3.3495	242189	
	APSO	4.9759	11.9987	4.2854	234701	Replace = 29
	GPSO	52.7326	103.9926	25.5738	161320	
	PSO-ring	10.2114	17.7786	4.9111	219241	
	PSO-wheel	5.9698	13.8498	7.5878	240344	
	FMINCON	73.6266	145.3029	43.0802	47899	
10/7s	SPSO	0	1.8307	1.3678	165232	
	APSO	0	2.2015	1.2802	157572	Replace = 27
	GPSO	16.9143	39.4002	14.5436	99834	
	PSO-ring	0	1.6186	1.2389	157482	
	PSO-wheel	0	2.3083	1.7166	168894	
	FMINCON	33.8285	73.0296	21.478	22154	
5/4.5s	SPSO	0	0	0	114054	
	APSO	0	0	0	111703	Replace = 0
	GPSO	1.9899	7.3627	3.5177	66314	
	PSO-ring	0	0	0	105550	
	PSO-wheel	0	0	0	112672	
	FMINCON	3.9798	30.6048	16.7186	9899	
2/0.3s	SPSO	0	0	0	8346	
	APSO	0	0	0	8075	Replace = 0
	GPSO	0	0.20202	0.49645	4552	
	PSO-ring	0	0	0	7424	
	PSO-wheel	0	1.0051e-012	4.5542e-012	8005	
	FMINCON	0.99496	9.7903	8.7763	488	

Tabell 5.13: *Tabell över resultat för F_7 Rastrigrin.*

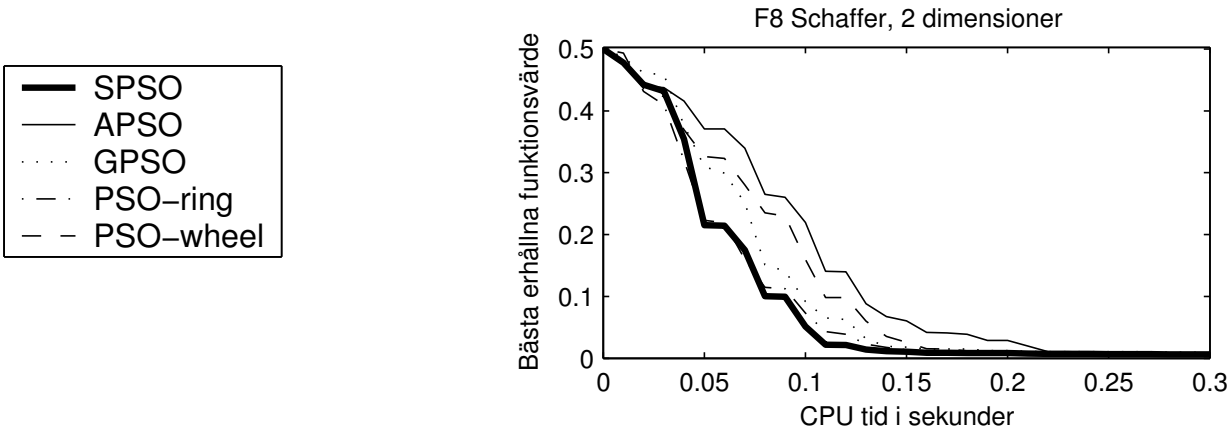


Figur 5.17: *Konvergeringskurvor för F_7 Rastrigrin*

F_8 **Schaffer**

Dim./tid	*Metod*	*Bäst*	*Medel*	σ	*f-eval*	*Anm.*
2/0.3s	SPSO	0	0.0062571	0.0047105	8464	
	APSO	3.7492e-013	0.0073844	0.0042352	7972	Replace = 830
	GPSO	0.002223	0.0094162	0.0014986	4676	
	PSO-ring	1.2341e-009	0.0054522	0.0046518	7636	
	PSO-wheel	6.0683e-007	0.0075833	0.0032983	8378	
	FMINCON	0.5	0.5	1.9434e-007	425	

Tabell 5.14: *Tabell över resultat för F_8 Schaffer.*



Figur 5.18: *Konvergeringskurvor för F_8 Schaffer*

5.5.3 Modellerar GPSO en partikelsvärm?

En fråga som dök upp under arbetets gång var huruvida GPSO verkligen modellerar en partikelsvärm. Utan tvekan utnyttjas kollektiv intelligens, då partiklars nya positioner beror på svärmens erfarenheter, men det finns ingen egentlig rörelse som i övriga PSO-algoritmer. Partikelsvärmsoptimering baseras på en flock flygande fåglar, och varje partikel har tidigare haft en associerad position och hastighet - partiklarna rör sig kontinuerligt och deras

hastigheter förändras kontinuerligt. I GPSO däremot placeras partiklar ut slumpvis runtomkring *gbest* eller partiklarnas *pbest* utan att ta partiklarnas tidigare position i beaktande. Rörelsemönstret hos partiklarna i GPSO kan således inte liknas med en flock flygande fåglar. GPSO modellerar alltså inte en partikelsvärm i ordets ursprungliga betydelse, men räknas algoritmen trots det som en partikelsvärmsalgoritm?

Kapitel 6

Slutsats och sammanfattande kommentarer

Optimering är en iterativ process vilken ämnar fatta ett så bra beslut som möjligt med hjälp av matematiska modeller och metoder, och har idag ett stort antal tekniska och ekonomiska tillämpningsområden. Optimering används ofta för att lösa komplexa problem - som det tar alltför lång tid att lösa med traditionella, exakta, metoder - ungefärligt eller ”tillräckligt bra”. Svärmintelligens, SI, är ett område inom Artificiell Intelligens, AI, som bygger på kollektiv intelligens för optimering. SI bygger på hur en grupp individer samarbetar för att lösa en specifik uppgift, i fallet med den här uppsatsen söka optimum till en funktion. Intelligensen höjs då en nivå till att inkludera samtliga individer i en grupp och deras samarbetsförmåga, istället för att enbart innefatta en enskild individ.

SI kan delas upp i två olika områden, partikelsvärmsoptimering och insektsvärmsoptimering. Insektsvärmsoptimering bygger på först och främst myrors feromonföljande beteende, och har visat sig vara effektivt vid lösandet av diskreta problem. Insektsvärmsoptimering går dock inte att överföra på kontinuerliga problem utan svårigheter, och av den anledningen ligger uppsatsens fokus på partikelsvärmsoptimering som visat sig vara utmärkt vid lösandet av kontinuerliga problem. Partikelsvärmsoptimering bygger på en flock

fåglars sociala beteende inom flocken, där en enskild fågel benämns som en partikel och flocken benämns som svärmen. Idén är här att de enskilda partiklarna rör sig genom det flerdimensionella tillståndsrummet i sökandet efter det globala optimat och deras rörelse påverkas av både deras egna och gruppens erfarenheter.

Det finns ett antal algoritmer inom partikelsvärmsoptimering vilka alla mer eller mindre liknar varandra. De i uppsatsen undersökta algoritmerna är SPSO, APSO, PSO med ring-topologi, PSO med wheel-topologi samt GPSO. GPSO utvecklades så sent som 2003 av Barry R. Secrest och Gary B. Lamont och är den algoritm av de tidigare nämnda som avviker mest från de övriga algoritmerna. Secrest och Lamont utvecklade GPSO för att tackla några svagheter som de upptäckt hos andra PSO-algoritmer. Secrest och Lamont visade genom experiment att GPSO var överlägsen SPSO vid lösning av fem olika ickelinjära begränsade tvådimensionella problem. Secrest och Lamont ansåg dock att mer forskning krävdes för att bekräfta deras resultat och för att undersöka om resultatet blev detsamma för problem av ett högre dimensionsantal. Secrest och Lamont nämnde även att den version av SPSO som de använt saknade en del vanliga förbättringar, och av denna anlendning ansåg Secrest och Lamont att det var för tidigt att säga att GPSO överlag var överlägsen SPSO.

Med Secrests och Lamonts artikel som grund utförde vi ett experiment för att undersöka effektiviteten hos GPSO gentemot SPSO och andra tidigare nämna algoritmer vid lösning av ickelinjära begränsade flerdimensionella problem. Efter att först ha återupprepat Secrests och Lamonts experiment, visade vi genom nya experiment att SPSO var bättre än GPSO utom i några enstaka specifika fall. Vidare noterades även en överlag snabbare konvergens för SPSO än för GPSO. De olika PSO-algoritmerna jämfördes även med en SQP-metod för att ha en referenspunkt att jämföra PSO-algoritmerna som grupp med. SQP är en erkänt bra metod för lösning av ickelinjära begränsade problem och den SQP-implementation som användes för jämförelsen var FMINCON från MathWorks Inc.. Det visade sig att SPSO var bättre än FMINCON utom i några enstaka specifika fall, och att

samtliga PSO-algoritmer var bättre än FMINCON i de flesta fallen.

Slutsatserna från vårt experiment är att GPSO överlag inte medför någon förbättring gentemot SPSO och att samtliga PSO-algoritmer, speciellt SPSO, överlag är minst lika effektiva som FMINCON i de undersökta testfallen.

Referenser

[1] James Kennedy & Russell C. Eberhart. *A new optimizer using Particle Swarm theory.* Proc. of the sixth international symposium on micro machine and human science, 1995:39-43.

[2] James Kennedy & Russell C. Eberhart. *Swarm Intelligence.* Morgan Kaufmann Publishers, 2001.

[3] Barry R. Secrest & Gary B. Lamont. *Visualizing Particle Swarm Optimization - Gaussian Particle Swarm Optimization.* IEEE Proc. of the Swarm Intelligence Symposium, 2003:198-204.

[4] David G. Luenberger. *Linear and Nonlinear Programming.* Kluwer Academic Publushers, 2nd Edition, 2003.

[5] Jan Lundgren, Mikael Rönnqvist, Peter Värbrand. *Linjär och ickelinjär optimering.* Studentlitteratur, 2001.

[6] Jorge Nocedal & Stephen J. Wright. *Numerical Optimization.* Springer, 1999.

[7] Marylin McCord Nelson & W.T. Illingworth. *A Practical Guide to Neural Nets.* Addison-Wesley, 1991.

[8] Andries P. Engelbrecht. *Computational Intelligence: An Introduction.* John Wiley & Sons Ltd, 2002.

[9] Eric Bonabeau, Marco Dorigo, Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, 1999.

[10] Zbigniew Michalewicz & D. B. Fogel. *How to Solve It: Modern Heuristics.* Springer, 2000.

[11] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 3rd edition, 1996.

[12] Yuhui Shi & Renato A. Krohling. *Co-evolutionary Particle Swarm Optimization to Solve min-max Problems.* IEEE Proc. of Evolutionary Computation, Vol.2, 2002:1682-1687.

[13] Russell C. Eberhart & Yuhui Shi. *Particle Swarm Optimization: Developments, Applications and Resources.* IEEE Proc. of Congress on Evolutionary Computation, 2001:81-86.

[14] E.C. Laskari, K. E. Parsopoulos, M. N. Vrahatis. *Particle Swarm Optimization for Minimax Problems.* IEEE Proc. of Congress on Evolutionary Computation, 2002:1576-1581.

[15] Xiao-Feng Xie, Wen-Jun Zhang, Zhi-Lian Yang. *Adaptive Particle Swarm Optimization on Individual Level.* IEEE 6th Int. Conf. on Signal Processing, 2002:1215-1218.

[16] Luk Schoofs & Bart Naudts. *Swarm Intelligence on the Binary Contraint Satisfaction problem.* IEEE Proc. of Congress on Evolutionary Computation, 2002:1444-1449.

[17] Marco Dorigo, Gianni Di Caro, Luca M. Gambardella. *Ant Algorithms for Discrete Optimization.* Artificial Life 5(3), 1999:137-172.

[18] Privat samtal med Andries P. Engelbrecht, 2004-02-24.

[19] Wei-Qing Xiong & Ping Wei. *A kind of ant colony algorithm for function optimization.* IEEE Proc. of Congress on Machine Learning and Cybernetics, 2002:552-555.

[20] Lei Wang & Qidi Wu. *Ant System Algorithm for Optimization in Continuous Space.* IEEE Proc. of Conf. on Control Applications, 2001:395-400.

[21] Lei Wang, Xiao-Ping Wang, Qidi Wu. *Ant System Algorithm based Rosenbrock function optimization in multi-dimension space.* IEEE Proc. of Congress on Machine Learning and Cybernetics, 2002:710-714.

[22] Eva Pärt-Enander & Anders Sjöberg. *The MATLAB 5 Handbook.* Addison-Wesley, 1999.

[23] The MathWorks. *Optimization Toolbox: User's Guide.* The MathWorks Inc, Version 2.1, 2000.

[24] Kenneth Alan De Jong. *Analysis of the behavior of a class of genetic adaptive systems.* University of Michigan, Logic of Computers Group, 1975.

[25] Zachary V. Hendershot. *A Differential Evolution Algorithm for Automatically Discovering Multiple Global Optima in Multidimensional, Discontinuous Spaces.* Proc. of the Midwest Artificial Intelligence and Cognitive Science Conference, 2004:92-97.

[26] Jason Digalakis & Konstantinos Margaritis. *Performance comparison of memetic algorithms.* Journal of Applied Mathematics and Computation, 2003.

Bilaga A

MATLAB kod

A.1 Standard PSO

```
% ------------------------------------------------------
% SPSO Algoritm
%   Av:    Mathias Johansson, Mikael Lindberg
%   Datum: 2004-10-10
% ------------------------------------------------------
% Parameterbeskrivning
%   func     - Funktion att optimera
%   N        - Antal partiklar i svärmen
%   D        - Antal dimensioner av problemet
%   u        - Begränsning av värdena på problemets
%              inparameter, för alla x vilka tillhör
%              {X1...Xn} hör till intervallet [-u, u]
%   stoptime - Antal sekunder algoritmen ska exekvera
%   nroflogs - Antal logg tillfällen exkl startvärde
% ------------------------------------------------------

function SPSO(func, N, D, u, stoptime, nroflogs);

% Initiering av startvärden
starttime = cputime;
V_MAX = u * 0.3;        % Det samma som 15%
W_start = 0.9;          % Tröghetsvikt att börja med (Inertia weight)
W_stop = 0.4;           % Tröghetsvikt att sluta med (Inertia weight)
W_cur = W_start;        % Nuvarande tröghetsvikt (Inertia weight)
C1 = 2;                 % Accelerationskonstant
```

```
C2 = 2;                 % Accelerationskonstant
log = 1;                % Inv: 1 <= log <= nroflogs

% Initierar partiklar och deras hastighet
for i = 1: N
    for j = 1: D
        p(i,j) = rand * u * 2 - u;
        v(i,j) = rand * V_MAX * 2 - V_MAX;
    end
end
pbest = p;
gbest = 1;
for j = 1 : N
    curfeval(j) = feval(func, p(j,:));
    pbestfeval(j) = curfeval(j);
end

% Logga initialvärden
data(log) = pbestfeval(gbest);
time(log) = 0;

% Huvudloop, avbryts efter en förutbestämd tid
while log <= nroflogs

    % Uppdaterar personbästa för varje partikel
    for j = 1 : N
        curfeval(j) = feval(func, p(j,:));
        if curfeval(j) < pbestfeval(j)
            pbest(j,:) = p(j,:);
            pbestfeval(j) = curfeval(j);
        end
    end

    % Uppdaterar globalt bästa partikel
    for j = 1 : N
        if pbestfeval(gbest) > curfeval(j)
            gbest = j;
        end
    end

    % Förflytta partiklarna genom tillståndsrummet
```

```
    for j = 1 : N

        % Beräkna ny hastighet för partikel j
        for k = 1 : D
            v(j,k) = W_cur * v(j,k) + C1 * rand * (pbest(j,k)-p(j,k))
                     + C2 * rand * (pbest(gbest,k)-p(j,k));
            if v(j,k) > V_MAX
                v(j,k) = V_MAX;
            elseif v(j,k) < -V_MAX
                v(j,k) = -V_MAX;
            end
        end

        % Beräkna nytt tillstånd för partikel j
        for k = 1 : D
            p(j,k) = p(j,k) + v(j,k);
            if p(j,k) > u
                p(j,k) = u;
            elseif p(j,k) < -u
                p(j,k) = -u;
            end
        end
    end

    % Uppdatera förfluten tid
    curtime = cputime - starttime;

    % Uppdatera tröghetsvikt
    W_cur = W_start - (W_start - W_stop) / (stoptime / curtime);

    % Logga algoritmens framgång
    if (stoptime / nroflogs) * log < curtime
        data(log + 1) = pbestfeval(gbest);
        time(log + 1) = curtime;
        log = log + 1;
    end
end

assignin('base','data', data);
assignin('base','time', time);
```

A.2 Adaptive PSO

```
% ------------------------------------------------------
% APSO Algoritm
%   Av:    Mathias Johansson, Mikael Lindberg
%   Datum: 2004-10-10
% ------------------------------------------------------
% Parameterbeskrivning
%   func     - Funktion att optimera
%   N        - Antal partiklar i svärmen
%   D        - Antal dimensioner av problemet
%   u        - Begränsning av värdena på problemets
%              inparameter, för alla x vilka tillhör
%              {X1...Xn} hör till intervallet [-u, u]
%   stoptime - Antal sekunder algoritmen ska exekvera
%   nroflogs - Antal logg tillfällen exkl startvärde
% ------------------------------------------------------

function APSO(func, N, D, u, stoptime, nroflogs);

% Initiering av startvärden
starttime = cputime;
diffp = 0.001;              % 0.1 Procent av det bäst funna funktionsvärdet
                            % vilket används vid kontroll av inaktiva
                            % partiklar
V_MAX = u * 0.3;            % Det samma som 15%
C1 = 2;                     % Accelerationskonstant
C2 = 2;                     % Accelerationskonstant
invcount = zeros(1,N);      % Vektor som håller reda på hur länge
                            % varje partikel varit "inaktiv"
Tc = 3;                     % Max antal iterationer en partikel får
                            % vara inaktiv innan den byts ut
replace = 0;                % Räknar antalet utbytta partiklar
W_start = 0.9;              % Tröghetsvikt att börja med (Inertia weight)
W_stop = 0.4;               % Tröghetsvikt att sluta med (Inertia weight)
W_cur = W_start;            % Nuvarande tröghetsvikt (Inertia weight)
log = 1;                    % Inv: 1 <= log <= nroflogs

% Initierar partiklar och dess hastighet
for i = 1: N
    for j = 1: D
```

```
        p(i,j) = rand * u * 2 - u;
        v(i,j) = rand * V_MAX * 2 - V_MAX;
    end
end
pbest = p;
gbest = 1;
for j = 1 : N
    curfeval(j) = feval(func, p(j,:));
    pbestfeval(j) = curfeval(j);
end

% Logga initialvärden
data(log) = pbestfeval(gbest);
time(log) = 0;

% Huvudloop, avbryts efter en förutbestämd tid
while log <= nroflogs

    % Uppdaterar personbästa för varje partikel
    for j = 1 : N
        curfeval(j) = feval(func, p(j,:));
        if curfeval(j) < pbestfeval(j)
            pbest(j,:) = p(j,:);
            pbestfeval(j) = curfeval(j);
        end
    end

    % Uppdaterar globalt bästa partikel
    for j = 1 : N
        if pbestfeval(gbest) > curfeval(j)
            gbest = j;
        end
    end

    % Uppdatera gränsen för inaktivitetskravet
    diff = pbestfeval(gbest) * diffp;

    % Förflytta partiklarna genom tillståndsrummet
    for j = 1 : N

        % Uppdatera inaktivitetsräknaren för partikel j
```

```
        if j ~= gbest && abs(curfeval(j) - pbestfeval(gbest)) < diff
            invcount(j) = invcount(j) + 1;
        else
            invcount(j) = 0;
        end

        % Byt ut partikel j om den varit inaktiv för länge, annars
        % förflytta partkel j
        if invcount(j) > Tc
            for k = 1 : D
                p(j,k) = rand * u * 2 - u;
                v(j,k) = rand * V_MAX * 2 - V_MAX;
            end
            pbest(j, :) = p(j, :);
            replace = replace + 1;
        else

            % Beräkna ny hastighet för partikel j
            for k = 1 : D
                v(j,k) = W_cur * v(j,k) + C1 * rand * (pbest(j,k)-p(j,k))
                         + C2 * rand * (pbest(gbest,k)-p(j,k));
                if v(j,k) > V_MAX
                    v(j,k) = V_MAX;
                elseif v(j,k) < -V_MAX
                    v(j,k) = -V_MAX;
                end
            end

            % Beräkna nytt tillstånd för partikel j
            for k = 1 : D
                p(j,k) = p(j,k) + v(j,k);
                if p(j,k) > u
                    p(j,k) = u;
                elseif p(j,k) < -u
                    p(j,k) = -u;
                end
            end
        end
    end

    % Uppdatera förfluten tid
```

```
    curtime = cputime - starttime;

    % Uppdatera tröghetsvikt
    W_cur = W_start - (W_start - W_stop) / (stoptime / curtime);

    % Logga algoritmens framgång
    if (stoptime / nroflogs) * log < curtime
        data(log + 1) = pbestfeval(gbest);
        time(log + 1) = curtime;
        log = log + 1;
    end
end

assignin('base','data', data);
assignin('base','time', time);
assignin('base','replace', replace);
```

A.3 PSO med ring-topologi

```
% ------------------------------------------------
% PSO Algoritm med ring-topologi
%   Av:    Mathias Johansson, Mikael Lindberg
%   Datum: 2004-10-10
% ------------------------------------------------
% Parameterbeskrivning
%   func     - Funktion att optimera
%   N        - Antal partiklar i svärmen
%   D        - Antal dimensioner av problemet
%   u        - Begränsning av värdena på problemets
%              inparameter, för alla x vilka tillhör
%              {X1...Xn} hör till intervallet [-u, u]
%   stoptime - Antal sekunder algoritmen ska exekvera
%   nroflogs - Antal logg tillfällen exkl startvärde
% ------------------------------------------------

function PSOring(func, N, D, u, stoptime, nroflogs);

% Initiering av startvärden
starttime = cputime;
V_MAX = u * 0.3;        % Det samma som 15%
```

```
W_start = 0.9;          % Tröghetsvikt att börja med (Inertia weight)
W_stop = 0.4;           % Tröghetsvikt att sluta med (Inertia weight)
W_cur = W_start;        % Nuvarande tröghetsvikt (Inertia weight)
C1 = 2;                 % Accelerationskonstant
C2 = 2;                 % Accelerationskonstant
log = 1;                % Inv: 1 <= log <= nroflogs
n = round(N * 0.15);    % Omgivningsstorlek, 15% av svärmstorlek

% Initierar partiklar och dess hastighet
for i = 1: N
    for j = 1: D
        p(i,j) = rand * u * 2 - u;
        v(i,j) = rand * V_MAX * 2 - V_MAX;
    end
    lbest(i) = i;
end
pbest = p;
gbest = 1;
for j = 1 : N
    curfeval(j) = feval(func, p(j,:));
    pbestfeval(j) = curfeval(j);
end

% Logga initialvärden
data(log) = pbestfeval(gbest);
time(log) = 0;

% Huvudloop, avbryts efter en förutbestämd tid
while log <= nroflogs

    % Uppdaterar personbästa för varje partikel
    for j = 1 : N
        curfeval(j) = feval(func, p(j,:));
        if curfeval(j) < pbestfeval(j)
            pbest(j,:) = p(j,:);
            pbestfeval(j) = curfeval(j);
        end
    end

    % Uppdaterar lokalt bästa partikel för varje partikel
    for j = 1 : N
```

```
        for k = j - 1 : j + n - 1
            if curfeval(mod(k,N)+1) < pbestfeval(lbest(j))
                lbest(j) = mod(k,N)+1;
            end
        end
        if pbestfeval(gbest) > pbestfeval(j)
            gbest = j;
        end
    end

    % Förflytta partiklarna genom tillståndsrummet
    for j = 1 : N

        % Beräkna ny hastighet för partikel j
        for k = 1 : D
            v(j,k) = W_cur * v(j,k) + C1 * rand * (pbest(j,k)-p(j,k))
                     + C2 * rand * (pbest(lbest(j),k)-p(j,k));
            if v(j,k) > V_MAX
                v(j,k) = V_MAX;
            elseif v(j,k) < -V_MAX
                v(j,k) = -V_MAX;
            end
        end

        % Beräkna nytt tillstånd för partikel j
        for k = 1 : D
            p(j,k) = p(j,k) + v(j,k);
            if p(j,k) > u
                p(j,k) = u;
            elseif p(j,k) < -u
                p(j,k) = -u;
            end
        end
    end

    % Uppdatera förfluten tid
    curtime = cputime - starttime;

    % Uppdatera tröghetsvikt
    W_cur = W_start - (W_start - W_stop) / (stoptime / curtime);
```

```
    % Logga algoritmens framgång
    if (stoptime / nroflogs) * log < curtime
        data(log + 1) = pbestfeval(gbest);
        time(log + 1) = curtime;
        log = log + 1;
    end
end

assignin('base','data', data);
assignin('base','time', time);
```

A.4 PSO med wheel-topologi

```
% ------------------------------------------------------
% PSO Algoritm med hjul-topologi (wheel)
%   Av:    Mathias Johansson, Mikael Lindberg
%   Datum: 2004-10-10
% ------------------------------------------------------
% Parameterbeskrivning
%   func     - Funktion att optimera
%   N        - Antal partiklar i svärmen
%   D        - Antal dimensioner av problemet
%   u        - Begränsning av värdena på problemets
%              inparameter, för alla x vilka tillhör
%              {X1...Xn} hör till intervallet [-u, u]
%   stoptime - Antal sekunder algoritmen ska exekvera
%   nroflogs - Antal logg tillfällen exkl startvärde
% ------------------------------------------------------

function PSOwheel(func, N, D, u, stoptime, nroflogs);

% Initiering av startvärden
starttime = cputime;
V_MAX = u * 0.3;        % Det samma som 15%
W_start = 0.9;          % Tröghetsvikt att börja med (Inertia weight)
W_stop = 0.4;           % Tröghetsvikt att sluta med (Inertia weight)
W_cur = W_start;        % Nuvarande tröghetsvikt (Inertia weight)
C1 = 2;                 % Accelerationskonstant
C2 = 2;                 % Accelerationskonstant
```

```
focal = 1;              % Sätt första partikeln till fokalpartikel
log = 1;                % Inv: 1 <= log <= nroflogs

% Initierar partiklar och dess hastighet
for i = 1: N
    for j = 1: D
        p(i,j) = rand * u * 2 - u;
        v(i,j) = rand * V_MAX * 2 - V_MAX;
    end
end
pbest = p;
gbest = 1;
for j = 1 : N
    curfeval(j) = feval(func, p(j,:));
    pbestfeval(j) = curfeval(j);
end

% Logga initialvärden
data(log) = pbestfeval(gbest);
time(log) = 0;

% Huvudloop, avbryts efter en förutbestämd tid
while log <= nroflogs

    % Uppdaterar personbästa för varje partikel
    for j = 1 : N
        curfeval(j) = feval(func, p(j,:));
        if curfeval(j) < pbestfeval(j)
            pbest(j,:) = p(j,:);
            pbestfeval(j) = curfeval(j);
        end
    end

    % Uppdaterar globalt bästa partikel
    for j = 1 : N
        if pbestfeval(gbest) > curfeval(j)
            gbest = j;
        end
    end

    % Förflytta partiklarna genom tillståndsrummet
```

```
for j = 1 : N

    % Beräkna ny hastighet för partikel j
    for k = 1 : D
        if j == focal
            v(j,k) = W_cur * v(j,k) + C1 * rand * (pbest(j,k)-p(j,k))
                     + C2 * rand * (pbest(gbest,k)-p(j,k));
        else
            v(j,k) = W_cur * v(j,k) + C1 * rand * (pbest(j,k)-p(j,k))
                     + C2 * rand * (pbest(focal,k)-p(j,k));
        end
        if v(j,k) > V_MAX
            v(j,k) = V_MAX;
        elseif v(j,k) < -V_MAX
            v(j,k) = -V_MAX;
        end
    end

    % Beräkna nytt tillstånd för partikel j
    for k = 1 : D
        p(j,k) = p(j,k) + v(j,k);
        if p(j,k) > u
            p(j,k) = u;
        elseif p(j,k) < -u
            p(j,k) = -u;
        end
    end
end

% Uppdatera förfluten tid
curtime = cputime - starttime;

% Uppdatera tröghetsvikt
W_cur = W_start - (W_start - W_stop) / (stoptime / curtime);

% Logga algoritmens framgång
if (stoptime / nroflogs) * log < curtime
    data(log + 1) = pbestfeval(gbest);
    time(log + 1) = curtime;
    log = log + 1;
end
```

```
end

assignin('base','data', data);
assignin('base','time', time);
```

A.5 Gaussian PSO

```
% ------------------------------------------------
% GPSO Algoritm
%   Av:    Mathias Johansson, Mikael Lindberg
%   Datum: 2004-10-10
% ------------------------------------------------
% Parameterbeskrivning
%   func     - Funktion att optimera
%   N        - Antal partiklar i svärmen
%   D        - Antal dimensioner av problemet
%   u        - Begränsning av värdena på problemets
%              inparameter, för alla x vilka tillhör
%              {X1...Xn} hör till intervallet [-u, u]
%   stoptime - Antal sekunder algoritmen ska exekvera
%   nroflogs - Antal logg tillfällen exkl startvärde
% ------------------------------------------------

function GPSO(func, N, D, u, stoptime, nroflogs);

% Initiering av startvärden
starttime = cputime;
C1 = 0.4;               % Accelerationskonstant
C2 = 0.6;               % Accelerationskonstant
log = 1;                % Inv: 1 <= log <= nroflogs

% Initierar partiklar och dess hastighet
for i = 1: N
    for j = 1: D
        p(i,j) = rand * u * 2 - u;
    end
end
pbest = p;
gbest = 1;
```

```
for j = 1 : N
    curfeval(j) = feval(func, p(j,:));
    pbestfeval(j) = curfeval(j);
end

% Logga initialvärden
data(log) = pbestfeval(gbest);
time(log) = 0;

% Huvudloop, avbryts efter en förutbestämd tid
while log <= nroflogs

    % Uppdaterar personbästa för varje partikel
    for j = 1 : N
        curfeval(j) = feval(func, p(j,:));
        if curfeval(j) < pbestfeval(j)
            pbest(j,:) = p(j,:);
            pbestfeval(j) = curfeval(j);
        end
    end

    % Uppdaterar globalt bästa partikel
    for j = 1 : N
        if pbestfeval(gbest) > curfeval(j)
            gbest = j;
        end
    end

    % Förflytta partiklarna genom tillståndsrummet
    for j = 1 : N
        temp = rand;
        if j == gbest
            dist = 1;
        else
            dist = sqrt(sum((pbest(j,:)-pbest(gbest,:)).^2));
        end
        if temp > C1
            magn = ((1-C2)*dist) * randn(1);    % lika med GRand()
        else
            magn = (C2*dist) * randn(1);        % lika med GRand()
        end
```

```
        % Beräkna ny hastighet för partikel j
        for k = 1 : D
            v(j,k) = magn * rand * 2 * pi;
        end

% Flytta partikel j
        if temp > C1
            p(j,:) = pbest(j,:) + v(j,:);
        else
            p(j,:) = pbest(gbest,:) + v(j,:);
        end

        % Kontrollera intervall för partikel j
        for k = 1 : D
            if p(j,k) > u
                p(j,k) = u;
            elseif p(j,k) < -u
                p(j,k) = -u;
            end
        end
    end

    % Uppdatera förfluten tid
    curtime = cputime - starttime;

    % Logga algoritmens framgång
    if (stoptime / nroflogs) * log < curtime
        data(log + 1) = pbestfeval(gbest);
        time(log + 1) = curtime;
        log = log + 1;
    end
end

assignin('base','data', data);
assignin('base','time', time);
```