ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

A NOVEL PARTICLE SWARM OPTIMIZATION ALGORITHM

M.Sc. THESIS

Shahriar ASTA

Department of Computer Engineering

Computer Engineering Programme

JANUARY 2012

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

A NOVEL PARTICLE SWARM OPTIMIZATION ALGORITHM

M.Sc. THESIS

Shahriar ASTA
(504091538)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Assoc.Prof. Dr. Şima ETANER UYAR

JANUARY 2012

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

A NOVEL PARTICLE SWARM OPTIMIZATION ALGORITHM

YÜKSEK LİSANS TEZİ

Öğrenci Shahriar ASTA
(504091538)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Şima ETANER UYAR

OCAK 2012

Shahriar ASTA, a **M.Sc.** student of ITU **Institute of / Graduate School of Science and Technology** student ID 504091538, successfully defended the **thesis/dissertation** entitled "**A NOVEL PARTICLE SWARM OPTIMIZATION**", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc.Prof. Dr. Şima Etaner Uyar** İstanbul Technical University	....
Jury Members :	**Assoc.Prof. Dr. Zehra ÇATALTEPE** İstanbul Technical University	....
	Prof. Dr. İbrahim EKSİN İstanbul Technical University	....

Date of Submission : 19 December 2011
Date of Defense : 24 January 2012

FOREWORD

Special thanks to Assoc.Prof.Dr Şima Etaner Uyar for her inspiring suggestions.

December 2011

Shahriar ASTA
(Computer Engineer)

TABLE OF CONTENTS

ABBREVIATIONS

PSO	**:** Particle Swarm Optimization
GA	**:** Genetic Algorithms
CMA-ES	**:** Covariance Matrix Adaptation Evolutionary Strategy
CBCW-PSO	**:** Closest Best , Closest Worst Particle Swarm Optimization
CLPSO	**:** Comprehensive Learning Particle Swarm Optimization

LIST OF TABLES

LIST OF FIGURES

A NOVEL PARTICLE SWARM OPTIMIZATION ALGORITHM

SUMMARY

Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995), which is a population-based global search method is known to suffer from premature convergence prior to discovering the true global minimizer. In this thesis, a novel memory-based method is proposed which aims to guide the particles through the information deduced from the external memory contents rather than to re-inject them into the population.

This is done by calculating a coefficient, based on the distance of the current particle to the closest best and closest worst particles in the external memory at each iteration. Later, when updating the velocity component, this coefficient is added to the current velocity of the particle with a certain probability.

Also randomized upper bound and lower bound values have been defined for the inertia component. The algorithm starts with the upper bound value of the inertia. At each particle evaluation the inertia is decreased non-linearly with a small value and when its value reaches the lower bound, the inertia value is reset to its upper bound.

The resulting PSO finds the global optima much faster than the original PSO and it have been shown that it also performs better compared with a recent improvement of PSO, CLSPO namely. A state-of-the-art algorithm, CMA-ES (Covariance Matrix Adaptation Evolutionary Strategy), has also been chosen for comparison purposes. It has been shown by experiments that although the CMA-ES shows a better performance than that of our algorithm, in some cases where the overall topology pointing to the global optimum is missing and the attractor volume of global optimum is small, our algorithm performs better and finds the desired optimum value of the function in lesser evaluation counts. The tests have been consucted on standard benchmark functions as well as a simulation of the Aldebaran NAO robot for developing a kick action.

YENİ BİR PARÇACIK SÜRÜ OPTİMİZASYON ALGORİTMASI

ÖZET

Parçacık Sürü Optimizasyonu (Particle Swarm Optimization) (PSO) 1995'te Dr. Eberhart ve Dr. Kennedy tarafından geliştirilmiş popülasyon temelli sezgisel bir optimizasyon tekniğidir. Bu teknik, kuş sürülerinin davranıslarından esinlenilerek geliştirilmiş popülasyon tabanlı stokastik optimizasyon tekniğidir. Doğrusal olmayan problemlerin çözümü için tasarlanmıştır. Çok parametreli ve çok değişkenli optimizasyon problemlerine çözüm bulmak için kullanılmaktadır. PSO, genetik algoritmalar gibi evrimsel hesaplama teknikleriyle bir çok benzerlik gösterir. Sistem rastgele çözümler içeren bir popülasyonla başlatılır ve nesilleri güncelleyerek en optimum çözümü araştırır. PSO da parçacık olarak adlandırılan olası muhtemel çözümler, o andaki optimum parçacığı izleyerek problem uzayında dolaşırlar.

PSO'nun klasik optimizasyon tekniklerinden en önemli farklılığı türev bilgisine ihtiyaç duymamasıdır. PSO'yu uygulamak, algoritmasında ayarlanması gereken parametre sayısının az olması sebebiyle oldukça basittir. PSO, fonksiyon optimizasyonu, bulanık sistem kontrolü, yapay sinir ağı eğitimi gibi bir çok alanda başarıyla uygulanabilmektedir.

Bu tekniğin bilinen dezavantajı, gerçek optimumu bulmadan önce erken yakınsama sergilenmesidir. Bu çalışmada, literatürde geleneksel hale gelen ve yeni parçacıkları popülasyona yeniden enjekte etmeyi öneren yöntemlere karşın, parçacıkları harici belleklerden alınan bilgilere göre yönlendiren yeni bir parçacık sürü optimizasyon algoritmasını sunmaktadır. Bunun için, algoritmada iki ayrı harici bellek ön görülmuştur. Bu belleklerden biri en kötü parçacıkları barındırmakta, diğeri ise en iyi parçacıkları barındırmaktadır. Bu belleklerin boytuları birbirinden bağımsız olmaktadır ve belleklerin içeriği her nesilde güncelleştirilmektedir.

Önerilen algoritma başlamadan önce, belleklerin içi doldurulmaktadır. İlk değerlerle dolduktan sonra, her nesilde tüm parçacıklar değerlendiriliyor ve uygunluk değerleri hesaplanıyor. Daha sonraki adımda, parçacığın harici bellekteki parçacıklar arasından, parçacığa en yakın olan en iyi ve en kötü parçacıkların uzaklığı hesaplanıp bir katsayi üretilmektedir. Bu katsayı adım boyutunu belirlemektedir. Sonra her parçacık için hız bileşenini hesaplarken bu katsayı belli bir olasılıkla mevcut hıza eklenmektedir. İlk 1998'de önerilen atalet bileşeni, önerdiğimiz algoritmada periyodik biçimde yeniden başlatılmaktadır. Bu, literatürde ilk kez önerilmektedir ve hesapladığımız katsyı ile birleştirildiği zaman, yakınsama hızını doğrudan etkilemektedir.

Ayrıca randomize bir üst ve alt sınır atalet bileşeni için tanınmaktadır. Algoritmada atalet bileşeni üst sınırdan başlayıp ve her parçacığı değerlendirdikten sonra küçük bir değerle non-linear bir şekilde azaltılmaktadır. Atalet bileşeni randomize alt sınıra ulaştığı zaman, bu bileşenin değeri randomize üst sınırın değeriyle sıfırlanmaktadır. Böylece parçacıklar, onlara en yakın olan en kötü parçacıktan uzaklaştırılmaktadır.
Çıkan PSO evrensel optima'yi orjinal PSO'den daha hızlı bulmaktadır. Bu çalışma önerilen algoritmanın en güncel PSO'ların arasında olan CLPSO algoritmasından daha hızlı olduğunu ve daha kaliteli çözümler ürettiğini ortaya çıkarılmıştır.

CLPSO Algoritması, Karınca Koloni Optimizasyon ve Parçacık Sürü Optimizasyon yöntemlerini birleştirilmektedir. Bu yöntemde, ilk önce orijinal PSO kaç nesil uygulamkata ve kullandığı harici bellekler doldurulmaktadır. Bundan sonraki adımda, algoritma parçacıklardan bir koloni oluşturarak, koloniden çıkan en iyi parçacığı popülasyona enjekte etmektedir. CLPSO ve önerdiğimiz yöntem arasında farklardan biri, CLPSO'nun harici belleği doldurmak için seçici davranmasıdır. Bu seçici davranış nedeniyle, CLPSO'deki orijinal PSO'den CLPSO algoritmasına geçiş uzayabilir. Ayrıca bu geçişin ne kadar süreceği tamamen fonksyona bağlı olduğu gözlemlenmiştir. Başka bir fark ise, CLPSO'nun dahili bellek kullanmasıdır. Dahili bellekler parçacığın kişisel deneyimlerini tutmaktadır. CLPSO'de dahilli bellekler randomize olmak üzere dolduruluyor ve bu de algoritmanın bazen bulduğu iyi çözümden geri dönmesini sağlamaktadır. Bu nedenlerden dolayı, CLPSO, önerdiğimiz yöntemden daha yavaş yakınsamakta ve bulduğu çözümlerin kalitesi daha düşük olmaktadır.

Literatürde en iyi ve güncel optimizasyon algoritmaların arasında olan CMA-ES algoritması de kıyaslamak amacıyla seçilmiştir. CMA-ES algoritmsaı problemdeki parametrelerinin normal dağılımını örneklemekte ve kovaryans matrisini hesaplamaktadır. Bu kovaryans matrisi vesilesiyle, algoritma yeni nesiller üretmektedir. CMA-ES, gerek yakınsama hızı, gerek bulduğu çözümün kalitesi açısından bilinen en iyi algoritmalardan biridir.

Deneyler için standart fonksyonler düşünülmüştür. Bu fonksyonler değişik özelliklere sahipler. Deneme fonksyonlar kümesi, bazi basit fonksyonlar yanısıra, arama uzayında plato oluşturan veya multi-modal ve multi-funnel özelliklere sahip olan fonksyonlardan oluşmaktadır. Multi-modal özelliğe sahip olan bir fonksyon, evrensel optimanın yanı sıra, çeşitli değerlerle bölgesel optimumler barındırmaktadır. Multi-funnel fonksyonlar ise, multi-modal olmakla beraber, iki (veya daha fazla) bölgede farklı bacalarda farklı bölgesel optimumlerı barındırmaktadır. Bir bölgede arama yapan bir parçacığın diğer bölgelere geçme olasılığı bu fonksyonlarda oldukca düşükdur. Multi-modal ve multi-funnel fonksyonlarda, evrensel optima'yi bulmak ve tüm parçacıkların bu optimaya yakınasamasını sağlamak zor bir problem olarak kabul ediliyor.

Deneylerin sonucunda, CMA-ES'in genelde PSO'den daha iyi olmasına rağmen, bazi durumlarda, sunulan yöntemin daha üstün bir performans sergilediği ortaya çıkarılmıştır. Evrensel optima'yı gösteren evrensel bir topolojinin eksik olduğu veya fonksyonun havzasının çeker hacmı küçük olan problemlerde, sunulan algoritma daha hızlı davrandığı gösterilmiştir.

Ayrıca, önerilen yöntem, kıyaslanan yöntemlerle birlikte, Aldebaran üretimi olan NAO insansıl robotu üzerinde denetilmektedir. NAO robotu RoboCup yarışmalarında, 3D benzetim ligin standart robotudur. Bu ligde Simspark yazılımı benzetim ortamı olarak kullanılmaktadır. Benzetim ortamı gerçkcil olmakla beraber, gerçek dünya koşullarına uyum sağlamak adına gürültü içermektedir. Benzetim ortamında öngörülen gürültülerin tipi normal ve tekdüzedir.

Robot deneylerinde, robot için daha önce tasarlanan ve düşük kalitesi olan bir top atışın kalitesinin artması amaçlanıyor. Tasarlanan top atışının manzili 3 metredir ve atışıtan sonra top robotun gövdesinin x aksanından sapmaktadır. Ayrıca top atışından sonra robot düşmektedir. Top atışını iyileştirmek için, robotun kinematic denkelmleri kullanılmaktadır. Bu denkelmler PFS modeli ile osilatör oluşturup harmonik bir şeklide robotu hareketini sağlamaktadır. Bu denkelmler, toplam olarak 40 parametre içeren bir fonksyon oluşturmaktadır. Ayrıca topun atıştan sonraki robot'dan uzaklığı ve sapma değerinden oluşan, uygunluk değeri hesaplayan bir fonksyon tasarlanmıştır. Robot'un atıştan sonraki düşmesi durumunda, bu uygunluk değerine ceza uygulanarak, arama sırasında iyi atışlara neden olan ama robotun düşmesine sebebiyet veren bireyler çözüm olarak sunulmamaktadır.

Deneylerin sonucunda, algoritmamız diğer algoritmalardan (CLPSO ve CMA-ES) daha üstün bir başarı sergilemektedir. Top atışının menzili 6 metre olmakla beraber, robot düşmemekte ve top gövdeden sapmamaktadır.

1. INTRODUCTION

The origins of Particle Swarm Optimization goes back to 1987 where Reynolds introduced the simulation of particles similar to that of bird flocks. In this simulation each particle or agent tries to keep a distance from too-close neighbors and steers towards the average heading of the flock while maintains it's position within the flock. Later Kennedy and Eberhart [1] introduced few parameters to this simulation and came up with the idea of Particle Swarm Optimization. In their work, each particle remembered it's best position regarding to the global optima and shared this information with others about that best position. **Figure 1.1** shows the evolution of particles in a simple, 2D search space.

In technical words, PSO, is a population-based stochastic search in which each particle represents a potential candidate solution to the problem. Elements of each particle are in fact the parameters of the problem. This technique shares many similarities with other evolutionary computing techniques like Genetic Algorithms (GA). First the population of particles are initialized by random parameter values and these parameter values are updated in each iteration (generation), until a stop criteria has been met or the algorithm is converged to some optima. However, this technique does not include special operators like mutation or crossover. In PSO the particles"fly" over the search space by following the current optimum particles. The ultimate purpose of any optimization algorithm is to estimate the inverse of the Hessian matrix (approximation of the second derivative of the function) of the function and calculate a function's contour map to find the global optima. However methods like PSO (population-based methods in general) as well as our method are independent from the gradient of the function and use "intuition" and social behavior of the individuals to achieve the global optimum of the function.

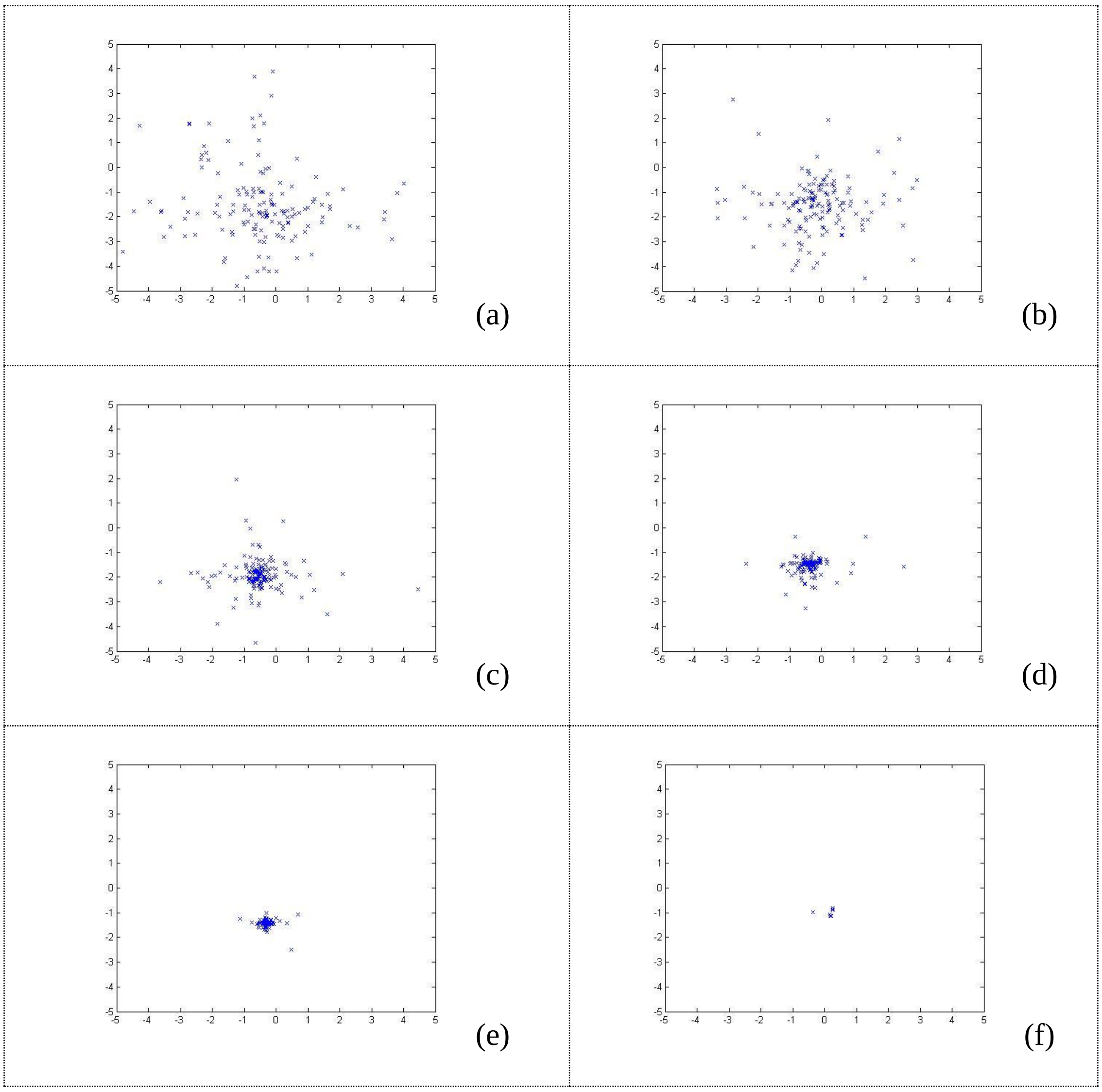


Figure 1.1 : The evolution of particles through search space

1.1 Research Motivations

Based on the observations achieved from the application of evolutionary algorithms on robot motion, one can say that sometimes, while producing new individuals, which represent a solution for the problem under optimization, a certain portion of the individual is close to that of the optimal solution. Only a few parameters inside the current individual are diverse and far from being a part of the optimum solution.

Although the number of these parameters may be small, the effect of it may turn the performance of the individual close to that of the worst case.Considering a landscape in which we have many minima and maxima points, one can also say that, while updating the position of the individual, keeping it far away from the maxima (in a minimization problem) will increase the speed of the convergence to the optima. However, this alone might not help since there are problems for which both minima and maxima are in close neighborhood or when there are many local minima. Then we need to equip the optimization algorithm with the ability to escape from the maxima and move towards the minima even when the minima are close to both the current position of the particle and the maxima.Some algorithms consider a combination of multiple algorithms to achieve this goal. Re-injecting the best solution found so far, to the population of solutions or applying genetical operators such as mutation or crossovers (partially exchanging the variabla vector of the current solution with the variable vector of the best-known solution) are few examples. In this work however, we consider to deduce information from the best available solutions in order to guide particles through the search space. The expected result is to reduce the number of iterations and function evaluations and find a better solution in a reasonably shorter time.

1.2 Contributions

A novel algorithm along with its parameter setting scheme in which finds the global minimum of a function in reasonable evaluation counts (less than that of the original PSO and the latest improvement on PSO) is presented in this thesis. The resulting algorithms converges faster and is a good means for application which have time constraints.

2. PARTICLE SWARM OPTIMIZATION

2.1 Problem Formulation

The goal of any optimization problem is to maximize or minimize an objective function $f\ (\vec{x})$ where $\vec{x}$ is the decision vector consisting of n dimensions or decision variables consisting of real numbers. Since maximization of any function $f\ (\vec{x})$ is equivalent to minimization of $-f\ (\vec{x})$, the literature generally focuses on minimization without loss of generality. Solution $\overrightarrow{x^*}$ is a global minimizer of $f\ (\vec{x})$ if and only if $f\ \left(\overrightarrow{x^*}\right) \leq f\ (\vec{x})$ for all x in the domain of $f\ (\vec{x})$. The unconstrained minimization problem of consideration here can be formulated as.

$$\text{Minimize } f\ (\vec{x})$$
$$\text{where : } f\ :\ R^n \rightarrow R$$

2.2 Evolution of The Particle Swarm Optimization Algorithm

2.2.1 The original particle swarm optimization

Particle Swarm Optimization (PSO) is a nature inspired meta-heuristic method. This method was first introduced by Kennedy and Eberhart in 1995 [1]. It is inspired by the swarm behavior of birds flocking, and utilizes this behavior to guide the particles to search for globally optimal solutions. Basically, in PSO, a population of particles is spread randomly throughout the search space. The particles are assumed to be flying in the search space. The velocity and position of each particle is updated iteratively based on personal and social experiences. Each particle possesses a local memory in which the best so far achieved experience is stored. Also a global memory keeps the best solution found so far. The sizes of both memories are restricted to one. The local memory represents the personal experience of the particle and the global memory represents the social experience of the swarm. The balance between the effect of the personal and social experiences are maintained using randomized correction coefficients. The philosophy behind the update procedure is to reduce the distance between the particle and the best personal and social known locations. PSO is very easy to implement and there have been many successful implementations based on PSO in several real world applications. PSO is a population based approach in which finding the optimal solution is not guaranteed. Also, it can get stuck in local optima when dealing with complex multi modal functions. This is why accelerating the convergence speed as well as avoiding the local optima problem are two primary goals in PSO research. Multiple methods and approaches have been suggested to improve the performance of the original PSO in terms of these goals. In [2], these efforts have been divided into four categories. The first category includes the parameter selection methods. Introducing the inertia and constriction factors into the basic velocity expression or developing strategies for time independent variation of algorithm parameters are among the many methodologies in this category. The method presented in this paper fits best within this category. Other categories pointed in [2] are Applications of PSO into different problems areas (second category), Generation of different algorithm strategies and analysis of convergence (third category) and finally Hybridization (Fourth category). Although, the novel approach presented in this paper focuses on parameter selection, it also tries to generate a different strategy and attempts to reduce the iteration count. This is why, this work can also be included in the third category. In the basic PSO, each particle is considered as a potential solution to the numerical optimization problem in a D dimensional space. Every particle has a position in this search space and a velocity assigned to it. The position of the particle is represented by $X_i = x_{i1}, x_{i2}, \dots, x_{iD}$. The velocity of a particle is given as $V_i = v_{i1}, v_{i2}, \dots, v_{iD}$. Also, each particle has a local memory (*pBest*) which keeps the best position that is experienced by the particle so far. A globally shared memory (*gBest*) keeps the best global position found so far. These information contribute to the flying velocity of each particle, using the following equation :

$$v_i = v_i + \varphi_1 \times rand \times (pBest_i - x_i) + \varphi_2 \times rand \times (gBest - x_i) \quad \textbf{(2.1)}$$

The position update is then as:

$$x_i = x_i + v_i \tag{2.2}$$

where, $\varphi_1$and φ_2 are positive constants determining the relative influence of the personal and social experiences during the search. Defining an upper bound for the velocity component increases the performance of the approach

2.2.2 Inertia

In [3], it has been shown that the introduction of an inertia factor to the Eq.(**2.1**) improves performance, since it adjusts the velocity over time and improves the search precision of the particles. Eq.(**2.1**) can be rewritten as:

$$v_i = \omega \times v_i + \varphi_1 \times rand \times (pBest_i - x_i) + \varphi_2 \times rand \times (gBest - x_i) \tag{2.3}$$

where ω is the inertia factor. The inertia weight ω is employed to control the impact of the previous history of velocities on the current velocity, thus to influence the trade-off between global (wide-ranging) and local (nearby) exploration abilities of the "flying points". A larger inertia weight ω facilitates global exploration (searching new areas) while a smaller inertia weight tends to facilitate local exploration to fine-tune the current search area. Suitable selection of the inertia weight ω can provide a balance between global and local exploration abilities and thus require less iterations on average to find the optimum. In this thesis a new inertia scheme is introduced for the first time in which the inertia is considered to be changing linearly between a lower and upper bound (starting from upper bound) and it is being periodically restarted to it's upper bound whenever it reaches to the value of of the lower bound. It has been shown that restarting has a tremendous effect on the performance of the algorithm.

2.2.3 Constriction factor

Clerc[4] introduced a constriction factor K for more efficient control and constraining of velocities. Then Eq.(**2.1**) was modified as:

$$v_i = K \times \left(v_i + \varphi_1 \times rand \times (pBest_i - x_i) + \varphi_2 \times rand \times (gBest - x_i)\right) \tag{2.4}$$

where *K* can be expressed as:

$$K = \frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|} \quad \textbf{(2.5)}$$

here, $\varphi = \varphi_1 + \varphi_2, \varphi > 4$. In [5] Eberhart concludes that constriction factors has a far better impact on convergence and it's speed.

2.2.4 Particle velocity

According to [2] success of inertia and constriction factor equations are problem dependent. This is why, in this thesis both equations have been used and the best out coming result has been presented. This is to say that algorithms with which the functions are optimized with and compared are tested in a limited and small number of times using each of the equations (**2.4**) and (**2.3**) for each function under optimization and the best responding equation is chosen for that. This is equivalent to chosing the best step size and velocity for particles for a specific problem.

3. MEMORY BASED PARTICLE SWARM OPTIMIZATION

3.1 Local Optima and Convergence Speed in Standard Particle Swarm Optimization

The convergence speed of PSO in it’s original form is fast, however, the main problem of the PSO is that it sometimes gets trapped in local optima and the convergence speed rate decreases considerably in later period of evolution. When the algorithm reaches near global optima the algorithm stops optimizing and thus the accuracy the algorithm can achieve is limited. Many approaches have been introduced to overcome this problem. Among them, linearly decreasing the inertia weight, randomizing the inertia weight, utilizing external memory and etc. Since external memory is employed in this thesis we will focus on the methods where this approach has been applied.

3.2 Utilizing Internal and External Memory

Memory based PSO proposals are mainly concentrated on various methods, by which the local and global best positions are selected and used. The work of CoelloCoello et al. [5] is one of the first in this respect. In this work, the global best is determined by selection of a non-dominant solution from the external memory. Local best is updated with respect to the Pareto dominance. Hu et al. [6] extended their work using dynamic neighborhoods and employed an external memory to memorize all potential Pareto optimal solutions. In their work the global best is selected among the candidates within the external memory by defining a neighborhood objective and an optimization objective. The global best is found among the particles neighbors, subject to the defined neighborhood objective.One other attempt to employ additional memory in PSO is the work of Wang [7]. In this method, The Triggered Memory-Based PSO, they try to adjust the parameters of the PSO in dynamic environments where the characteristics of the search space change over time. However, this method is novel and successful in terms of presenting the effects of utilizing additional memory into PSO. In their work a certain, predefined number of globally best positions, are kept and re-injected into the search population when necessary. This method is particularly successful when the location of the optima change over time. One other successful work which utilizes external memory is the work of Acan [8]. In their work a single globally best position is kept along with a certain number of globally worst positions. A crossover operator is used to replace a randomly chosen set of particles after each iteration. Yet in another work, Acan [2] introduced a hybrid method where there is a global memory and also a local memory for each particle. A colony, consisting of the local and global positions is then constructed and at each evaluation, members of these colonies are used to update the velocity and position of the particle in process. Then the new positions are evaluated and the best outcome, replaces the particle's current velocity. There are many approaches, both employing additional memory and/or hybridization or other techniques, for which additional information can be found in [2,9]. However, the main idea in almost all of these memory utilizing approaches is to re-inject the globally best position into the population during the search. In our study, however, the main idea is to deduce information from the contents of the external memory in order to affect the velocity of the particles towards the global optima.

4. THE PROPOSED ALGORITHM

4.1 Motivation For A New Memory Based Particle Swarm Optimization

The resulting algorithm is supposed to find the optimum parameters of a kick action for a soccer playing humanoid robot in simulation environment. Although the tests are conducted on simulator, and since each experiment takes approximately 11 seconds to complete, there is a need for a faster algorithm which finds the optimum values of robot’s joint oscillators for the kick action in less evaluation counts. This is why in this thesis we focus on finding an algorithm which is optimizing the target function in a reasonably shorter time. As shown in the test results, even the most recent version of memory utilizing PSO is slower than the algorithm presented in this thesis. Also comparative tests has been conducted on one the best known algorithms and it is shown by experiments that our algorithm performs better than the forth mentioned algorithm (CMA-ES) under certain circumstances. These circumstances include noisy test environments which are not rare and one can assume it as the normal nature of any robot experiment.

4.2 The Design of The Proposed Algorithm

In this thesis we propose an algorithm, with which the particles, are guided far away from the closest worst location by correcting their position to the location at which the particle is supposed to be, prior to updating its position. In order to do this, we construct two lists of the so far found global best and global worst positions. Each of the two lists are in fact, two separate external memories of the PSO. Before updating the velocity of the particles, we scan the external memory which keeps the best positions and determine the best location inside the external memory which is closest to the particle. Let's call this closest best position CB. Again we scan the external memory which keeps the global worst positions and in a similar way, we find the global worst position which is closest to the particle. We call this position CW. In our experiments Euclidean distances are used. However, any other distance measure may also be used for this purpose. After choosing the closest best and worst elements of the two external memories with respect to the particle, we measure the similarity of the particle with each of these elements, using the two parameters CB and CW:

$$c_1 = \frac{\sqrt{\sum_{i=1}^{D}(CB-x_i)^2}}{(x_u-x_l)} \tag{4.1}$$

$$c_2 = \frac{\sqrt{\sum_{i=1}^{D}(CW-x_i)^2}}{(x_u-x_l)} \tag{4.2}$$

Here x_u and x_l are the upper and lower bounds for the values that the particle can take in each dimension. In our experiments this value range has been considered to be equal for all dimensions. Equation **(4.1)** measures the amount of similarity between the closest best and the particle. Equation **(4.2)** measures the similarity between the closest worst and the particle. The position correction coefficient can be defined as in follow:

$$C = \frac{|c_1-c_2|}{(c_1+c_2)} \tag{4.3}$$

Here $-1 \leq C \leq 1$. We now are able to present our algorithm as follows. First, as mentioned earlier, we define two external memories: one contains a certain fixed number of global best positions and the other one contains a certain fixed number of global worst positions. The sizes of these two external memories are not necessarily equal. Before utilizing these memories in our approach, we have to initialize them properly. In order to do this, we run the basic PSO until the iteration count is equal to the maximum of the two external memory sizes. At each iteration, we insert the global best and worst positions into the corresponding external memories. This way, the external memories are initialized. After initialization, we now are able to calculate the C coefficient in equation **(4.3)**. In other words, after each evaluation of all the particles, we refresh the external memory contents, and based on the information available in the external memories we calculate the necessary coefficient. However, there is a special method with which we use this coefficient. According to a particular probability, we either use the basic equation or the following equations which are the modified forms of equations **(2.3)** and **(2.4)** respectively:

$$v_i = \omega.(v_i + C) + \varphi_1.rand.(pBest_i - x_i) + \varphi_2.rand.(gBest - x_i) \qquad \textbf{(4.4)}$$

$$v_i = K.\left((v_i + C) + \varphi_1.rand.(pBest_i - x_i) + \varphi_2.rand.(gBest - x_i)\right) \qquad \textbf{(4.5)}$$

In equations **(4.4)** and **(4.5)**, we add the current velocity of the particle with the coefficient, in order to move the particle away from the worst position towards the global best. When we use equation **(4.4)** for velocity update, based on observations, we can say that decreasing the inertia factor linearly, will decrease the effect of the coefficient and yield early convergence. Decreasing the inertia factor means that during this period we increase the importance of the social and cognitive factors, and pay less attention to the actual velocity of the particle. In cases where the particle is stuck in local optima, using a linearly decreasing inertia leaves the particle with a velocity less than the amount that the particle requires to escape those optima. This is why we have tried to decrease the inertia value linearly until a certain threshold is reached. At such a point, the inertia is reset to its initial value. Here we have defined a lower bound and an initial value for the inertia. In order to achieve diversification, some randomness is added in the upper bound, lower bound and decreasing factor. Since this algorithm is based on the distance of the particle to its globally closest best and globally closest worst, we refer to it as CBCW PSO from here on. **Figure 4.1** shows the pseudo-code of the algorithm.

As mentioned earlier this algorithm has the effect that it guides the particles away from the closest worst towards the direction of the optima. This is due to the fact that without applying the *C* coefficient in **equations (4.4)** and **(4.5)**, the particle heads in a direction which is a direction between the direction of local best and that of the global best (depending on the value of φ_1 and φ_2 variables). *C* coefficient is realtive to the distance difference between the closest best and worst and the closest best is not any worse than particle’s local best. So, without applying the coefficient the particle already moves away from the closest worst. Applying the coefficient will add to the speed without changing the direction of the particle. Thus when particle is closer to a worst position than it's best position, the *C* coefficient will be higher and the speed of the particle towards the local and global best will be higher. This is illustrated in **Figure 4.2.**

```
Initialize the particles uniformly in acceptable range and velocities to zero
Define external memory for best positions (bests) . Initialize it to zeros
Define external memory for worst positions (worsts). Initialize it to zeros
while a certain number of iterations is reached or converged do
    for each particle $x_i$ do
    $x_i = x_i + v_i$
    if particle position at each dimension exceeds its acceptable range
        $x_i$= random from range
    val = evaluate particle
    update the local best
    replace the worst in bests array with gbest
    replace the best in worsts array with gworst
    if iteration count > max(size(bests), size(worsts))
        for each particle $x_i$
            CB = closest best to particle
            CW = closest worst to particle
            Calculate $c_1$ and $c_2$ according to equation (4.1) and (4.2)
            calculateC according to equation (4.3)
            for each dimension
                with probability p
                update the position like the basic PSO in equation (2.3) or (2.4)
                otherwise
                update the position like equation (4.4) or (4.5)
        if $\theta \geq lower\ bound + \varepsilon$
            $\theta = \theta - \Delta\theta - \varepsilon$
        else

            $\theta = upper\ bound - \varepsilon$

end
```

Figure 4.1 : Pseudo-code of CBCW-PSO algorithm

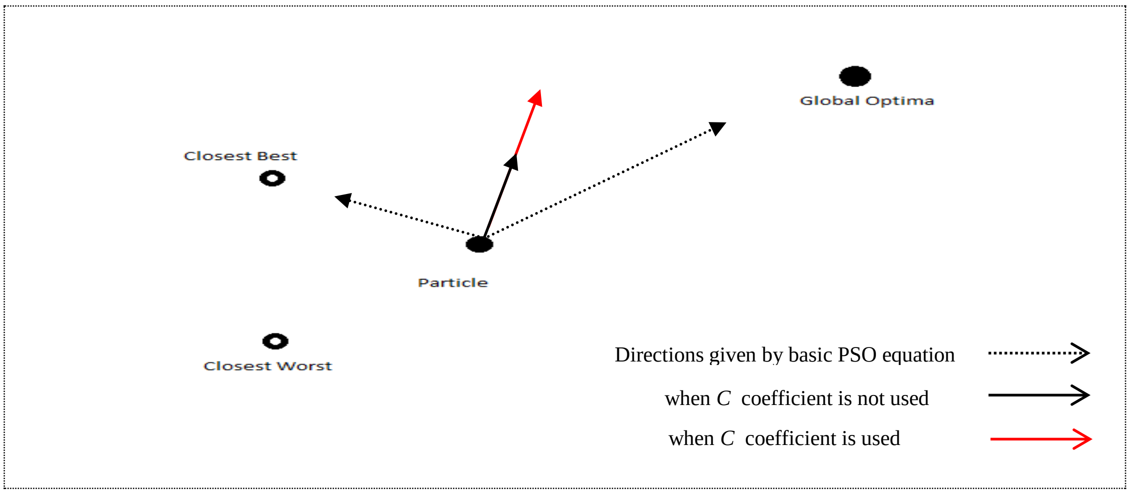

Global Optima
Closest Best
Particle
Closest Worst
Directions given by basic PSO equation
when C coefficient is not used
when C coefficient is used

Figure 4.2 : The effect of the **C** coefficient in equations **(4.4)** and **(4.5)**

5. PROBLEMS AND COMPARED ALGORITHMS

Tests are performed in two platforms. In one platform, the new algorithm as well as CLPSO and CMA-ES algorithms are tested and evaluated on standard benchmark functions. The global minimum of these functions are known and properties of each function is described in [7]. Knowing these properties helps us to understand the weaknesses and strengths of our algorithm comparing with other algorithms. However, the ultimate purpose is to apply our new algorithm on a Humanoid Robot (Aldebaran NAO) and develop a Kick action for a soccer-playing robot. Also in this platform, all the three algorithms are employed for optimizing an available kick action and the results are compared with each other. Thus, we have two platforms for our tests. The former is the standard benchmark functions platform and the latter is humanoid robot platform. The subsequent sections describe each platform in detail.

5.1 Optimization Problems

5.1.1 Benchmark optimization functions

Two rounds of experiments has been conducted on benchmark functions. In the first round, as will be described later, only CBCW-PSO and CLPSO are being compared to one another. The purpose is to understand the basic behaviour of the proposed algorithm when a wide range of functions (simple, uni-modal, multi-modal and hard-to-solve functions) is considered. In the second round of experiments more attention is paid to more complicated functions where the benchmark functions are chosen to be hybrid, highly multi-modal and/or multi-funnel. All the benchmark problems are adopted from [6], [7] and [8]. The functions used in the first round of experiments are shown in Table 5.1 and the results are discussed further in [13]. The majority of these functions are adopted from 2005 IEEE Congress on Evolutionary Computing (CEC 2005). The benchmark functions provided by the conference are standard functions and algorithms are being tested and compared to each other, using these benchmark functions.

Table 5.1 : Benchmark functions for preliminary experiments

Benchmark Problems	Min	Range
$f_1(\vec{x}) = \sum_{i=1}^{D} x_i^2$	0	$-5.12 \le x_i \le 5.12$
$f_2(\vec{x}) = \sum_{i=1}^{D} \sum_{j=1}^{i} x_i^2$	0	$-65 \le x_i \le 65$
$f_3(\vec{x}) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} x_i$	0	$-10 \le x_i \le 10$
$f_4(\vec{x}) = \sum_{i=1}^{D} \left(\left\lfloor x_i + \frac{1}{2} \right\rfloor \right)$	0	$-100 \le x_i \le 100$
$f_5(\vec{x}) = \sum_{i=1}^{D} \lvert x_i^{i+1} \rvert$	0	$-1 \le x_i \le 1$
$f_6(\vec{x}) = \left(\sum_{i=1}^{D} (i+1) x_i^4 \right) + rand[0,1[$	0	$-5.12 \le x_i \le 5.12$
$f_7(\vec{x}) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} x_i^2$	0	$-100 \le x_i \le 100$
$f_8(\vec{x}) = \sum_{i=1}^{D} (100 \times (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	0	$-30 \le x_i \le 30$
$f_9(\vec{x}) = \sum_{i=1}^{D} -x_i \times \sin(\sqrt{\lvert x_i \rvert})$	-12569.5	$-500 \le x_i \le 500$
$f_{10}(\vec{x}) = \sum_{i=1}^{D} (x_i^2 - 10 \times \cos(2\pi x_i) + 10)$	0	$-5.12 \le x_i \le 5.12$
$f_{11}(\vec{x}) = \frac{1}{4000} \left(\sum_{i=1}^{D} x_i^2 \right) + \left(\prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i+1}} \right) \right) + 1$	0	$-600 \le x_i \le 600$
$f_{12}(\vec{x}) = -20 \times \exp\left(-0.2 \times \sqrt{\frac{1}{n} \sum_{i=1}^{D} x_i^2} \right) - \exp\left(\frac{1}{n} cos(2\pi x_i) \right) + 20 + e$	0	$-32 \le x_i \le 32$
$f_{13}(\vec{x}) = \sum_{i=1}^{D} \left(\sum_{k=0}^{Kmax} a^k \cos\left(2\pi b^k (x_i + 0.5) \right) \right) - D \sum_{k=0}^{Kmax} (a^k \cos(2\pi b^k \times 0.5))$	0	$-0.5 \le x_i \le 0.5$
$f_{14}(\vec{x}) = 4x_0^2 - 2.1x_0^4 + \frac{1}{3} x_0^6 + x_0 x_1 - 4x_1^2 + 4x_1^4$	1.0316	$-5 \le x_i \le 5$
$f_{15}(\vec{x}) = \left(x_1 - \frac{5.1}{4\pi^2} x_0^2 + \frac{5}{\pi} x_0 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) \cos(x_0) + 10$	0.398	$-5 \le x_i \le 15$
$f_{16}(\vec{x}) = \frac{-(1 + \cos(12\sqrt{x_0^2 + x_1^2}))}{\frac{1}{2}(x_0^2 + x_1^2) + 2}$	0	$-5.12 \le x_i \le 5.12$

In the second round of experiments, in addition to more multi-modal and multi-funnel functions, hybrid functions are also employed. The reason is that they inherit various properties with different optimization considerations from their consisting components and expose each algorithm to a hard-to-solve problem. As an example function f_{16} in table 5.2 is consisted of a sphere function with two plateaues, a Weierstrass function which guarantees a huge number of local optima and continuity with the exception of being differentiable only on a set of points, Griewank and Ackley function to make sure that there are local minima at the boundaries and a Rastrigin function to make sure that this already hard-to-solve function shapes the global minimum and that the hybrid function is separable near this global minima.

The list of the chosen test functions and some of their characteristics can be seen in Table5.2.

Function f_7 is the Shifted rotated Griewank function which is non-separable, scalable and multimodal. Function f_8 is the shifted rotated Ackley's function with global minima on the boundary. It is multi-modal and non-separable. Functions f_9 and f_{10} are two different versions of the Rastrigin function in which the former shifted and the latter is both shifted and rotated, both are based on the function of De Jong with the addition of cosine modulation in order to produce frequent local optima. This function is highly multimodal, however the location of local minima are regularly distributed. Function f_{11} is the shifted rotated Weierstrass and as noted before is Continuous but differentiable only at a set of points. This function is both highly multi-modal and multi-funnel. The concept of multi-funnel functions and teir characteristics are covered in section 5.2.2.2. Function f_{13} is consisted of Griewank and Rosenbrock's functions. It is multi-modal, multi-funnel and non-separable. This function is a hybrid function. Function f_{14} is the shifted rotated version of Scaffer's functionand is multi-modal. Finally function f_{16}, which was discussed earlier, is a hybrid multi-funnel and highly multi-modal function.

5.1.2 Humanoid robot , the kick problem

5.1.2.1 Introducing the robot

Aldebaran Nao robot model in Simspark simulation environment has been used for experiments. Simspark is the official simulator for RoboCup competitions and uses ODE (Open Dynamics Engine) for physics simulation [9]. Physical rules along with physical interactions with the environmen (collision, friction, gravity and etc.) have been modeled in the simulator. The main difference with the real world is the fact that robot's servo motors are somehow ideal comparing with the real ones. This only affects the motion speed and the parameters found by this (or any) optimization method could be used in the real world.

Table 5.2 : Benchmark Functions for the second round of experiments.

Benchmark Problems	Biased Min	Range
$Sphere\ Function = \sum_{i=1}^{D} x_i^2$	0	$-5.12 \leq x_i \leq 5.12$
$Rosenbrock's\ function = \sum_{i=1}^{D} (100 \times (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	0	$-30 \leq x_i \leq 30$
$f_7(\vec{x}) = \frac{1}{4000}\left(\sum_{i=1}^{D} x_i^2\right) + \left(\prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i+1}}\right)\right) + 1$	-180	$-600 \leq x_i \leq 600$
$f_8(\vec{x}) = -20 \times \exp\left(-0.2 \times \sqrt{\frac{1}{n}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{n} cos(2\pi x_i)\right) + 20 + e$	-140	$-32 \leq x_i \leq 32$
$f_9(\vec{x}) = \sum_{i=1}^{D} (x_i^2 - 10 \times \cos(2\pi x_i) + 10)$	-330	$-5.12 \leq x_i \leq 5.12$
$f_{10}(\vec{x}) = \sum_{i=1}^{D} (x_i^2 - 10 \times \cos(2\pi x_i) + 10)$	-330	$-5.12 \leq x_i \leq 5.12$
$f_{11}(\vec{x}) = \sum_{i=1}^{D}\left(\sum_{k=0}^{Kmax} a^k \cos(2\pi b^k (x_i + 0.5))\right) - D \sum_{k=0}^{Kmax} (a^k \cos(2\pi b^k \times 0.5))$	90	$-0.5 \leq x_i \leq 0.5$
$f_{13}(\vec{x}) = (Hybridization\ of\ f_{8\,and} f_2\ )$	-130	$-5 \leq x_i \leq 5$
$f_{14}(\vec{x}) = 0.5 + \left(\frac{(sin^2(\sqrt{x^2+y^2}) - 0.5)}{(1 + 0.001(x^2+y^2))^2}\right)$	-300	$-5 \leq x_i \leq 15$
$f_{16}(\vec{x}) = Rotated\ version\ of\ Hybrid\ Composition\ of\ (f_7, f_8, f_9, f_{11}\ and\ Sphere)$	120	$-5.12 \leq x_i \leq 5.12$

The robot's vision system is equipped with a camera installed at the torso of the robot. In the simulation environment, the robot actually does not perform any image processing or 3D vision computing. Instead, the seen objects are reported to the robot from the simulation server. These objects are reported by their polar coordinates . This includes the relative distance, horizontal and vertical angles of the object with respect to the robot. The distance is in meters and the angles are in radians.

According to [9] two types of intentional noise are added to the vision system to keep the environment realistic. There is a static calibration error for the camera position in each axis. This noise is of uniform type and is distributed between -0.005m and 0.005m. This error is calculated once and stays the same for the entire match. A dynamic noise is employed for the percepted objects. This noise has a zero mean Gaussian distribution and is measured as:

- $\sigma = 0.0965 * distance/100$ For Polar Distance

- $\sigma = 0.1225$ For Horizontal Angle
- $\sigma = 0.1480$ For Vertical Angle

As for Nao robot, it has 22 degrees of freedom of which only 14 have been used in the kick motion model. The height of the robot is 57 cm and its weight is 4.3 kg. Since the simulated Nao robot is a realistic representation of the Nao humanoid robot, its joint structure is the same with the real one.

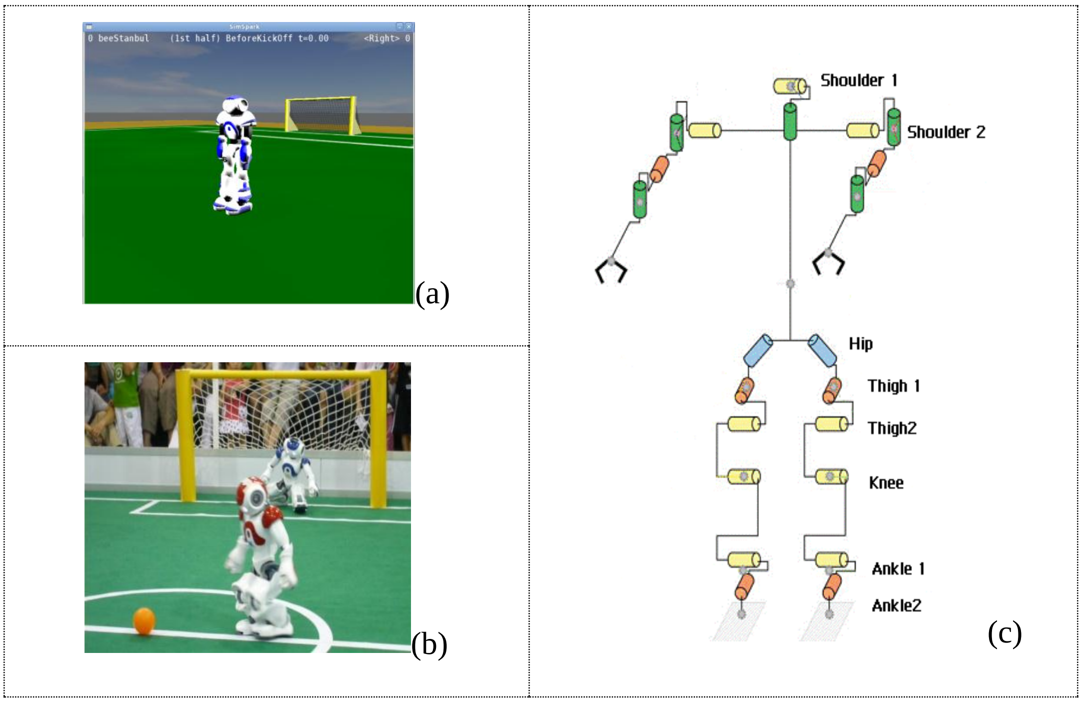


Figure 5.1 : Aldebaran NAO (a)Simulation. (b)Real Robot. (c)Joint Structure

5.1.2.2 Formulating the kick action as an optimization problem

The kick motion can be modeled by a smoothed rolling polygonal shape and a non-periodic function accordingly. A preliminary design has been made for the kick action. The parameters however are not optimum and need to be optimized. The purpose of the kick action is to shoot the ball as far and as straight as possible. The available kick action has a range of 3 meters. In order to analyze the kick behaviour easier, the action has been separated into four phases.

- Expansion Phase: In which the kicking leg is raised above the ground without relocating the Center of Mass (COM). At the same time the support leg is moved such that the torso makes an inclination outwards while the ankle makes an inward inclination in order to compensate the absense of the kicking leg on the ground.
- Preparation Phase: In which the kicking leg is moved backward in order to achieve energy for the kicking phase. The support leg continues the action it performed in the previous phase.
- Execution Phase: In which the kicking leg actually performs the kick and moves towards the ball. In this phase the support leg makes a reverse inclination comapring to the first two phases.
- Wrap-around Phase: In which the legs are returned to their initial position which is the standing position.

The motion of each joint is considered to be pandulum around a certain offset with an amplitude, special to each joint and each phase. This is mathematically presented in the following equation:

$$f(t) = C + \sum_{i=1}^{N} A_i \sin(i\frac{2\pi}{T}t + \emptyset_i) \qquad \textbf{(5.1)}$$

Where C is the offset of the joint, N is the number of frequencies, A_i is the amplitude, T the period, t the time for each phase and finally $\emptyset_i$ is the phase shift. One can think of equation (**5.1**) as a function of multiple frequencies and decompose it using Partial Fourier Transform (PFS) into it's components. Each component represents a joint. An example of a joint's motion equation in a certain moment can then achieved by using the set of equations in **(5.2)** applied to the respective joints. There is an oscillation equation for each joint and it includes the joints in right and left legs. In most of robot motion models however, one leg is considered to be the support leg and the other leg, the swinging leg. In non-periodic motions (including the kick action which is the motion of ineterst in this thesis) the optimum value of the left leg joints and corresponding right leg joints are considered separately. What we are trying to find is the optimum value for $A_{1\ldots12}, C_{1\ldots12}, \emptyset_{1\ldots12}\ and\ T$ in equation set (**5.2**) with which the robot kicks best. It also has to be mentioned thaht only execution pahse has been considered for optimization. The parameters of the oscillators of other kick phases are considered to be fixed.

$$
\begin{array}{ll}
f_{LThigh1}(t) = C_1 + A_1 \sin\left(\frac{2\pi t}{T} + \emptyset_1\right) & f_{RThigh1}(t) = C_2 + A_2 \sin\left(\frac{2\pi t}{T} + \emptyset_2\right) \\
f_{LThigh2}(t) = C_3 + A_3 \sin\left(\frac{2\pi t}{T} + \emptyset_3\right) & f_{RThigh2}(t) = C_4 + A_4 \sin\left(\frac{2\pi t}{T} + \emptyset_4 + \pi\right) \\
f_{LKnee}(t) = C_5 + A_5 \sin\left(\frac{2\pi t}{T} + \emptyset_5\right) & f_{RKnee}(t) = C_6 + A_6 \sin\left(\frac{2\pi t}{T} + \emptyset_6 + \pi\right) \\
f_{LAnkle1}(t) = C_7 + A_7 \sin\left(\frac{2\pi t}{T} + \emptyset_7\right) & f_{RAnkle1}(t) = C_8 + A_8 \sin\left(\frac{2\pi t}{T} + \emptyset_8 + \pi\right) \\
f_{LAnkle2}(t) = C_9 + A_9 \sin\left(\frac{2\pi t}{T} + \emptyset_9\right) & f_{RAnkle2}(t) = C_{10} + A_{10} \sin\left(\frac{2\pi t}{T} + \emptyset_{10}\right) \\
f_{LHip}(t) = C_{11} + A_{11} \sin\left(\frac{2\pi t}{T} + \emptyset_{11}\right) & f_{RHip}(t) = C_{12} + A_{12} \sin\left(\frac{2\pi t}{T} + \emptyset_{12}\right)
\end{array} \tag{5.2}
$$

However there is a parameter which has a global effect on the motion performance. In order to send the new angles calculated in equation (**5.2**) to the servos, the following equation has been used:

$$Speed * \left(\theta_{Target} - \theta_{Current}\right) \tag{5.3}$$

Where θ is the angular value of the joint. Here, the *Speed* value is global in the sense that it is considered to be a constant for all the phases. However the $Thigh_2$, $Ankle_1$ and $knee$ joints of the kicking leg have their own angular speed values. In other words, the *Speed*, together with the angular speed of the the $Thigh_2$, $Ankle_1$ and $knee$ joints of the kicking leg are also subjected to optimization. To achieve the maximum kick range, one has to raise the ball above the ground right before kicking in order to reduce the friction between the ball and the ground. To do this the ankle of the robot has to pitch upwards in a very specific moment in the execution phase (here by ankle we refer to $Ankle_1$ which is a pitch joint). This is why we have considered this specific time as a parameter subjected to optimization. This parameter is symbolized by w . So in general, we want to achieve the optimum value of the following variables in order to perform an optimum kick:

- $A_{1\ldots12}$: The amplitudes of the oscillators of the kicking leg.
- $C_{1\ldots12}$: The offsets of the oscillators of the kicking leg.

- $\emptyset_{1\ldots12}$: The phase shift of the oscillators of the kicking leg.
- T: The motion period.
- Speed: Consists of four angular speed variables. One for the general angular speed of the joints during all the kick phases. And three angular speed variables for $Thigh_2$, $Ankle_1$ and $knee$ joints of the kicking leg at the execution phase.
- w: The time at which the $Ankle_1$of the kicking leg raises in order to lift the ball and reduce the friction in the execution phase.

5.1.2.3 Designing the fitness function

The fitness function is rather simple. What we expect from the kick action is to kick the ball as far as it can and as straight as possible without falling down. So literally the fitness function can be calculated as:

$$f = \frac{10}{\max(D)} + \frac{|\alpha|}{180} \qquad \textbf{(5.4)}$$

Where $\max(D)$ is the maximum ball distance within 4.25 seconds after starting the kick action and α is the angle of the ball with respect to the robot at the end of each evaluation. Each parameter in the above equation is normalized with respect to its maximum achievable value. A penalty value of 0.2 has been considered and added to the fitness value if the robot falls during the evaluation of the individuals.

5.2 Compared Algorithms

5.2.1 Comprehensive learning particle swarm optimization algorithm

5.2.1.1 Comprehensive learning particle swarm optimization, an introduction

CLPSO was introduced by Acan[2]. The algorithm employs externally implemented global (shared) and particle-based (local) memories and a colonization approach similar to artificial immune system algorithms is considered. At any iteration, particle-based memories keep a number of previously best performing personal positions for each particle and the global memory keeps a number of globally best positions found so far. A set of velocities is computed for each particle using each of the personal best positions within its local memory and a number of randomly selected positions from the global memory. This way, a colony of new positions is obtained for each particle and the one with the best fitness is selected and put within the new swarm. Global and local memories are also updated using the solutions within each colony.

5.2.1.2 Advantages and disadvantages

Given enough evaluation counts as the maximum number of evaluation that the algorithm can actually perform, CLPSO is an algorithm with good optimization quality. However according to [13] The algorithm suffers from few drawbacks. Randomness has been considered to fill and update the local memory. As a result, the algorithm sometimes replaces a better solution with a worse and may divert from a globally good solution to a worse solution and hence the best fitness does not show a monotonicaly decreasing descent. This restarting from a worse solution also causes the algorithm to require more iterations comparing with CBCW-PSO. One other problem is that this method is based on reinjection of the best particle among the individuals inside the colony into the population. This reduces the convergence speed since the algorithm needs individuals (in the memories) which enables it to move in the direction of the global minimum. Finding this kind of individuals is time consuming and in fact it is all the optimization task is about. Finally, CLPSO, in it’s initialization phase only performs the original PSO to fill its local and external memories with “promising” individuals [2]. Not only this takes time but also the “prmising” constraint sometimes causes the algorithm to perform many iterations on the function using only original PSO. As a result, when there is a maximum number of evaluations constraint, sometimes the algorithm reaches that maximum evaluation counts even without actually getting out of the initialization phase.

5.2.2 Covariance matrix adaptation evolutionary strategies

5.2.2.1 Covariance matrix adaptation evolutionary strategies, an introduction

CMA-ES stands for Covariance Matrix Adaptation Evolutionary Strategy. The CMA-ES is a stochastic method for real-parameter (continuous domain) optimization of non-linear, non-convex functions. As mentioned in the introduction approximating the second derivative or Hessian information of the function under optimization, one can infer the rate of descent and construct a Taylor polynomial. This polynomial could then be used as a contour map. This is the basic idea behind Newton's method in function optimization where the inverse of the Hessian matrix is the solution of the system. CMA-ES in it's purest form tries to estimate the inverse of the Hessian matrix through covariance matrix adaptation. In the CMA Evolution Strategy, a population of new search points (individuals, offspring) is generated by sampling a multivariate normal distribution. The basic equation for sampling the search points, for generation number $g = 0, 1, 2, \ldots$, reads:

$$x_k^{(g+1)} \sim m^{(g)} + \sigma^{(g)} \mathcal{N}\left(0, C^{(g)}\right) \quad for\ k = 1, \ldots, \lambda \qquad \textbf{(5.5)}$$

$\mathcal{N}\left(0, C^{(g)}\right)$ is a multivariate normal distribution with zero mean and covariance matrix $C^{(g)}$. $x_k^{(g+1)} \in \mathbb{R}^n$ is the *k*-th offspring from generation $g+1$. $m^{(g)} \in \mathbb{R}^n$ is the mean value of the search distribution at generation g. $\sigma^{(g)} \in \mathbb{R}_+$ is the overall standard deviation or step size, at generation g. $C^{(g)} \in \mathbb{R}^{n \times n}$ is the covariance matrix at generation g. $C^{(g)}$ is the covariance matrix of the search distribution. $\lambda \geq 2$ is the population size (sample size or number of offsprings). At each iteration, new offsprings are being produced according to the following parameter update in equations (**5.6**) and (**5.7**):

$$m^{(g+1)} \leftarrow m^{(g)} + \sigma \langle z_{sel} \rangle \ where \quad \langle z_{sel} \rangle = \sum_{i=1}^{\mu} \omega_i z_{i:\lambda} \qquad \textbf{(5.6)}$$

Where $z_i = \mathcal{N}_i\left(0, C^{(g)}\right)$ and $z_{i:\lambda}$ means the first best λ individuals achieved from equation (**5.1**). Also:

$$C^{(g+1)} \leftarrow (1 - c_{cov})C^{(g)} + c_{cov} \frac{1}{\sum_{i=1}^{\lambda} {\omega_i}^2} \langle z_{sel} \rangle \langle z_{sel} \rangle^T \quad \textbf{(5.7)}$$

Here $c_{cov} \approx {}^{2}/_{n^2}$. c_{cov} is the learning rate of the algorithm. Iterating the steps (producing new offsprings using equation (**5.5**) and then updating the mean and covariance matrix according to equations (**5.6**) and (**5.7**)) will result in adaptation of the covariance matrix into the inverse of the Hessian matrix. Various stopping conditions could be considered for the algorithm of which few are Stopping when maximum iteration counts reached or when the algorithm is experiencing stagnation.

5.2.2.2 Advantages and disadvantages

CMA-ES is a very strong algorithm with state-of-the-art converging speed and optimization quality. As shown in the section **5.3.3** the algorithm outperforms both CBCW-PSO and CLPSO for many of the benchmark functions. But accroding to the “No Free Lunch Theorem”, there is no algorithm that optimizes all the functions. Every algorithm has it’s own weaknesses. According to [10], [11] and [12], though CMA-ES is a very strong algorithms in terms of convergence speed and optimization quality, it sometimes suffers from some drawbacks. As an example, CMA-ES experiences difficulties to solve the problems in which the atractor volume of the global optimum is small and an overall topology pointing to the global optimum is missing. This is mainly a concern when the function under optimization is non-separable. In [12] it has been shown that, in multi-funnel functions, where local optima can not be interpreted as perturbations to an underlying convex (unimodal) topology, the performance of the CMA-ES algorithm decreases. An example of a multi-funnel function is shown in **Figure 5.2**. As shown in the **Figure 5.2**, the function contains two funnels where the gray ridge seprates funnel 1 (left) from funnel 2 (right). Funnel 1 consists of global minimum as well as multiple local minima. Also several local minima are spread across the funnel 2. Funnel 2 is a broad funnel and any heuristic can be trapped in funnel 2. This topology is considered to be a hard one for optimization algorithms.

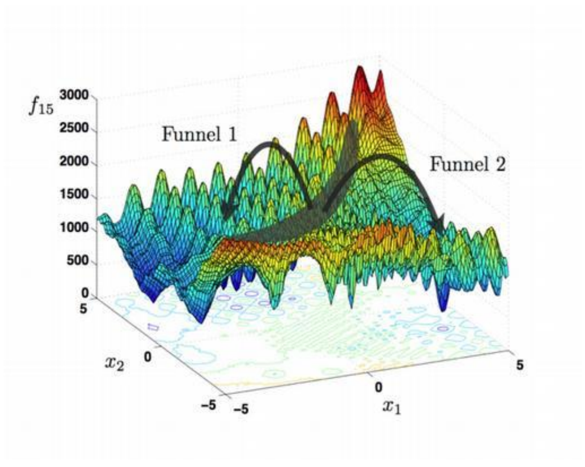


Figure 5.2 : Adapted from [11], a 2D multi-funnel function.

6. TEST AND COMPARISONS

6.1 Experimental Settings

As mentioned earlier, two rounds of experiments has been conducted. The experimental setting for both round of settings are identical unless otherwise is stated.
Each experiment consists of 50 runs. In the first round of experiments, the maximum number of evaluation counts is set to 40000 for both algorithms (CBCW-PSO and CLPSO). In the second round of experiments however, Since the number of evaluations in CMA-ES algorithm is not fixed and the stopping criteria of CL PSO and CBCW-PSO algorithms is to reach a certain evaluation count, we first conducted the tests using CMA-ES and measured the number of evaluation in each experiment for 50 runs and then set the maximum number of evaluations in CLPSO and CBCW-PSO algorithms to the maximum number of evaluations in CMA-ES algorithm for the corresponding experiment.

Since the environment under which the robot performance is evaluated is noisy, we also consider the effect of noise on benchmark functions to have an idea about the effect of noise (both uniform and Gaussian) on the robot experiments. This is why three separate experiments has been considered for each algorithm-benchmark function pair: (1) when the fitness value of the function has no noise (2) when the fitness value of the function has uniform noise and (3) when the function is considered to have Gaussian noise in fitness value.

The size of the external memory for best and worst global positions is 2 and 4 respectively. This is according to [13] where the best results are achieved. Also the sizes of local and external memories in CLPSO is set to 4 and 5 respectively, since the best results for this algorithm in the literature have been achieved using these settings. The swarm size in all the experiments is constant and set to be equal to 30 which is the dimension of the functions. The algorithm settings for which the experiments have been done are as follows: $\Delta\theta = 10^{-5}$, $\varepsilon = 10^{-6}$, $lower\ bound = 0.95$, $upper\ bound = 0.99$ The upper and lower bound of θ as well as Δθ are randomized with a small number, ε. The value of ε has been chosen to be smaller than the value of Δθ to prevent θ from exceeding its boundaries. Choosing a small value for ε also makes sure that θ's value maintains a smooth and slow motion between its boundaries as if Δθ where not randomized. The value for Δθ is chosen such that it shows a decrement rate close to that of [2] for the sake of comparison. The upper bound value is used as suggested in several publications and its validity has been verified in our experiments. The lower bound, however, is experimentally tuned to the value with which the best results are achieved.

Each experiment on the robot consists of 5 runs and has the following settings. Before each evaluation, the robot is beamed at the center of the field (coordinates *X=0* , *Y=0*) and the head's roll joint 'he1' in **Figure 5.1**, is set to 0 degrees and the head's pitch joint, 'he2' is set to the inverse (negative) value of the ball's vertical angle. This way, after the initialization the robot has a guaranteed visual of the ball. The time for each evaluation for all the algorithms is equal and is set to 11 seconds. After this initiation phase, there is one second waiting to make sure that everything is loaded and the robot will start the simulation normally without extreme system load. At this moment the simulator enters the kick-off phase where the robot will be able to move the joints. All the mentioned procedures has a duration of 2.5 seconds. This is where the robot starts to perform the kick action and it enters the Execution phase in the 3rd second of the game. No matter if the kick action is successful (the robot's leg hits the ball) or not, the robot's head is fixed on the ball during the experiment. The purpose is to determine a good quality fitness value, based on the distance of the ball to the robot. It has to be mentioned that if the robot falls (no matter if it sees the ball after falling or not), the last distance value, recorded when the robot was standing will be considered by the fitness evaluation.This is necessary due to the fact that if the robot falls backward, it still may be able to see the ball, however in this case, the distance between the robot and the ball will increase proportional to the length of the robot (57.5 cm). This, in turn, will decrese the fitness value of equation **(5.4)**, and the algorithm(s) may converge in favour of the individuals which result in falling of the robot backward. This is certainly not desirable. The same rule is accounted for the ball angle with respect to the robot, say, the last ball angle value for which the robot had a stable kinematic state (a no-fall state) is considered for the fitness evaluation. The stop criteria is set to maximum evaluation counts which is set to be 8000. The test platform is an 8 core (3.2 GHz) system running on Ubuntu 10.04 with 4 GB RAM.

6.2 Analysis of Parameter Settings

The value of the probability p has a direct effect on the convergence speed of the algorithm. The lower this value is, the more the algorithm switches to equations **(4.4)** or (4.5). This adds an extra amount to the velocity of the particle in the direction of the global best position. But there is a trade-off. Frequently using the extra velocity for a long time, results in convergence in a wrong direction. This is due to the fact that local optimum may be scattered and far from each other but at a relatively equal distance with respect to the particle. This is why an amount above 0.5 has been considered for the probability. However, our experiments show that setting the probability p proportional to 1-θ, increases the convergence speed, since, binding the p to the value of θ, which is a randomized value, provides more diversification. Replacing the linearly decreasing inertia in equation **(4.5)** with θ , which is subject to multiple restarts, also shows to be a very influencing factor on convergence speed. This is the methodology of choice in our experiments.

One can easily show through experiments that eliminating one the parameters introduced in CBCW-PSO decreases the performance of the algorithm. This is to say that if we reject using the coefficient C and instead employ only the inertia restart, or vice versa the performance will decrease. Though one can replave the probability p to a constant positive less than 1 instead of changing it dynamically with the complement value of inertia. For some functions it does not affect the performance but for some other functions it will dramatically decrease the performance. Also lower and upper bounds have a huge effect on the performance and thus cannot be eliminated.

6.3 Experimental Results on Benchmark Functions

A preliminary set of experiments has been conducted on CBCW-PSO and CLPSO on some of the benchmark functions in [6],[7] and [8]. The benchmark functions along with some of their properties are shown in **Table 5.1**. The goal is to compare the convergence speed of CBCW-PSO and that of CLPSO. The results are shown in **Table 6.1**. As can be seen in the table, CBCW-PSO outperforms CLPSO in terms of solution quality. The results of the employed t-test show the cases where CBCW-PSO is statistically significant than CLPSO. In the last column, S+ means that CBCW-PSO is statistically significant comparing to CLPSO and S means that none of the two methods are statistically significant. However, considering the convergence speed as a criteria, the minimum evaluation count in which each algorithm finds the optima (first-hit) is also measured in the experiments. The results are shown in **Table 6.2**. Again here S+ means that CBCW-PSO is statistically significant than CLPSO and S means that none of the two methods are statistically significant. Achieving the satisfactory results in the preliminary experiments, we ran another round of tests and included CMA-ES algorithm in comparisons. In this new round of comparative tests, as shown in **Table 5.2**, special attention has been paid to more complicated benchmark functions .

Table 6.1 : Perliminary experimental results

Functions	Dim	CLPSO			CBCW-PSO			t-test
		Min	Max	Avg	Min	Max	Avg	
$f_1(\vec{x})$	30	0	0	0	0	0	0	S
$f_2(\vec{x})$	30	0	0	0	0	0	0	S
$f_3(\vec{x})$	30	0	0	0	0	0	0	S
$f_4(\vec{x})$	30	0	0	0	0	0	0	S
$f_5(\vec{x})$	30	0	0	0	0	0	0	S
$f_6(\vec{x})$	30	0	0	0	0	0	0	S
$f_7(\vec{x})$	30	0	0	0	0	0	0	S
$f_8(\vec{x})$	30	0.033	7.93	3.899	**0.001**	**4.001**	**0.911**	**S+**
$f_9(\vec{x})$	30	-12451	-11622	-12107.5	**-12569.5**	**-12332**	**-12438**	**S+**
$f_{10}(\vec{x})$	30	48.915	237.94	173.8	**0**	**1.989**	**0.994**	**S+**
$f_{11}(\vec{x})$	30	0	0.11	0.042	**0**	**0.041**	**0.012**	**S+**
$f_{12}(\vec{x})$	30	2.2209	5.22	3.318	**0**	**0**	**0**	**S+**
$f_{13}(\vec{x})$	30	2.7555	7.19	5.922	**0**	**1.57**	**0.15**	**S+**
$f_{14}(\vec{x})$	2	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	S
$f_{15}(\vec{x})$	2	0.398	0.398	0.398	0.398	0.398	0.398	S
$f_{16}(\vec{x})$	2	0	0	0	0	0	0	S

They are more complicated in the sense that they are highly multi-modal, highly multi-funnel and hybrid. In **Table 6.3**, the minimum, average and the value of standard deviation, obtained over 50 runs of CMA-ES, CLPSO and CBCW-PSO are given. Outcomes with a difference less than or equal to 10^{-5} with respect to the known global optima, is considered as the value of the global optima.

Table 6.2 : First Hit

Functions	**CLPSO**	**CBCW-PSO**	**t-test**
$f_1(\vec{x})$	122820	**15255**	**S+**
$f_2(\vec{x})$	165900	**31419**	**S+**
$f_3(\vec{x})$	299880	**122265**	**S+**
$f_4(\vec{x})$	1199400	**77832**	**S+**
$f_5(\vec{x})$	100920	**2340**	**S+**
$f_6(\vec{x})$	122280	**7605**	**S+**
$f_7(\vec{x})$	247860	**33582**	**S+**
$f_8(\vec{x})$	-	-	
$f_9(\vec{x})$	-	**1109850**	**S+**
$f_{10}(\vec{x})$	-	**1076010**	**S+**
$f_{11}(\vec{x})$	1094340	**963669**	**S**
$f_{12}(\vec{x})$	-	**57153**	**S+**
$f_{13}(\vec{x})$	-	**646980**	**S+**
$f_{14}(\vec{x})$	3180	**531**	**S+**
$f_{15}(\vec{x})$	2700	**489**	**S+**
$f_{16}(\vec{x})$	2120	**775**	**S+**

The dimension of each function is set to 30 for all the experiments. Better results are shown in bold in the table. An ANOVA test at a significance level of 0.95 has been performed to test for the statistical significance of the differences. In the last column, S_{CBCW} means that CBCW PSO results are statistically significant than the results of the other algorithms and S_{CL} means that the CLPSO algorithm performs better than others and finally S_{CMA} means that CMA-ES offers better solutions. The details of the ANOVA analysis can be found in **Figures A.1, A.2** and **A.3** in the Appendices. Also multi column comparison results for all the experiments can be found in **Figures A.4, A.5** and **A.6** in the Appendices. These figures, show those algorithms which are statistically significant from the others.

Table 6.3 : Comparative results

Functions		CBCW-PSO			CLPSO			CMA-ES			ANOVA
		Min	μ	σ	Min	μ	σ	Min	μ	σ	
No Noise	$f_7(\vec{x})$	-179.9	-176.6	11.0	-144.8	-73.8	51.85	**-180**	**-179**	**0.001**	S_{CMA}
	$f_8(\vec{x})$	**-119.2**	**-119.0**	**0.09**	-119.1	-118	0.053	-118	-118	0.068	S_{CBCW}
	$f_9(\vec{x})$	-291.1	-254.0	25.4	-316.5	-297	13.0	**-311**	**-289**	**27.04**	S_{CMA}
	$f_{10}(\vec{x})$	-260.3	-174.7	48.3	-224.76	-148	35.36	**-313**	**-298**	**25.18**	S_{CMA}
	$f_{11}(\vec{x})$	113.1	118.9	3.3	118.08	128.8	3.308	**93.8**	**103.7**	**4.04**	S_{CMA}
	$f_{13}(\vec{x})$	-125.3	-119.7	2.78	-120.76	-110	9.17	**-128**	**-126**	**0.75**	S_{CMA}
	$f_{14}(\vec{x})$	**-288.2**	**-287.0**	**0.44**	-287.18	-286	0.27	-286	-285	0.19	S_{CBCW}
	$f_{17}(\vec{x})$	245.4	414.2	111.1	240.73	412.4	133.0	**155.2**	**171.7**	**9.31**	S_{CMA}
Uniform Noise	$f_7(\vec{x})$	-178.9	-170.5	24.33	-169.91	-106	44.64	**-178**	**-178**	**0.01**	S_{CMA}
	$f_8(\vec{x})$	**-119.0**	**-118.7**	**0.08**	-118.9	-118	0.07	-118	-118	0.07	S_{CBCW}
	$f_9(\vec{x})$	-282.6	-234.2	37.56	**-288.7**	**-256**	**21.89**	-279	-131	33.44	S_{CL}
	$f_{10}(\vec{x})$	**-270.9**	**-162.8**	**60.67**	-167.6	-60.1	43.45	-310	-114	31.97	S_{CBCW}
	$f_{11}(\vec{x})$	**113.7**	**121.8**	**3.96**	128.9	134.2	2.05	131.1	135.4	1.40	S_{CBCW}
	$f_{13}(\vec{x})$	-124.7	-117.2	4.82	-32.01	390.7	495.8	**-127**	**-118**	**6.88**	S_{CMA}
	$f_{14}(\vec{x})$	**-287.1**	**-286.3**	**0.31**	-286.6	-285	0.27	-286	-285	0.15	S_{CBCW}
	$f_{17}(\vec{x})$	250.6	475.9	161.6	382.3	562.4	105.3	**317.3**	**368.9**	**17.75**	S_{CMA}
Gaussian Noise	$f_7(\vec{x})$	-178.9	-169.9	18.0	-163.1	-117	32.25	**-178**	**-178.8**	**0.026**	S_{CMA}
	$f_8(\vec{x})$	**-118.9**	**-118.7**	**0.084**	-118.8	-118	0.05	-118	-118.6	0.07	S_{CBCW}
	$f_9(\vec{x})$	**-267.2**	**-209.1**	**33.48**	-267.46	-203	39.14	-115	-86.15	16.4	S_{CBCW}
	$f_{10}(\vec{x})$	**-248.1**	**-137.9**	**68.62**	-85.6	-2.29	46.06	-99.7	-65.72	16.45	S_{CBCW}
	$f_{11}(\vec{x})$	**113.3**	**121.9**	**3.88**	129.2	134.8	1.62	132.3	135.4	1.37	S_{CBCW}
	$f_{13}(\vec{x})$	**-124.7**	**-115.9**	**4.70**	-70.1	518.1	985.3	-114	-109.7	1.58	S_{CBCW}
	$f_{14}(\vec{x})$	**-286.6**	**-286.0**	**0.29**	-286.5	-285	0.21	-286	-285.7	0.14	S_{CBCW}
	$f_{17}(\vec{x})$	235.4	458.2	160.4	364.9	547.4	162.2	**376.5**	**414.4**	**17.6**	S_{CMA}

Table 6.3 shows that, for non-noisy function CMA-ES has a better performance comparing with CBCW-PSO and CLPSO. However functions, f_8 and f_{14} are exceptions. In case of function f_8 since the optimum value is in the boundary and CBCW-PSO, due to its smart boundary control, has a good ability to find the optima at the boundaries, it outperforms the other two algorithms. However it also implies that CBCW-PSO performs well when the basin of attraction is small. As of f_{14}, the Schaffer's function, there are multiple local optima in the form of circles and the global optima lies in the center circle in the function. **Figure 6.1** shows the function in a 2D demonstration.

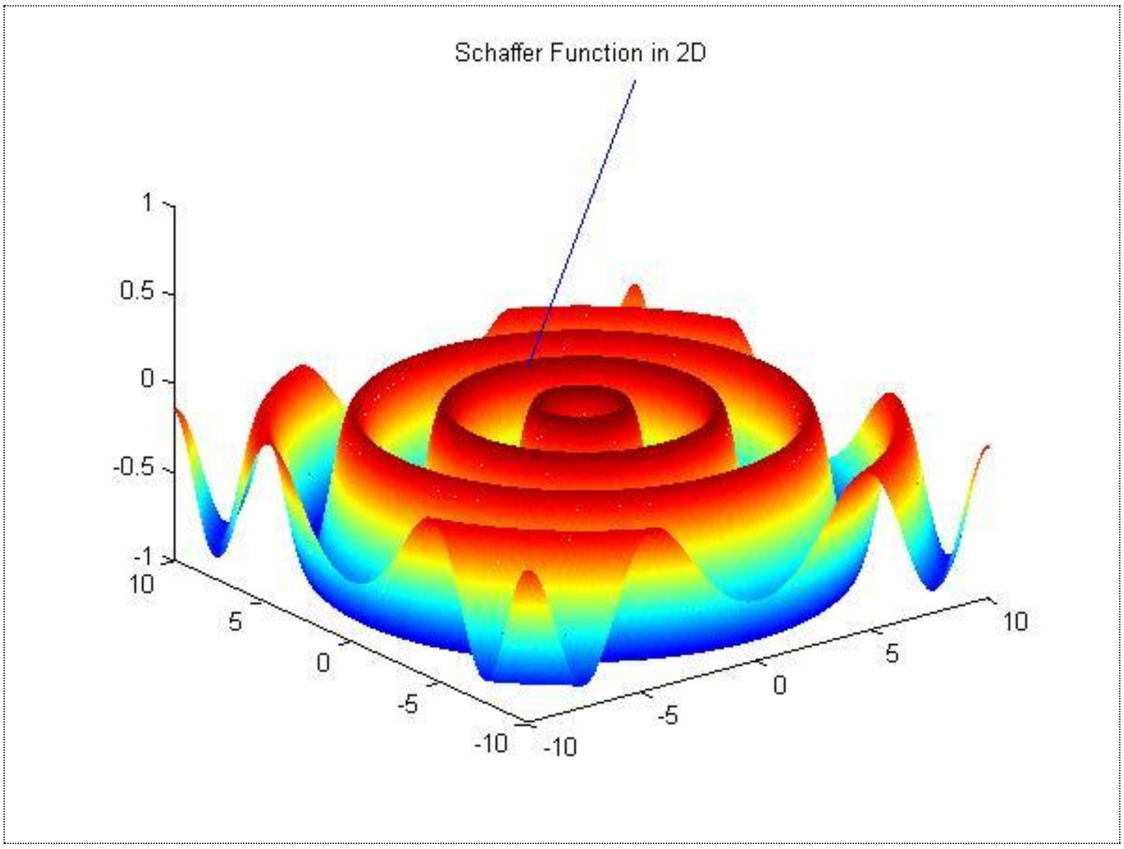


Figure 6.1 : A demonstration of Schaffer's function in 3D

Here the inertia restart strategy as well as linear decrement of the inertia value, helps the CBCW-PSO to escape from local optima and avoids the premature convergence. At the same time the structure of the function could be considered a plus for the employed external memory policy since subsequent (neighboring) local maxima and minima are much more probable and hence the value of the C variable in equations **(4.3), (4.4)** and **(4.5)** maintains a constant value close to 1 which at the same time increases the step size and forces the particles to go one step higher in the function's circle levels. This helps the algorithm in the sense that it prevents the particles from getting trapped inside many local optima in the function.

The results are also presented in graphical form in **Figure 6.2**. Please note that if there is a missing algorithm in the figures, it means that the values produced by that algorithm is so high that plotting it may avoid investigating the details of the other

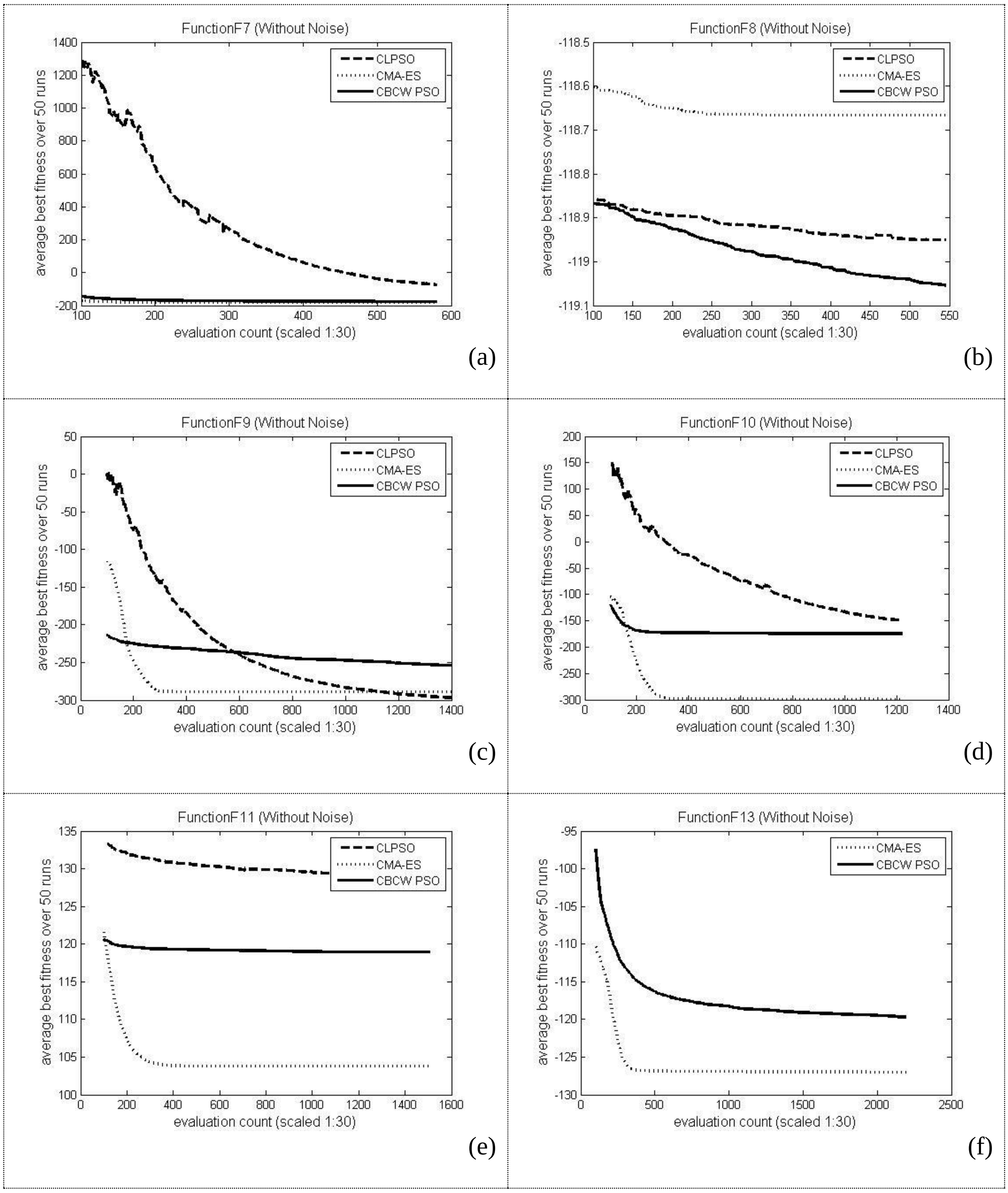
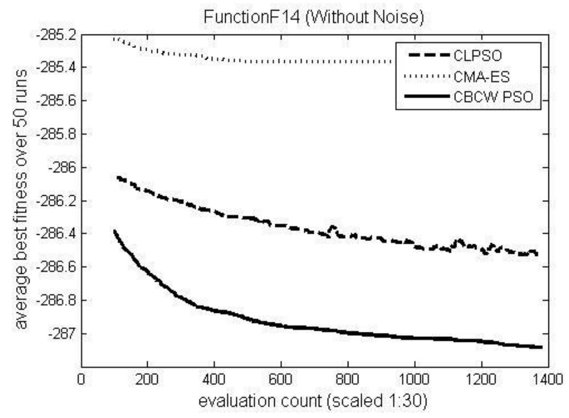
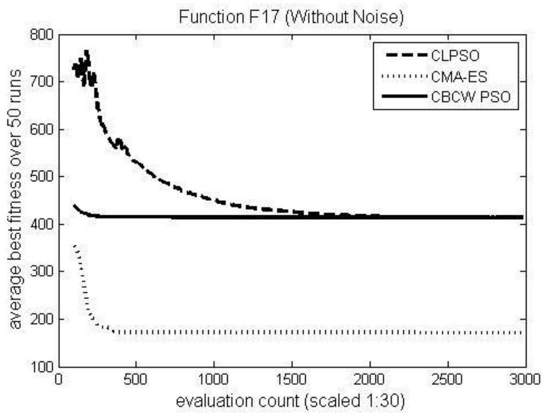


Figure 6.2 : The evolution of the best fitness, averaged over 50 runs for each function for non-noisy fitness values. (a)f_7. (b)f_8. (c)f_9. (d)f_{10}.(e) f_{11}. (f) f_{13}. (g) f_{14}. (h) f_{17}



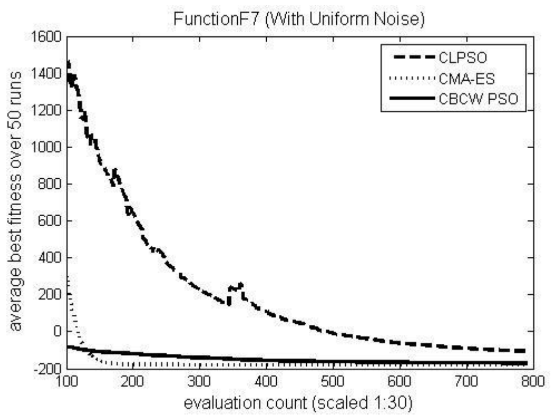
(g)



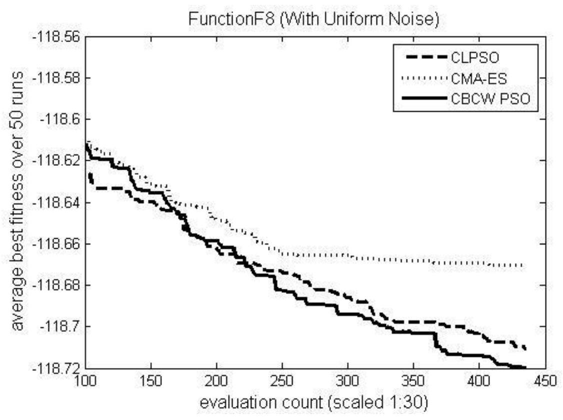
(h)

Figure 6.2 (continued): The evolution of the best fitness, averaged over 50 runs for each function for non-noisy fitness values. (a)f_7. (b)f_8. (c)f_9. (d)f_{10}. (e) f_{11}. (f) f_{13}. (g) f_{14}. (h) f_{17}

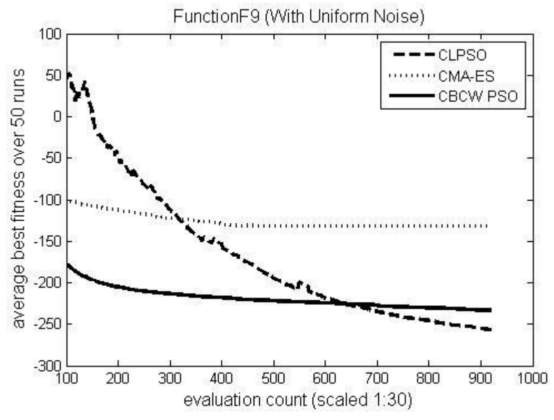
algorithms. A missing CLPSO plot in **Figure 6.2 (f)** is an example. In such cases **Table 6.3** could be considered. Again in **Table 6.1** as well as in **Figure 6.3**, one can see that CBWC-PSO outprtforms the other two algorithms in functions f_8, f_{10}, f_{11} and f_{14} when there is a Uniform Noise added to the fitness value. The function f_{10} is more or less similar to functions f_8 and f_{14} in the sense that the number of local optima is very large, and that they are regularly spread over the function space. Again, the inertia restart strategy as well as the new velocity update helps the CBCW-PSO to escape from those local optima. The regular spread of local optima across the search space has the obvious advantage that they produce almost subsequent worst and best neighborhoods which results in a continuous boost in the value of the variable *C* in equations **(4.3), (4.4)** and (**4.5)** and this forces the particles to check for new and better neighborhoods (if any). This is similar to the case of function f_{14} in the previous discussion. In general, this yields that, whenever the basin of attraction has a small size or there is an overal topology, pointing towards the global optimum, our algorithm performs well. As discussed earlier, the reason for this is the way the *C* value in equations **(4.3), (4.4)** and **(4.5)** is boosted in such topologies.



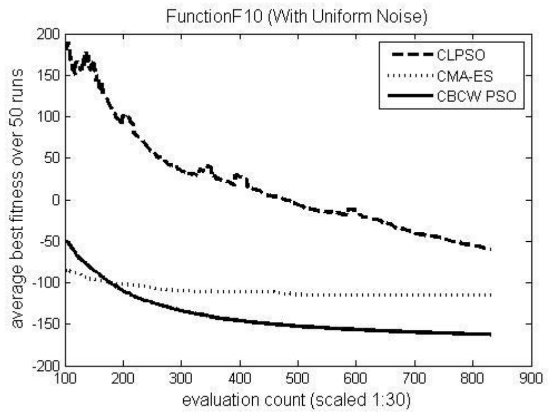
(a)



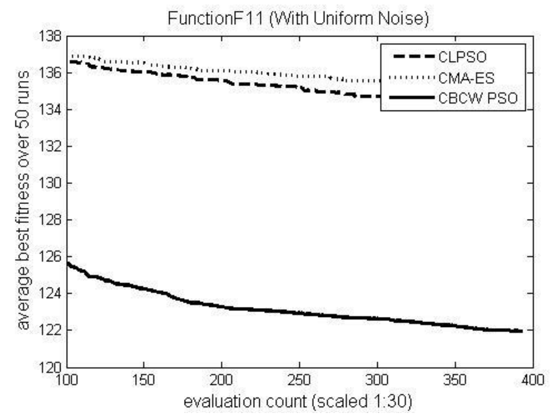
(b)



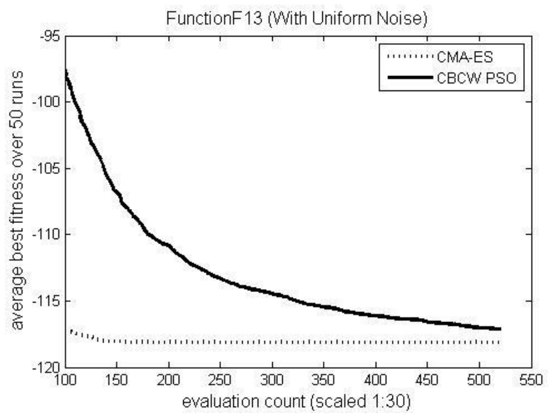
(c)



(d)

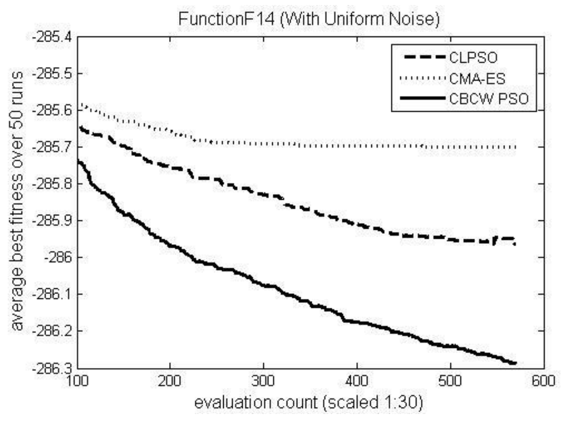


(e)

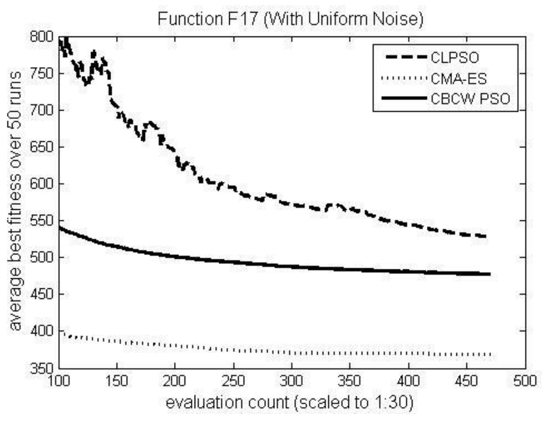


(f)

Figure 6.3 : The evolution of the best fitness, averaged over 50 runs for each function for fitness values with uniform noise. (a)f_7. (b)f_8. (c)f_9. (d)f_{10}. (e) f_{11}. (f) f_{13}. (g) f_{14}. (h) f_{17}



(g)



(h)

Figure 6.3 (continued): The evolution of the best fitness, averaged over 50 runs for each function for fitness values with uniform noise. (a)f_7. (b)f_8. (c)f_9. (d)f_{10}. (e) f_{11}. (f) f_{13}. (g) f_{14}. (h) f_{17}

Finally **Figure 6.4** and the last 8 rows of **Table 6.1** show the experiment results for the case when Gaussian noise is added to the fitness value. The better results in favour of CBCW-PSO are partially due to the fact that the original PSO and the descendants of this algorithm are noise resistant. It's easy to see that the performance difference between CBCW-PSO and CLPSO are preserved when the noise type is changed. In case of the CMA-ES algorithm, however, setting the NOISE option in CMA-ES algorithm to ON, ends up in worse results for noisy functions and, Setting the same option to OFF for the same noisy function, ends up in fitness values close to that of the first 8 rows of **Table 6.1** (this is to say that the results will be similar to the case where there is no noise in the fitness value). However, in addition to discussing the effect of the noise and it's type on the performance of each algorithm, our goal here is also to show that for some functions, no matter if CMA-ES treats the function as a noisy function or not, and if there actually is noise or not, there are cases where CBCW-PSO outperforms the CMA-ES. f_{14} is an example that emerges from these experiments. As stated earlier in section **5.2.2.2**, it has been shown in literature that, in multi-funnel functions, where local optima can not be interpreted as perturbations to an underlying convex (unimodal) topology, the performance of the CMA-ES algorithm decreases. This is why we believe that, in fact in case of noisy or multi-funnel functions, the main advantage of our algorithm is it's independence from the underlying global topology and the use of intuitive coefficient which allows it to simply escape local optima towards better coordinates in the search space. A question may rise about the performance of the CLPSO algorithm in multi-funnle and/or noisy functions. The main reason why CLPSO fails to converge to the global optimum in these functions is it's time consuming initialization phase. Observations during the experiemnts (as well as the graphical representations in **Figures 6.2, 6.3** and **6.4**) show that the CLPSO algorithm consumes a lot of time in the initialization phase and due to its random substitution policy in external memories, it also may reset to a worse fitness value as global best (the various peaks that are observable in many graphics, representing CLPSO's performance).

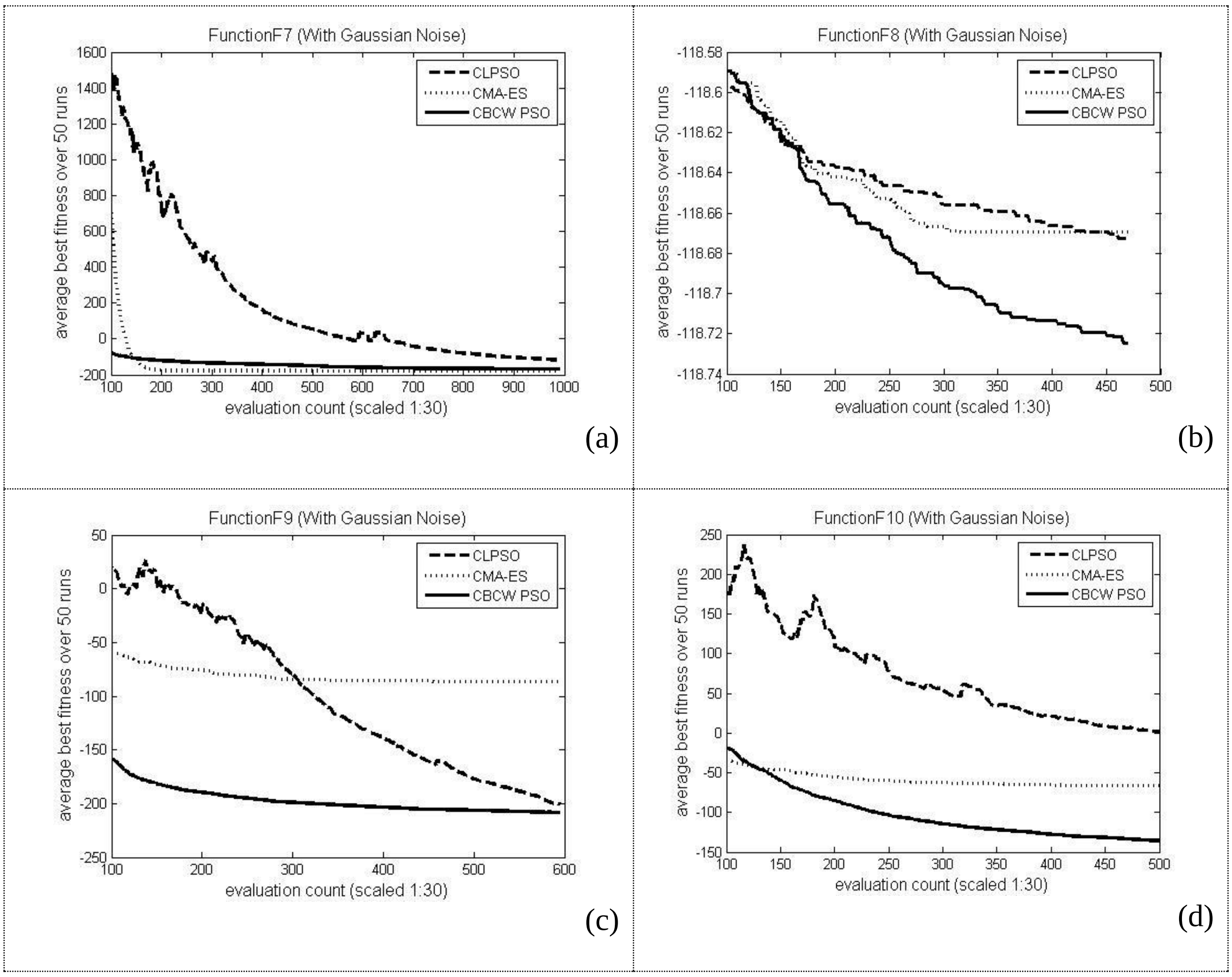
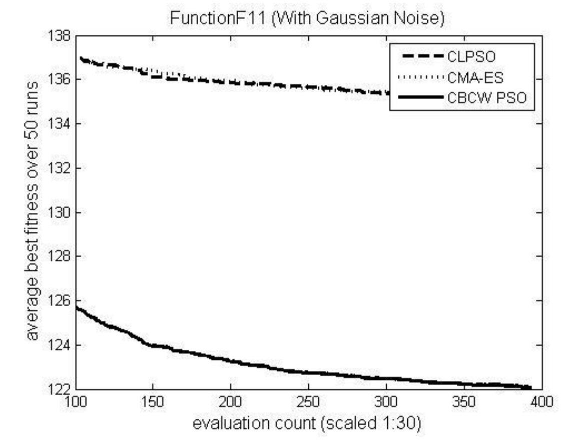
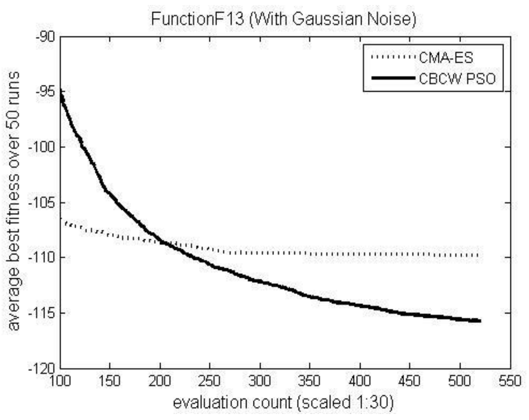


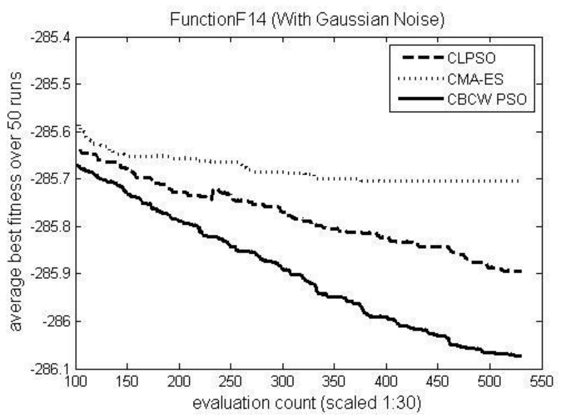
Figure 6.4 : The evolution of the best fitness, averaged over 50 runs for each function for fitness values with Gaussian Noise. (a)f_7. (b)f_8. (c)f_9. (d)f_{10}. (e) f_{11}. (f) f_{13}. (g) f_{14}. (h) f_{17}



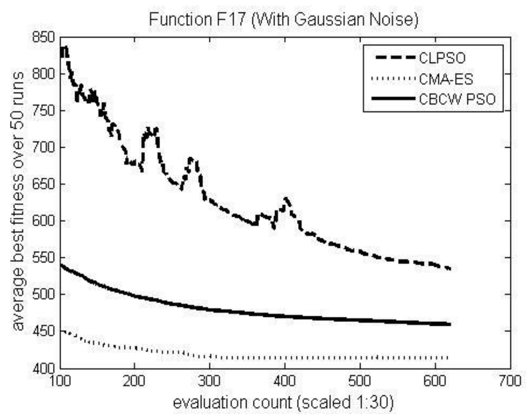
(e)



(f)



(g)



(h)

Figure 6.4 (continued): The evolution of the best fitness, averaged over 50 runs for each function for fitness values with Gaussian Noise. (a)f_7. (b)f_8. (c)f_9. (d)f_{10}. (e) f_{11}. (f) f_{13}. (g) f_{14}. (h) f_{17}

6.4 Experimental Results on The Robot

Finally, **Figure 6.5** and **Table 6.4** show the experimental results on the robot kick action. As can be seen, CBCW-PSO outperforms the other two algorithms. From this experiment, one can conclude in many ways about the robot's kick search space. Earlier, during the experiments on benchmark functions, we saw that small basin of attraction, missing overall topology in the search space and multi-funnelty in the function under optimization may all be the reasons for an inferior performance In CMA-ES and a superior performance in CBCW-PSO. However, also the performance of CLPSO helps us to provide an explanation about the reason why, CBCW-PSO outperforms CMA-ES. The resulting fitness evolution is similar to that of f_{14} , in terms of the order with which each algorithm performs better and also the way the best fitness value decsends. The resulting kick action has a range of nearly 6 meters. The ball does not deviate from the x axis of robot's torso as desired. The described kick action has been utilized in RoboCup competitions at Istanbul, 2011 and the robot was able to kick a goal score using the designed kick action. An ANOVA analysis also has been made on the outcomes of the three algorithms in which can be seen in Figure A.4 in the Appendices (**Appendix A.1**). **Figure 6.6** shows the resulting kick in Simspark simulator.

Table 6.4 : Comparative results of the robot experiment

Functions	**Dim**	**CBCW-PSO**			**CLPSO**			**CMA-ES**			**ANOVA**
		Min	**μ**	**σ**	**Min**	**μ**	**σ**	**Min**	**μ**	**σ**	
$Robot'sKick$	40	**0.170**	**0.17956**	**0.0098849**	0.20704	0.21373	0.00626	0.19646	0.23387	0.051844	S_{CBCW}

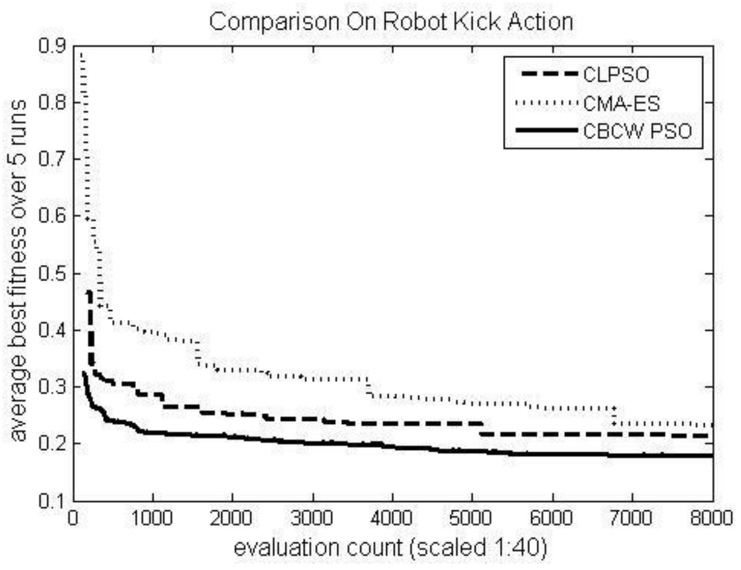


Figure 6.5 : Best fitness evolution of the robot experiment

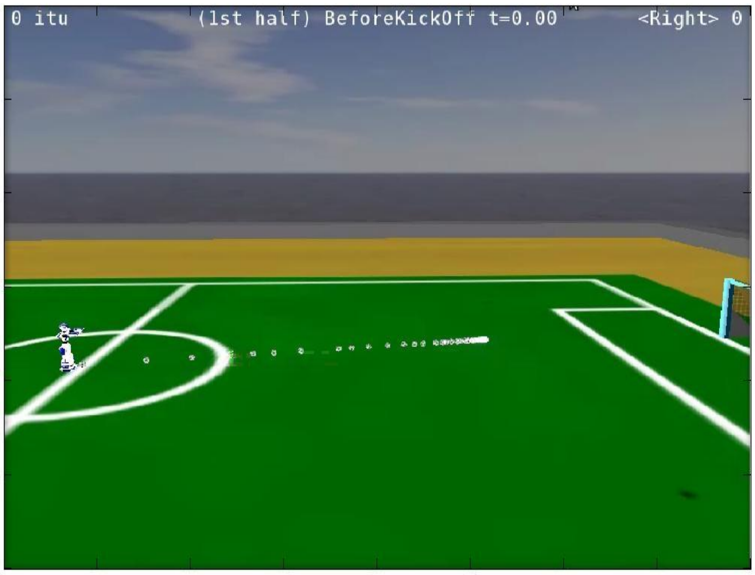


Figure 6.6 : The resulting kick action

7. CONCLUSIONS

In this thesis, we have proposed a new and novel PSO, based on external memory for Optimization problems. The algorithm has been subjected to comparative experiments on well-known benchmark functions as well as an optimization problem defined on the robot action (kick action). We showed that the proposed algorithm outperforms a recent improvement of PSO, namely CLPSO in the benchmark function experiments. Also in some cases, the proposed algorithm outperforms the CMA-ES algorithm. It has been explicitly shown that our new algorithm outperforms the other two algorithm in terms of convergence speed and the solution quality for the cases where the function under optimization is either noisy, multi-funnel, has small basin of attraction or does not have a global topology. Also the algorithm performs very well for the robot case where an optimum set of parameters are being sought for the kick action.Future work will include an adaptation strategy with which the algorithm tunes it's parameter according to the function it optimizes. These parameters include the sizes of the external memories and lower and upper band values in the algorithm.

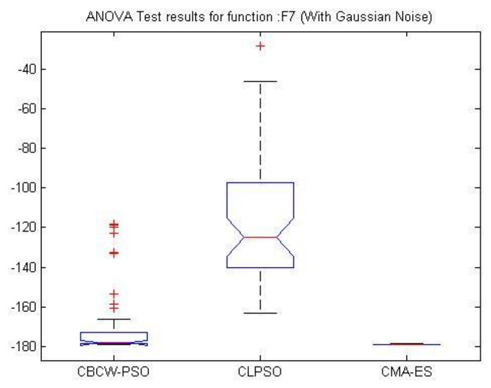
REFERENCES

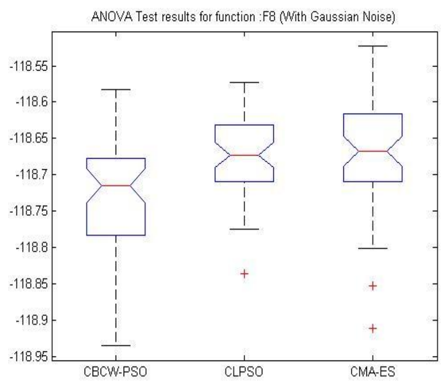
J.Kennedy , R.C.Eberhart . (1995) “A New Optimizer Using Particle Swarm Theory” , Sixth International symposium on Micro Machine and Human Science , IEEE
A.Acan . (2009) “A Memory-Based Colonization Scheme for Particle Swarm Optimization”, IEEE Congress on Evolutionary Computation (CEC)
Y.Shi ,R.C.Eberhart , (1998) “Parameter Selection in Particle Swarm Optimization” , Proceedings of the 7th International Conference on Evolutionary Programming VII. (EP)
M.Clerc (1999), “The Swarm and The Queen: Towards a deterministic and Adaptive Particle Swarm Optimization” , IEEE Congress on Evolutionary Computation (CEC). Eberhart,
R.C., Shi, Y. (2000) "Comparing Inertia Weights and Constriction Ffactors in Particle Swarm Optimization", IEEE Proceedings of the Congress on Evolutionary Computation, La Jolla, CA , USA
R.Thomsen ,J.Vesterstrom. (2004) , “A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems”, IEEE Congress on Evolutionary Computation (CEC).
P. N. Suganthan1, N. Hansen, J. J. Liang, K. Deb, Y. -P. Chen, A. Auger, S. Tiwari. (2005), “Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization” , IEEE Congress on Evolutionary Computation (CEC)
M. Molga, C. Smutnicki. (2005), Test functions for optimization needs, Available at http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf
Boedecker, J., Dorer, K., Rollman, M., Xu, Y., Xue, F., Buchta,. M., Vatankhah, H.,”Simspark Manual” , 2010
C.L.Müller, I.F.Sbalzarini . (2009) “A Tunable Real-World Multi-Funnel Benchmark Problem For Evolutionary Optimization”, In Proc. Intl. Joint Conf. Computational Intelligence (IJCCI), pages 248–253, Funchal, Madeira, Portugal, October 5–7 2009. INSTICC Press.
C. L. Müller, B. Baumgartner, and I. F. Sbalzarini. (2009) Particle swarm CMA evolution strategy for the optimization of multi-funnel landscapes. In Proc. IEEE Congress on Evolutionary Computation (CEC), pages 2685-2692, Trondheim, Norway, May 2009
Stefan Kern, Sibylle D. Müller, Nikolaus Hansen, Dirk Büche, Jiri Ocenasek and Petros Koumoutsakos. (2004) “Learning probability distributions in continuous evolutionary algorithms – a comparative review”, Journal of Natural Computing, Volume 3 Number 1pages 77-112. Netherlands
S.Asta, Ş.E.Uyar. (2011), “A Novel Particle Swarm Optimization”, In proceedings of Artificial Evolution, 24-26 October 2011, Angers, France.

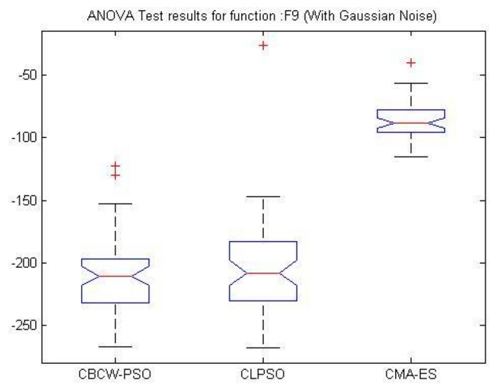
APPENDICES

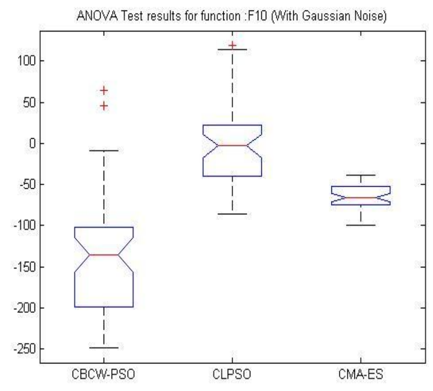
APPENDIX A : ANOVA Test Results

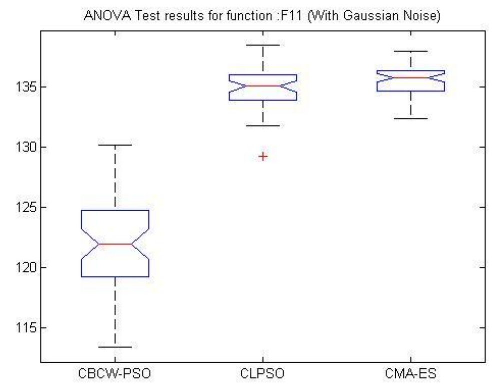
APPENDIX A

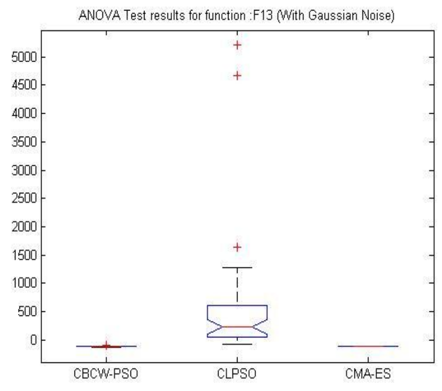

ANOVA Test results for function :F7 (With Gaussian Noise)
-40
-60
-80
-100
-120
-140
-160
-180
CBCW-PSO
CLPSO
CMA-ES

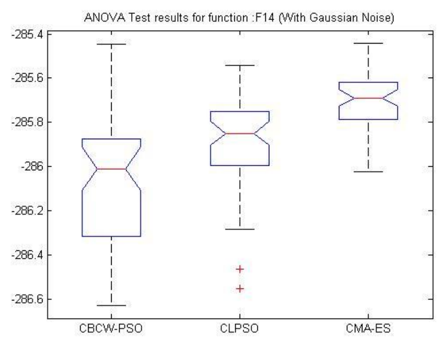

ANOVA Test results for function :F8 (With Gaussian Noise)
-118.55
-118.6
-118.65
-118.7
-118.75
-118.8
-118.85
-118.9
-118.95
CBCW-PSO
CLPSO
CMA-ES


ANOVA Test results for function :F9 (With Gaussian Noise)
-50
-100
-150
-200
-250
CBCW-PSO
CLPSO
CMA-ES


ANOVA Test results for function :F10 (With Gaussian Noise)
100
50
0
-50
-100
-150
-200
-250
CBCW-PSO
CLPSO
CMA-ES


ANOVA Test results for function :F11 (With Gaussian Noise)
135
130
125
120
115
CBCW-PSO
CLPSO
CMA-ES


ANOVA Test results for function :F13 (With Gaussian Noise)
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
CBCW-PSO
CLPSO
CMA-ES



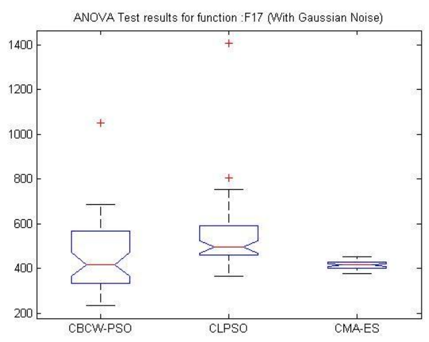
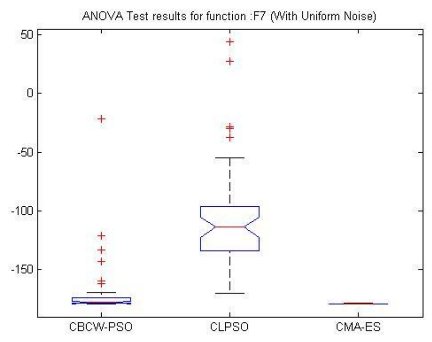
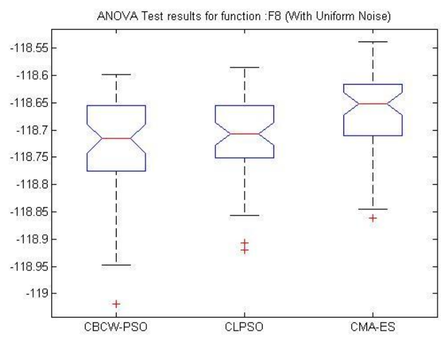
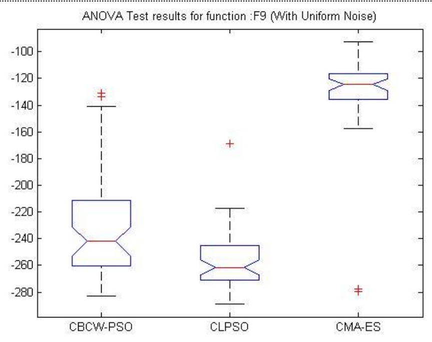
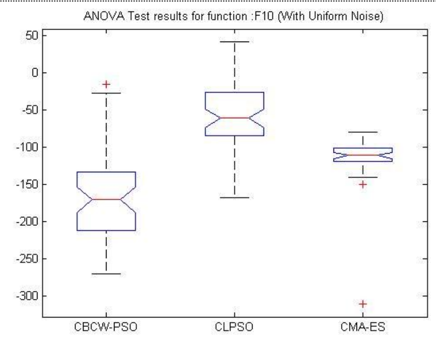


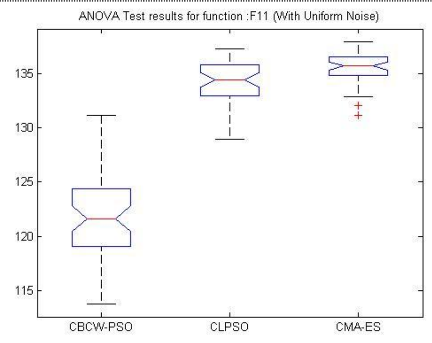
Figure A.1 : The ANOVA Test results for various benchmark functions with noisy fitness value (Gaussian noise type).

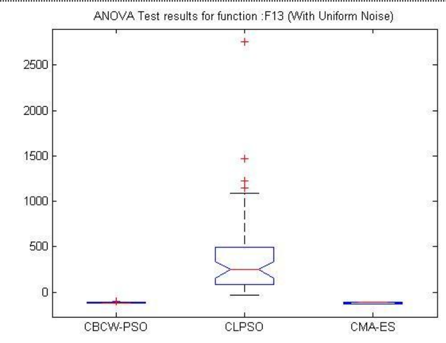












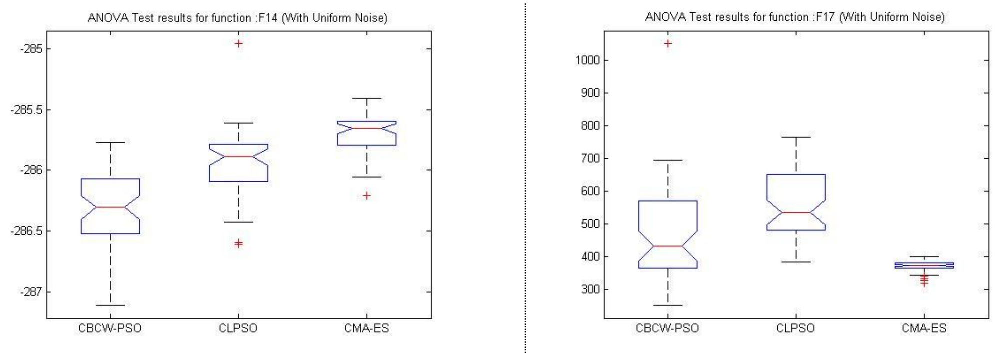
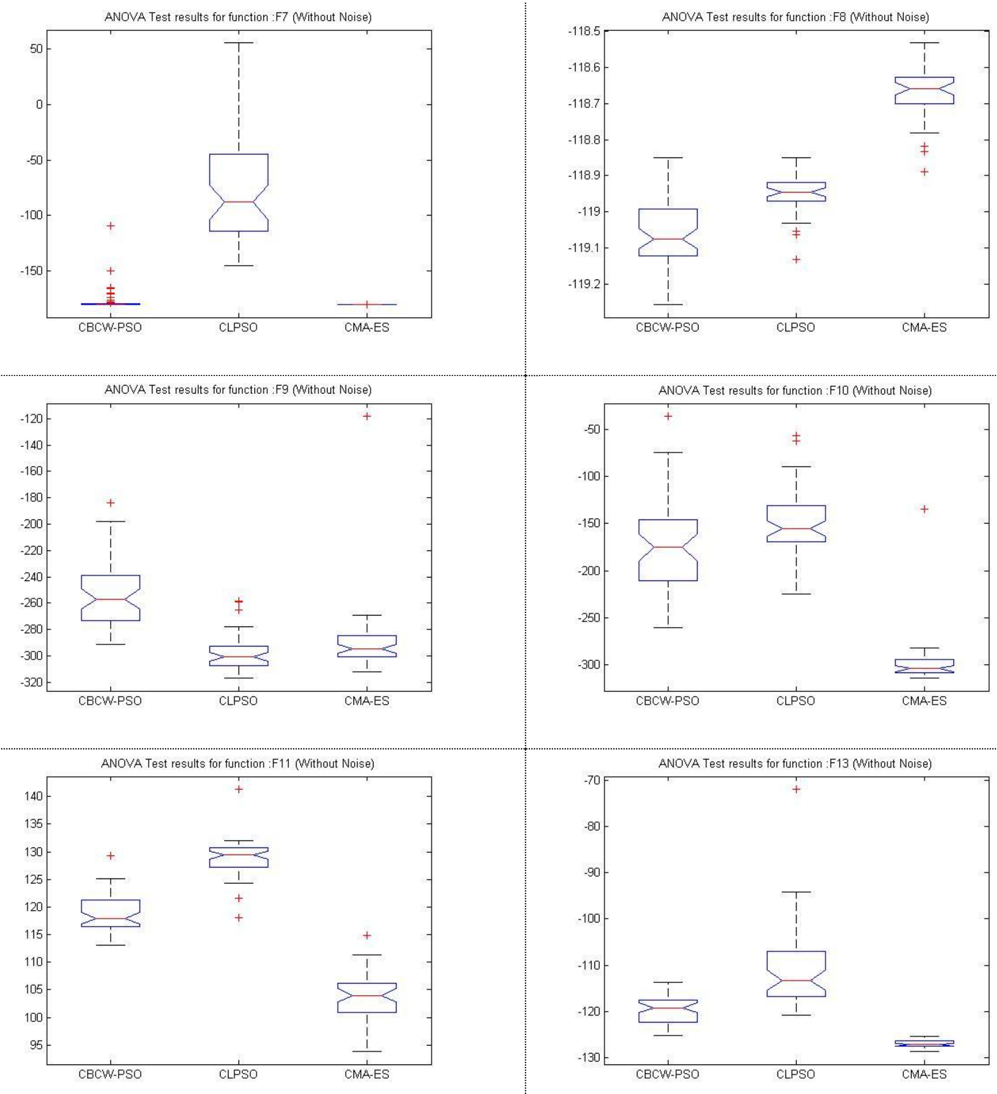


Figure A.2 : The ANOVA Test results for various benchmark functions with noisy fitness value (Uniform noise type).



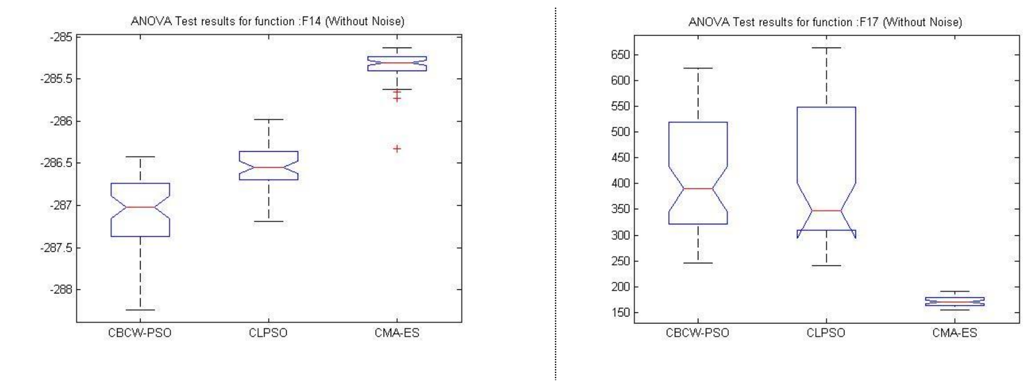
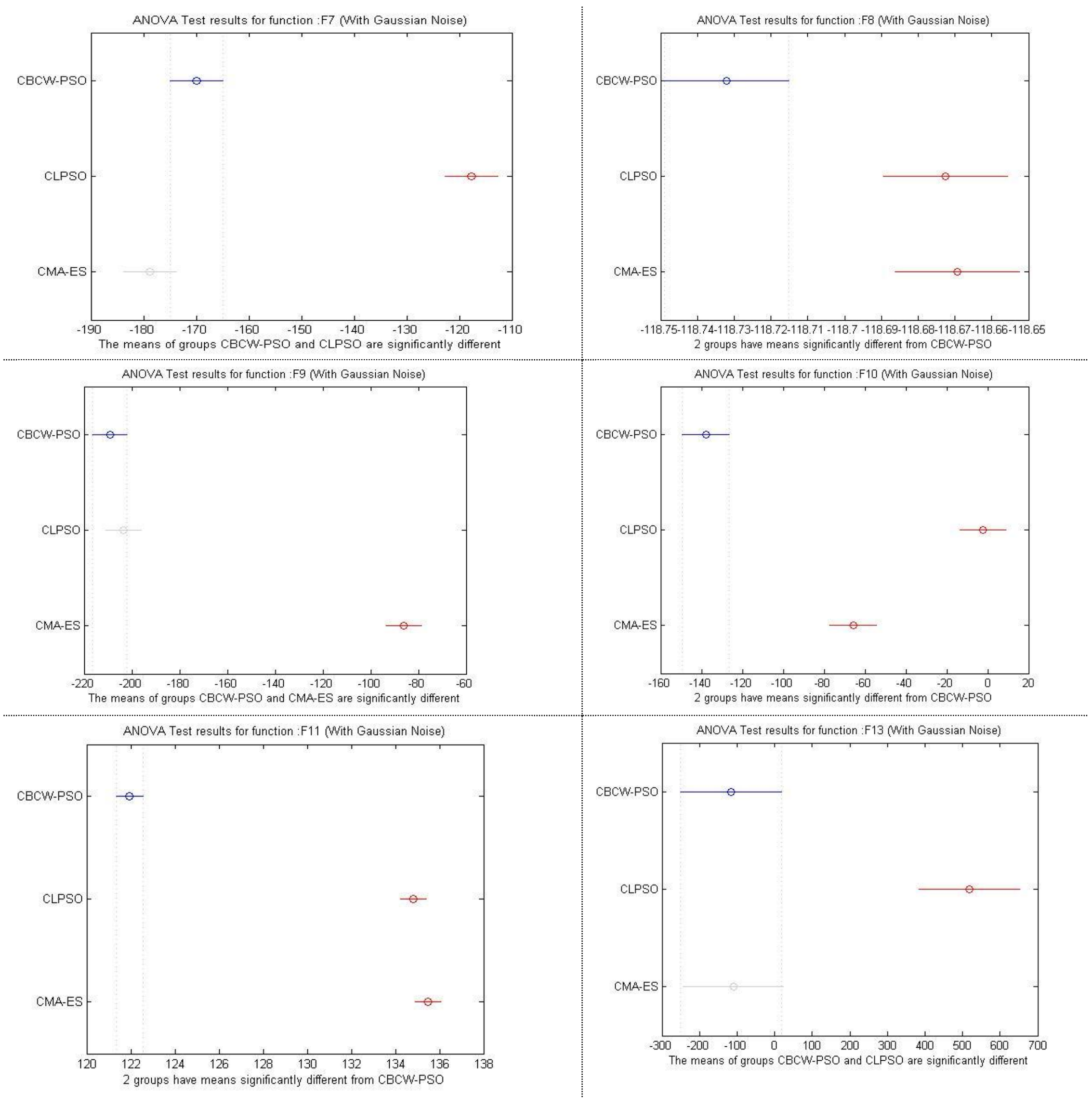


Figure A.3 : The ANOVA Test results for various benchmark functions with non-noisy fitness value.



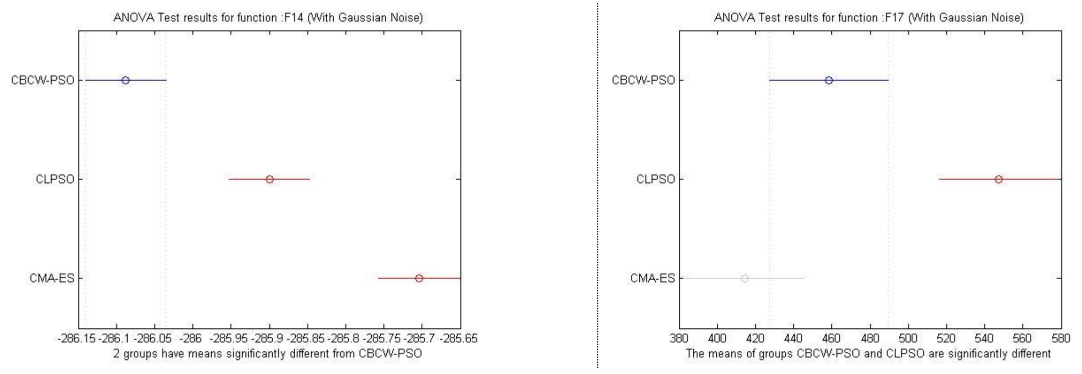
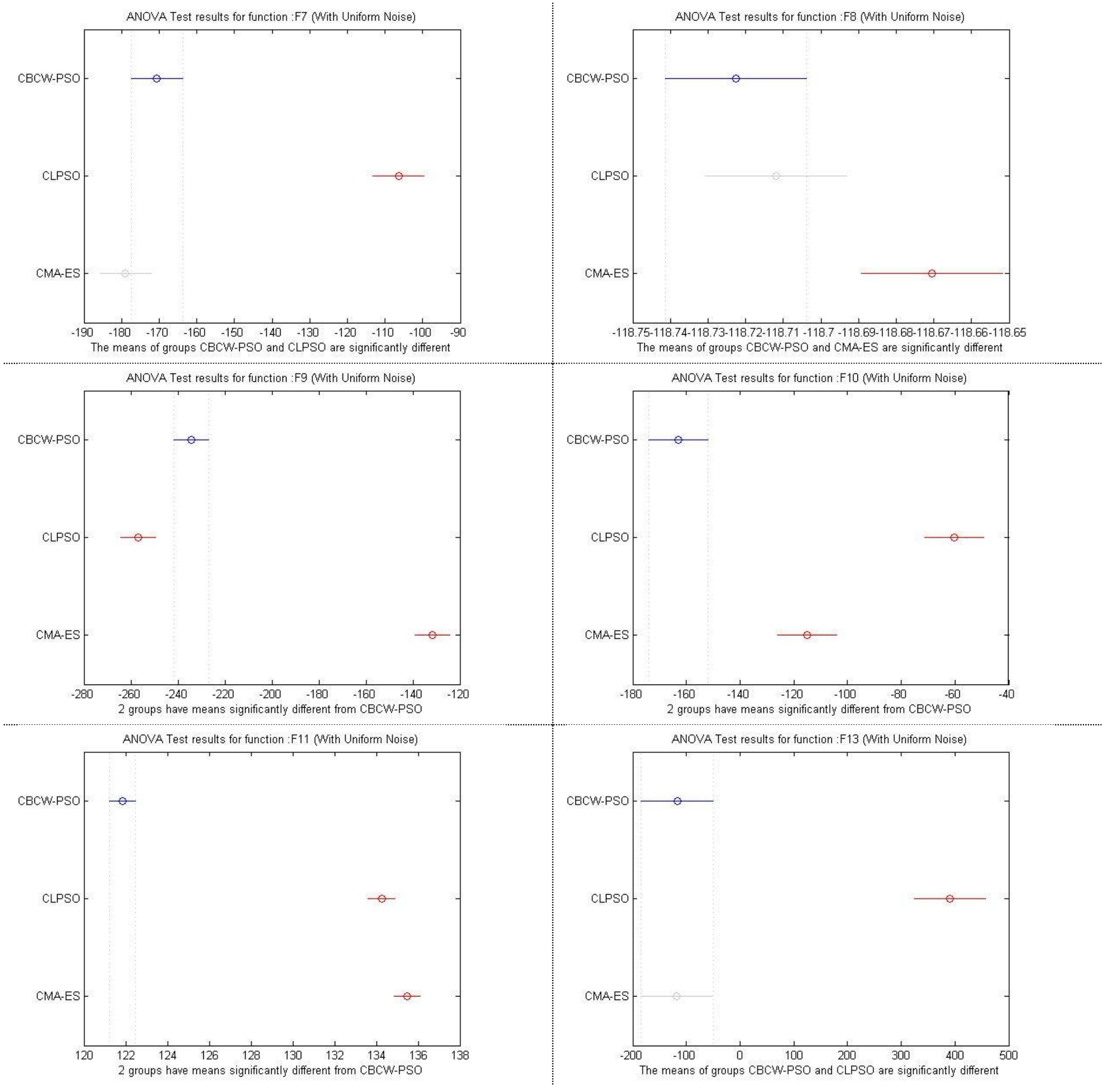


Figure A.4 : The ANOVA Test results for various benchmark functions with noisy fitness value (Gaussian noise type).



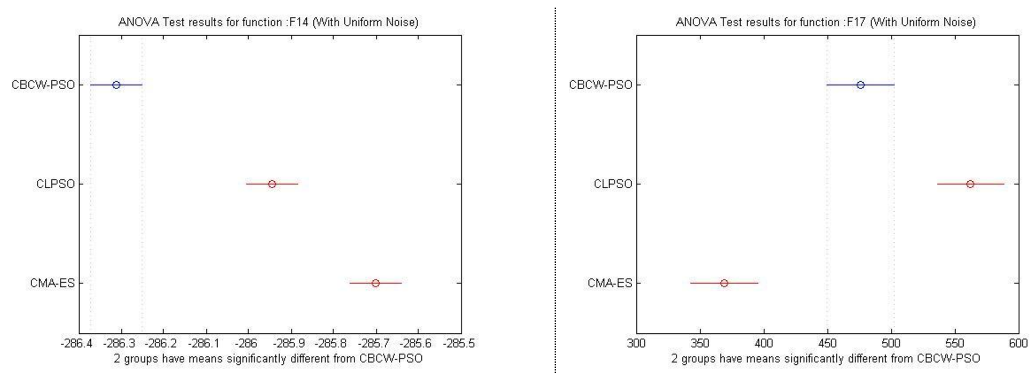
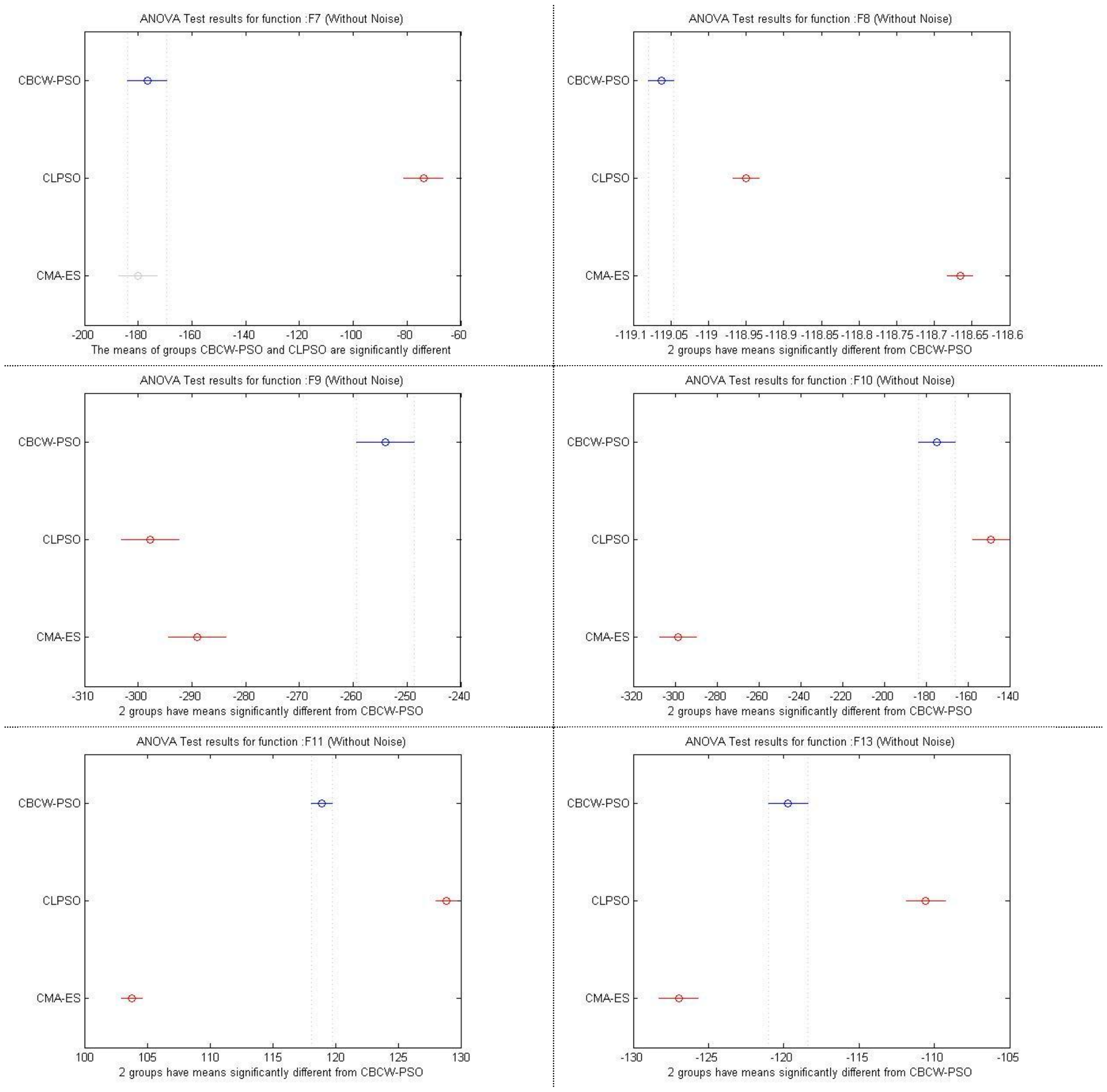


Figure A.5 : The ANOVA Test results for various benchmark functions with noisy fitness value (Uniform noise type).



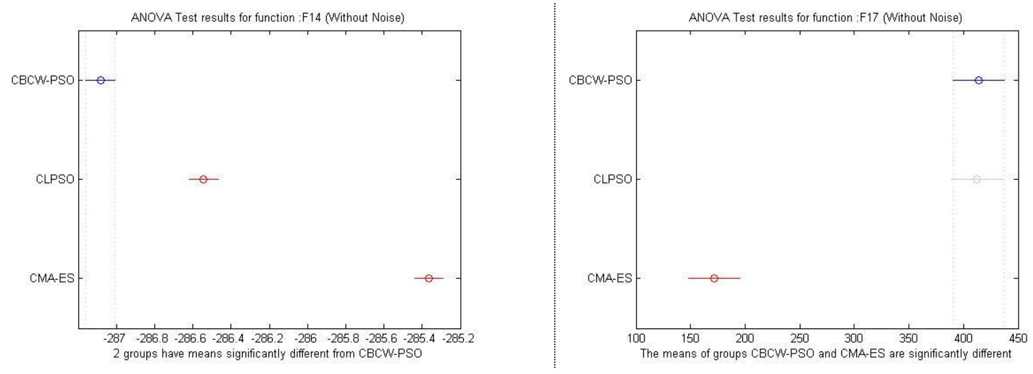


Figure A.6 : The ANOVA Test results for various benchmark functions with non-noisy fitness value.

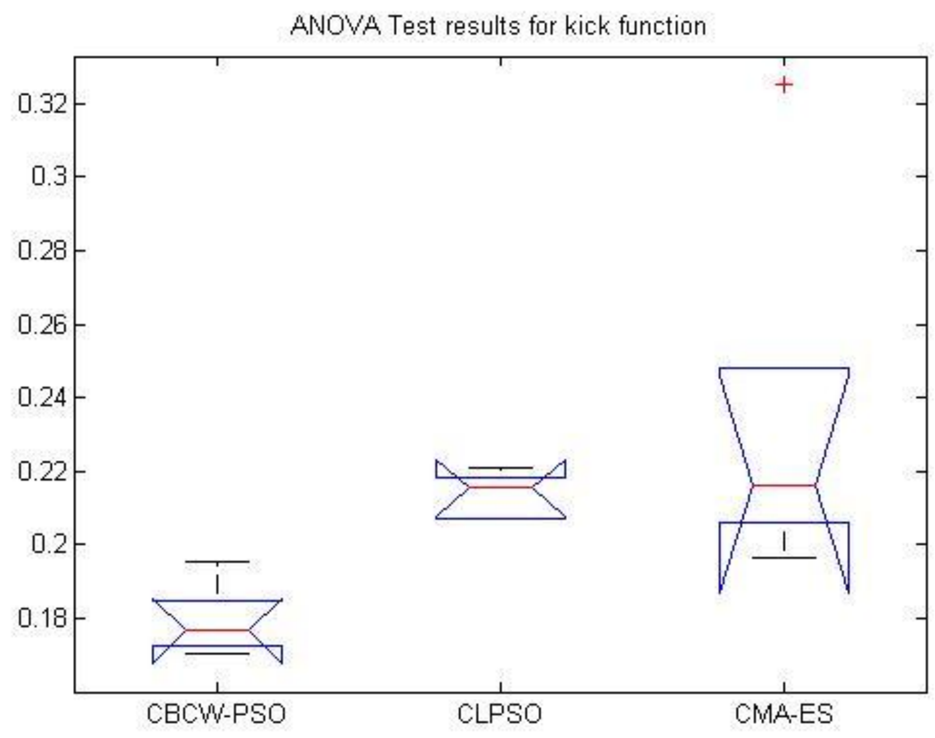


Figure A.7 : The ANOVA Test results for the robot kick function.

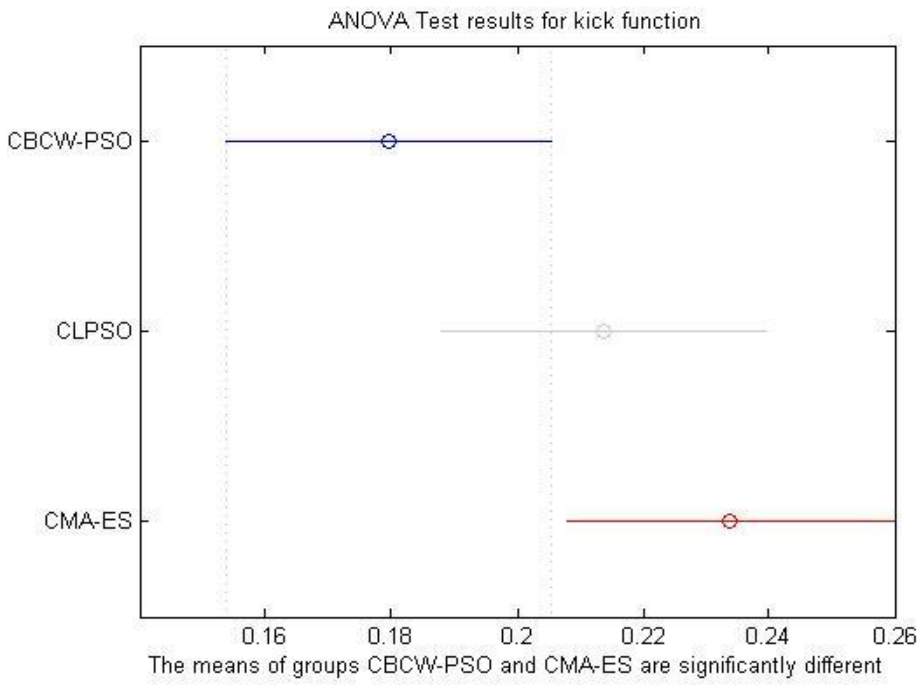


Figure A.8 : The ANOVA Test results for the robot kick function.

CURRICULUM VITAE

Name Surname: Shahriar ASTA

Place and Date of Birth: Oroumieh IRAN , 11 June 1980

Address: İTÜ Ayazağa campus, Institute of Science and Technology

E-Mail: shahriarasta@gmail.com

B.Sc.: University of Isfahan , IRAN

M.Sc. (If exists):

Professional Experience and Rewards:

List of Publications and Patents:

PUBLICATIONS/PRESENTATIONS ON THE THESIS

- **S.Asta, Ş.E.Uyar.** (2011), “A Novel Particle Swarm Optimization”, In proceedings of Artificial Evolution, 24-26 October 2011, Angers, France.