Hybrid PSO-SA Type Algorithms for Multimodal Function Optimization and Reducing Energy Consumption in Embedded Systems

Lhassane Idoumghar, Mahmoud Melkemi, René Schott, Maha Idrissi Aouad

Hindawi Publishing Corporation
Applied Computational Intelligence and Soft Computing
Volume 2011, Article ID 138078, 12 pages
doi:10.1155/2011/138078

Research Article

Hybrid PSO-SA Type Algorithms for Multimodal Function Optimization and Reducing Energy Consumption in Embedded Systems

Lhassane Idoumghar,[1,2] Mahmoud Melkemi,[2] René Schott,[3] and Maha Idrissi Aouad[1]

[1] *INRIA Nancy—Grand Est/LORIA, 615 Rue du Jardin Botanique, 54600 Villers-Lès-Nancy, France*
[2] *LMIA—MAGE, Université de Haute-Alsace, 4 Rue des Frères Lumière, 68093 Mulhouse, France*
[3] *IECN—LORIA, Nancy-Université, Université Henri Poincaré, 54506 Vandoeuvre-Lès-Nancy, France*

Correspondence should be addressed to Lhassane Idoumghar, lhassane.idoumghar@uha.fr

Received 31 December 2010; Revised 22 March 2011; Accepted 11 April 2011

Academic Editor: Chuan-Kang Ting

The paper presents a novel hybrid evolutionary algorithm that combines Particle Swarm Optimization (PSO) and Simulated Annealing (SA) algorithms. When a local optimal solution is reached with PSO, all particles gather around it, and escaping from this local optima becomes difficult. To avoid premature convergence of PSO, we present a new hybrid evolutionary algorithm, called HPSO-SA, based on the idea that PSO ensures fast convergence, while SA brings the search out of local optima because of its strong local-search ability. The proposed HPSO-SA algorithm is validated on ten standard benchmark multimodal functions for which we obtained significant improvements. The results are compared with these obtained by existing hybrid PSO-SA algorithms. In this paper, we provide also two versions of HPSO-SA (sequential and distributed) for minimizing the energy consumption in embedded systems memories. The two versions, of HPSO-SA, reduce the energy consumption in memories from 76% up to 98% as compared to Tabu Search (TS). Moreover, the distributed version of HPSO-SA provides execution time saving of about 73% up to 84% on a cluster of 4 PCs.

1. Introduction

Several optimization algorithms have been developed over last few decades for solving real-world optimization problems. Among them, we have many heuristics like Simulated Annealing (SA) [1] and optimization algorithms that make use of social or evolutionary behaviors like Particle Swarm Optimization (PSO) [2, 3]. SA and PSO are quite popular heuristics for solving complex optimization problems, but they have some strengths and limitations.

Particle Swarm Optimization (PSO) is based on the social behavior of individuals living together in groups. Each individual tries to improve itself by observing other group members and imitating the better ones. This way, the group members are performing an optimization procedure which is described in [3]. The performance of the algorithm depends on how the particles (i.e., potential solutions to an optimization problem) move in the search space, given that the velocity is updated iteratively. Large research body is therefore devoted to the analysis and proposal of different motion rules (see [4–6] for recent accounts of PSO research). To avoid premature convergence of PSO, we combine it with SA: PSO contributes to the hybrid approach in a way to ensure that the search converges faster, while SA makes the search jump out of local optima due to its strong local-search ability. In this paper, we present a hybrid optimization algorithm, called HPSO-SA, which exploits intuitively the positive features of PSO and SA. We also validate HPSO-SA using ten benchmark functions given in [7] and compare the results with classical PSO, ATREPSO, QIPSO, and GMPSO algorithms described in [2], TL-PSO [8], PSO-SA [9], SAPSO and SUPER-SAPSO presented in [10]. We provide also two versions of HPSO-SA (sequential and distributed) for minimizing the energy consumption in

embedded systems memories. The two versions, of HPSO-SA, reduce the energy consumption in memories from 76% up to 98% as compared to Tabu Search (TS). Moreover, the distributed version of HPSO-SA provides execution time saving of about 73% up to 84% on a cluster of 4 PCs.

The rest of the paper is organized as follows. Section 2 introduces, briefly, PSO and SA algorithms. Section 3 is devoted to detailed description of HPSO-SA. In Section 4, benchmark functions are applied on HPSO-SA. In Section 5, HPSO-SA is used to solve the energy consumption problem in memory. In addition, simulation results are provided and compared with those of [11]. Conclusions and further research aspects are given in Section 6.

2. Background

2.1. Simulated Annealing Algorithm. SA [12] is a probabilistic variant of the local search method, which can, in contrast to PSO, escape local optima. SA is based on an analogy taken from thermodynamics which is as follows: In order to grow a crystal, we start by heating material until it reaches its molten state. Then, we reduce the temperature of this crystal melt gradually, until the crystal structure is formed. A standard SA procedure begins by generating an initial solution at random. At initial stages, a small random change is made in the current solution s_c. Then the objective function value of the new solution s_n is calculated and compared with that of the current solution. A move is made to the new solution if it has better value or if the probability function implemented in SA has a higher value than a randomly generated number. Otherwise a new solution is generated and evaluated. The probability of accepting a new solution is given as follows:

$$p = \begin{cases} 1, & \text{if } f(s_n) - f(s_c) < 0, \\ \exp\left(\dfrac{-|f(s_n) - f(s_c)|}{T}\right), & \text{otherwise.} \end{cases} \quad (1)$$

The calculation of this probability relies on a parameter T, which is referred to as temperature, since it plays a similar role as the temperature in the physical annealing process. To avoid getting trapped at a local minimum point, the rate of reduction should be slow. In our problem we use the following method to reduce the temperature: $\gamma = 0.99$ and

$$T_{i+1} = \gamma T_i, \quad (2)$$

where $i = 0, 1, \ldots$.

Thus, at the start of SA most worsening moves may be accepted, but in the end only improving ones are likely to be allowed, which can help the procedure jump out of a local minimum. The algorithm may be terminated after a certain volume fraction of the structure has been reached or after a prespecified runtime.

2.2. Particle Swarm Optimization. PSO is a population based stochastic optimization technique developed by [13], inspired by social behavior patterns of organisms that live and interact within large groups. In particular, it incorporates swarming behaviors observed in flocks of birds, schools of fish, or swarms of bees, and even human social behavior.

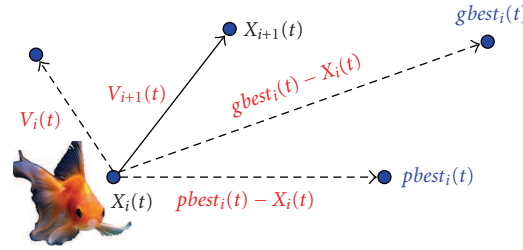

Figure 1: Movement of each particle.

PSO algorithm is based on an idea that particles move through the search space with velocities that are dynamically adjusted according to their historical behaviors. Therefore, the particles have the tendency to move towards the better and better search area over the course of search process. PSO algorithm starts with a group of random (or not) particles (solutions) and then searches for optima by updating each generation. Each particle is treated as a volume-less particle (a point) in the n-dimensional search space. The ith particle is represented as $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$. At each generation, the particles are updated by using following two *best* values.

(i) The first value is the best solution (fitness) a particle has achieved so far (the fitness value is also stored). This value is called *pbest*.

(ii) The second value is the best value tracked by the particle swarm optimizer so far (by any particle) in the population. This best value is a global best and is called *gbest*. When a particle takes part of a population as its topological neighbors, the best value is a local best and is called *lbest*.

At each iteration, these two best values are combined to adjust the velocity along each dimension, which is then used to compute a new iteration step for the particle. A portion of adjustment to the velocity is influenced by the individual's previous best position (*pbest*), considered as the *cognition* component, and this portion is influenced by the best in the neighborhood (*lbest* or *gbest*), the social component (see Figure 1). With the addition of the inertia factor, ω, by [14] (for balancing the global and the local search), the equations for velocity adjustment are

$$\begin{aligned} v_{i+1} &= \omega v_i + c_1 * \text{random}(0,1) * (pbest_i - x_i) \\ &\quad + c_2 * \text{random}(0,1) * (gbest_i - x_i), \end{aligned} \quad (3)$$

$$x_{i+1} = x_i + v_{i+1}, \quad (4)$$

where random$(0,1)$ is a random number independently generated within the range of $[0,1]$ and c_1 and c_2 are two learning factors which control the influence of the social and cognitive components (usually, $c_1 = c_2 = 2$, see [15]).

In (3), if the sum on the right side exceeds a constant value, then the velocity on that dimension is assigned to be V_i min or V_i max. Thus, particle velocities are clamped to the range of $[V_i \text{ min}; V_i \text{ max}]$, which serves as a constraint to control the global exploration ability of PSO algorithm.

This also reduces the likelihood of particles leaving the search space. Note that the values of x_i are not restricted to the range $[V_i \min; V_i \max]$; it only limits the maximum distance that a particle will move during one iteration.

3. HPSO-SA Hybrid Algorithm

This section presents a new hybrid HPSO-SA algorithm which combines the advantages of both PSO (that has a strong global-search ability) and SA (that has a strong local-search ability). Other applications of hybrid PSO and SA algorithm can be found [9, 10, 16–19].

This hybrid approach makes full use of the exploration capability of both PSO and SA and offsets the weaknesses of each. Consequently, through application of SA to PSO, the proposed algorithm is capable of escaping from a local optimum. However, if SA is applied to PSO at each iteration, the computational cost will increase sharply and at the same time the fast convergence ability of PSO may be weakened. In order to flexibly integrate PSO with SA, SA is applied to PSO every K iterations if no improvement of the global best solution does occur. Therefore, the hybrid HPSO-SA approach is able to keep fast convergence (most of the time) thanks to PSO, and to escape from a local optimum with the aid of SA. In order to allow PSO jump out of a local optimum, SA is applied to the best solution in the swarm found so far, each K iterations that is predefined to be $270 \sim 500$ (based on our experimentations).

The hybrid HPSO-SA algorithm works as illustrated in Algorithm 1, where one has the following.

(i) *Description of a Particle*. Each particle (solution) $X \in S$ is represented by its $n > 0$ components, that is, $X = (x_1, x_2, \ldots, x_n)$, where $i = 1, 2, \ldots, n$ and n represents the dimension of the optimization problem to solve.

(ii) *Initial Swarm*. Initial Swarm corresponds to population of particles that will evolve. Each particle x_i is initialized with uniform random value between the lower and upper boundaries of the interval defining the optimization problem.

(iii) *Evaluate Function*. Evaluate (or fitness) function in HPSO-SA algorithm is typically the objective function that we want to minimize in the problem. It serves for each solution to be tested for suitability to the environment under consideration.

(iv) *SA Algorithm*. If no improvement of the global best solution occur during the last K iterations, then it means that the algorithm is trapped in a local optimum point. To escape out from local optimum, we apply SA algorithm to global best solution. The performance of SA depends on the definition of the several control parameters.

(a) *Initial Temperature*. Kirkpatrick [20] suggested that a suitable initial temperature is one that results in an average probability χ_0 of a solution that increases f being accepted of about 0.8. The value of T_0 will clearly depend on the scaling of f and, hence, be problem-specific. It can be estimated by conducting an initial search (100 iterations in next simulations) in which all increases in f are accepted and calculating the average objective increase observed δf. T_0 is then given by

$$T_0 = -\frac{\delta f}{\ln(\chi_0)}. \tag{5}$$

(b) *Accept Function*. Function *Accept* (*current_solution, Neighbor, T*) is decided by the acceptance probability given by (1), which is the probability of accepting configuration *Neighbor*.

(c) *Generate Function*. The neighborhood of each solution x is generated by using the following Equation:

$$x \longleftarrow x + d\sigma r, \tag{6}$$

where d is the direction of the new neighborhood and takes either 1 or -1, σ is random number with Gaussian (0,1) distribution and r is a constant that correspond to the radius of neighborhood generator.

(d) *SA_Stop_Criterion*. The stopping criterion of SA algorithm defines when the system has reached *3000 function evaluations* or *maximum number of functions evaluations* or *Optimal_solution* are not attained.

(e) *Decrementing the Temperature*. The most commonly used temperature reducing function is geometric (see (2)). In next simulations $\gamma = 0.99$.

(f) *Inner-Loop*. The length of each temperature level determines the number $q = 150$ of solutions generated at each temperature, T.

4. Experiments Results

4.1. Benchmark Functions. In order to compare the performance of HPSO-SA hybrid algorithm with those described in [2, 8–10], we use benchmark functions [7] described in Table 1. These functions provide a good starting point for testing the credibility of an optimization algorithm. For each of these functions, there are many local optima in their solution spaces. The number of local optima increases with increasing complexity of the functions, that is, with increasing dimension. In the following experiments, we used 10-, 20- and 30-dimensional functions except in the case of Himmelblau and Shubert functions that are two-dimensional by definition (see Figure 2 for 3D representation).

4.2. Simulation Results and Discussions. To verify the efficiency and effectiveness of HPSO-SA hybrid algorithm, the experimental results of HPSO-SA approach are compared with those obtained by [2, 8–10]. Our HPSO-SA hybrid

```
(1) iter ←0, cpt ← 0, Initialize swarm_size particles
(2) stop_criterion←maximum number of function evaluations or Optimal_solution is not attained
(3) while Not stop_criterion do
(4)     for each particle i ← 1 to swarm_size do
(5)         Evaluate(particle(i)) if the fitness value is better than the best fitness value (cbest) in history then
(6)             Update current value as the new cbest.
(7)         end
(8)     end
(9)     Choose the particle with the best fitness value in the neighborhood (gbest)
(10)    for each particle i ← 1 to swarm_size do
(11)        Update particle velocity according to Equation (3)
(12)        Enforce velocity bounds
(13)        Update particle position according to Equation (4)
(14)        Enforce particle bounds
(15)    end
(16)    if there is no improvement of global best solution then
(17)        cpt ← cpt + 1
(18)    end
(19)    Update global best solution
(20)    cpt ← 0
(21)    if cpt = K then
(22)        cpt ← 0
(23)        //Apply SA to global best solution
(24)        iterSA ← 0, Initialize T according to Equation (5)
(25)        current_solution ← global_best_solution
(26)        current_cost ←Evaluate(current_solution)
(27)        while Not SA_stop_criterion do
(28)            while inner-loop stop criterion do
(29)                Neighbor ←Generate(current_solution)
(30)                Neighbor_cost ← Evaluate(Neighbor)
(31)                if Accept(current_cost, Neighbor_cost, T) then
(32)                    current_solution ← Neighbor
(33)                    current_cost ← Neighbor_cost
(34)                end
(35)                iterSA ← iterSA + 1
(36)                Update (global_best_solution)
(37)            end
(38)            Update(T) according to Equation(2)
(39)            Update (SA_stop_criterion)
(40)        end
(41)    end
(42)    iter ← iter + 1, Update (stop_criterion)
(43) end
```

Algorithm 1: HPSO-SA hybrid algorithm.

algorithm is written in C++ and was compiled using *gcc* version 2.95.2 (Dev-cpp) on a laptop with Windows Vista x64 Premium Home Edition running Intel Core 2 Quad (Q9000) at 2 GHz and having 4 Gb memory.

4.2.1. Comparison with Results Obtained by Using TL-PSO Algorithm [8]. In this section we compare HPSO-SA approach with TL-PSO method [8] that is based on combining the excellence of both PSO and Tabu Search. As described in [8], we apply HPSO-SA algorithm to the four following benchmark problems: Rastrigin, Schwefel, Griewank and Rosenbrock function. Here, the number of particles in the swarm is 30. The number of dimension of searching $n = 20$ and the number of objective function evaluations is 60000 (i.e., 2000×30). The results obtained after numerical simulations are shown in Table 2. These results indicate the Mean, Best, and Worst values obtained under the same condition over 50 trials. By analyzing Table 2, we conclude that the results obtained by HPSO-SA algorithm are preferable in comparison with those obtained by TL-PSO algorithm.

4.2.2. Comparison with Other PSO Algorithms Described in [2]. Performance of four Particle Swarm Optimization algorithms, namely classical PSO, Attraction-Repulsion based PSO (*ATREPSO*), Quadratic Interpolation based PSO

Table 1: Standard benchmark functions adopted in this work.

Function	Problem	Range	$f(x^*)$	Classification
Sphere	$\sum_{i=1}^{n} x_i^2$	$[-100; 100]$	0	Unimodal
Rastrigin	$\sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i) + 10)$	$[-5.12; 5.12]$	0	Multimodal
Griewank	$(1/4000)\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(x_i/\sqrt{i}) + 1$	$[-600; 600]$	0	Multimodal
Rosenbrock	$\sum_{i=1}^{n-1}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	$[-2.048; 2.048]$	0	Unimodal
Quartic	$(\sum_{i=1}^{n} ix_i^4) + \text{rand}[0, 1]$	$[-1.28; 1.28]$	0	Noisy
Schwefel	$420.9687n - \sum_{i=1}^{n}(x_i \sin(\sqrt{\lvert x_i\rvert}))$	$[-500.0; 500.0]$	0	Multimodal
Ackley	$20 + e - 20e^{-0.2((1/n)\sum_{i=1}^{n} x_i^2)^{1/2}} - e^{(1/n)\sum_{i=1}^{n}\cos(2\pi x_i)}$	$[-30.0; 30.0]$	0	Multimodal
Michalewicz	$-\sum_{i=1}^{n}\sin(x_i)\sin^{2m}((i - x_i^2)/\pi)$	$[-\pi; \pi]$	—	Multimodal
Himmelblau	$(x_2 + x_1^2 - 11)^2 + (x_1 + x_2^2 - 7)^2 + x_1$	$[-5.0; 5.0]$	-3.78396	Multimodal
Shubert	$\sum_{i=1}^{5} i\cos((i + 1)x_1 + i)\sum_{i=1}^{5} i\cos((i + 1)x_2 + i)$	$[-10.0; 100]$	-186.7309	Multimodal

Table 2: Comparison of hybrid HPSO-SA algorithm with TL-PSO approach [8].

	Mean		Best		Worst	
Function	TL-PSO	**HPSO-SA**	TL-PSO	**HPSO-SA**	TL-PSO	**HPSO-SA**
Rastrigin	8.7161	**5.23034e − 05**	4.0589	**4.35994e − 09**	17.053	**8.15835e − 04**
Schwefel	445.8830	**206.693255**	0.6748	**0**	829.4431	**355.318**
Griewank	0.0228	**1.18294e − 03**	$1.0507e - 5$	**1.2299e − 15**	0.0739	**0.0172263**
Rosenbrock	19.5580	**0.58359742**	0.1331	**0.0442625**	71.5439	**1.36171**

Table 3: Comparison of mean/standard deviation of solutions obtained by using hybrid HPSO-SA algorithm and approaches described in [2].

Function	*PSO*	*QIPSO*	*ATREPSO*	*GMPSO*	**HPSO-SA**
Rastrigin	22.339158	11.946888	19.425979	20.079185	**0**
	15.932042	9.161526	14.349046	13.700202	**0**
Sphere	$1.167749e - 45$	0.000000	$4,000289e - 17$	$7,263579e - 17$	**5.3656e − 32**
	$5.222331e - 46$	0.000000	0.000246	$6.188854e - 17$	**2.98492e − 31**
Griewank	0.031646	0.011580	0.025158	0.024462	**3,32255e − 20**
	0.025322	0.012850	0.028140	0.039304	**2.68415e − 20**
Rosenbrock	22.191725	8.939011	19.490820	14.159547	**0,227048188**
	$1.615544e + 04$	3.106359	$3.964335e + 04$	$4.335439e + 04$	**0.243978057**
Noisy	8.681602	0.451109	8.046617	7.160675	**0,002019998**
	9.001534	0.328623	8.862385	7.665802	**0.000650347**
Schwefel	−6178.559896	−6355.586640	−6183.677600	−6047.670898	**−8379.66**
	$4.893329e + 02$	477.532584	469.611104	482.926738	**2.20425e − 19**
Ackley	$3.483903e - 18$	$2.461811e - 24$	0.018493	$1.474933e - 18$	**7.43546e − 16**
	$8.359535e - 19$	0.014425	0.014747	$1.153709e - 08$	**1.09382e − 15**
Michalewizc	−18.159400	−18.469600	−18.982900	−18.399800	**−19,62555806**
	1.051050	0.092966	0.272579	0.403722	**0.00926944**
Himmelblau	−3.331488	−3.783961	−3.751458	−3.460233	**−3,78396**
	1.243290	0.190394	0.174460	0.457820	**3.16001e − 15**
Shubert	−186.730941	−186.730942	−186.730941	−186.730942	**−186,730942**
	$1.424154e - 05$	0.000000	$1.424154e - 05$	$1.525879e - 05$	**8.66746e − 14**

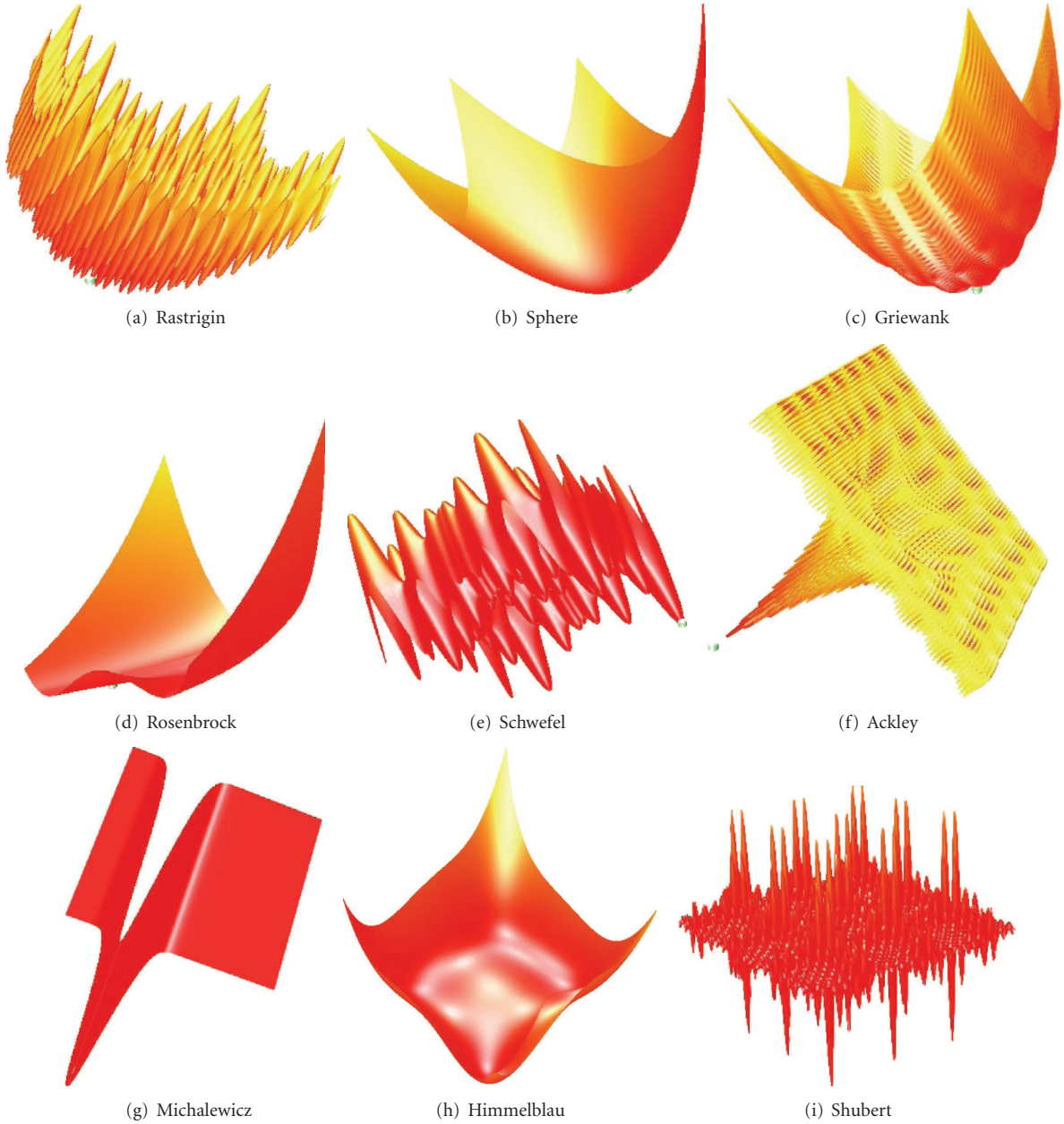

Figure 2: 3D mathematical benchmark functions.

(*QIPSO*) and Gaussian Mutation based PSO (*GMPSO*) is evaluated in [2]. The algorithms presented in this paper are guided by the diversity of the population to search the global optimal solution of a given optimization problem, where as GMPSO uses the concept of mutation and QIPSO uses the reproduction operator to generate a new member of the swarm.

In order to make a fair comparison between classical PSO, ATREPSO, QIPSO, GMPSO and HPSO-SA approach, we fixed, as indicated in [2], the same seed for random number generation so that the initial swarm population is same for all five algorithms. The number of particles in the swarm is 30. The algorithms use a linearly decreasing inertia weight ω which starts at 0.9 and ends at 0.4, with the user defined parameters $c_1 = c_2 = 2.0$. For each algorithm, the number of objective function evaluations is 300000. A total of 30 runs for each experimental setting were conducted and the average fitness of the best solutions throughout the run was recorded. The mean solution and the standard deviation (note that the standard deviation indicates the stability of the algorithms), found by the five algorithms, is listed in Table 3. The numerical results given in Table 3 show the following.

(i) All the algorithms outperform the classical Particle Swarm Optimization.

(ii) HPSO-SA algorithm gives much better performances in comparison to PSO, QIPSO, ATREPSO, and GMPSO, out of the Sphere's and Ackley's functions.

(iii) On Sphere's function, QIPSO obtains better results than those obtained by HPSO-SA approach. But when the maximum number of iterations is fixed to 1.5×10^6, HPSO-SA obtains the optimal value.

(iv) The analysis of the results, obtained for Ackley's function, shows that QIPSO obtains better mean result than HPSO-SA algorithm. However, HPSO-SA has a much smaller standard deviation.

4.2.3. Comparison with Other PSO Algorithms Described in [10]. In this section four benchmark functions are used to compare the relative performance of HPSO-SA algorithm with SUPER-SAPSO, SAPSO, and PSO algorithms described in [10].

For all comparisons, the number of particles was set to 30. HPSO-SA algorithm uses a linearly decreasing inertia weight ω which starts at 0.9 and ends at 0.4, with the user defined parameters $c_1 = c_2 = 2.0$, 20 runs are conducted for each experimental setting and, for each algorithm, the average value is given in Table 4.

In all above experiments, HPSO-SA algorithm obtains better results in comparison to those obtained by both the standard PSO and SAPSO algorithm [10]. A comparison of HPSO-SA algorithm and SUPER-SAPSO [10], shows that the last one converges faster than HPSO-SA.

SUPER-SAPSO uses an expression for the particle movements ($x_{t+1} = (x_t + v_{t+1})T$ where $T \ll 1$) which is well-adapted to the case where the global optimum is 0. This is the reason why SUPER-SAPSO needs a very small number of iterations in this case.

4.2.4. Comparison with PSO-SA Algorithm Described in [9]. In this section performances of HPSO-SA are compared with these of PSOSA [9], Genetic Algorithm and hybrid algorithm [21].

Table 5 lists different results obtained for three different dimensions of each function. The optimum value of Sphere, Rastrigrin and Griewank was set to be $1e-10$ and the goal value of Rosenbrock function was set to be $1e-06$ (as indicated in [9]).

To make a fair comparison, the maximum number of function evaluations allowed was set to 20000, 30000 and 40000 for HPSO-SA and PSOSA algorithms when the number of particle was set to 20. HPSO-SA algorithm uses a linearly decreasing inertia weight ω which starts at 0.9 and ends at 0.4, with the user defined parameters $c_1 = c_2 = 2.0$.

The numerical results given in Table 5 show that:

(i) Over four benchmark functions, HPSO-SA and PSOSA do better than standard GA and hybrid algorithm [21].

(ii) For Sphere, Rastrigin and Griewank functions, HPSO-SA and PSOSA algorithms obtain optimal solutions within specified constrains (number of objective function evaluations).

(iii) For Rosenbrock function, PSOSA obtains better results than HPSO-SA for dimension 20, but for dimensions 10 and 30, HPSO-SA does better and has smaller standard deviation.

5. Reducing Memory Energy Consumption in Embedded Systems

5.1. Description of the Memory Problem. According to trends in [22], memory will become the major energy consumer in an embedded system. Indeed, embedded systems must integrate multiple complex functionalities which needs bigger battery and memory. Hence, reducing memory energy consumption of these systems has never been as topical. In this paper, we will focus on software techniques for the memory management. In order to reduce memory energy consumption, most authors rely on Scratch-Pad Memories (*SPMs*) rather than caches [23]. Although cache memory helps a lot with program speed, it is not the appropriate for most of the embedded systems. In fact, cache increases the system size and its energy cost (cache area plus managing logic). Like cache, SPM consists of small, fast SRAM. The main difference is that SPM is directly and explicitly managed at the software level, either by the developer or by the compiler which makes it more predictable. SPM requires up to 40% less energy and 34% less area than cache [24]. In this paper, we will therefore use an SPM in our memory architecture. Due to the reduced SPM size, we allocate space for interesting data only whereas, the remaining is placed in main memory (*DRAM*). In order to determine interesting data, we use *data profiling* to gather memory access frequency information. The Tabu Search (TS) approach consists of allocating space for data in SPM based on TS principles [25]. More details about how TS is implemented can be found in [11].

In order to compute energy cost of the system, we propose an energy consumption estimation model, for our memory architecture composed by an SPM, an instruction cache and a DRAM. Equation (7) gives the energy model where the three terms refer to the total energy consumed, respectively, in SPM, in instruction cache and in DRAM.

$$E = E_{tspm} + E_{tic} + E_{tdram}. \tag{7}$$

In this model, we distinguish between the two cache write policies: Write-Through (*WT*) and Write-Back (*WB*). In a WT cache, every write to the cache causes a synchronous write to DRAM. Alternatively, in a WB cache, writes are not immediately mirrored to DRAM. Instead, the cache tracks which of its locations have been written over and then, it marks these locations as dirty. The data in these locations is written back to DRAM when that data is evicted from the cache [26]. In this paper, the aim is to minimize the energy for the detailed estimation model presented as follows:

$$E = N_{spmr} * E_{spmr} \tag{8}$$

$$+ N_{spmw} * E_{spmw} \tag{9}$$

Table 4: Performance results of HPSO-SA, SUPER-SAPSO, SAPSO, and PSO algorithms on benchmark functions.

Function	Algorithm	Number of iterations	Average Error
Rastrigin	PSO	2989	0.097814
	SAPSO	2168	0.07877
	SUPER-SAPSO	5	0.0
	HPSO-SA	2022.43	0.0
Sphere	PSO	805	0.094367
	SAPSO	503	0.085026
	SUPER-SAPSO	4	0.0
	HPSO-SA	501.15	$7,92e-32$
Griewank	PSO	2004	0.082172
	SAPSO	1517	0.075321
	SUPER-SAPSO	3	0.0
	HPSO-SA	1496.35	$3,61e-20$
Ackley	PSO	4909	0.099742
	SAPSO	3041	0.099461
	SUPER-SAPSO	5	0.0
	HPSO-SA	3020.65	$8,92e-16$

Table 5: Performance comparison between PSOSA, Sid.GA, Hybrid and HPSO-SA for benchmark functions [9].

Function	*Dimension*	*PSOSA*	*Std.GA*	*Hybrid*	**HPSO-SA**
Sphere	10	0 ± 0	$2.43e-04 \pm 1.14e-05$	$2.42e-04 \pm 2.17e-05$	0 ± 0
	20	0 ± 0	$0.00145 \pm 6.22e-05$	$0.00212 \pm 2.75e-04$	0 ± 0
	30	0 ± 0	$0.00442 \pm 1.78e-04$	$0.01203 \pm 6.33e-04$	0 ± 0
Rastrigin	10	0 ± 0	3.1667 ± 0.2237	3.0599 ± 0.1535	0 ± 0
	20	0 ± 0	16.8732 ± 0.6007	11.6590 ± 0.3602	0 ± 0
	20	0 ± 0	49.3212 ± 1.1204	27.8119 ± 0.8059	0 ± 0
Griewank	10	0 ± 0	283.251 ± 1.812	0.09078 ± 0.03306	0 ± 0
	20	0 ± 0	611.266 ± 3.572	0.00459 ± 0.01209	0 ± 0
	30	0 ± 0	889.537 ± 3.939	0.09911 ± 0.00106	0 ± 0
Rosenbrock	10	0.17856 ± 1.25988	109.810 ± 6.212	43.521 ± 16.047	0.146 ± 0.224193
	20	0.00043 ± 0.00111	146.912 ± 10.951	169.112 ± 21.535	0.246897 ± 0.231982
	30	0.57431 ± 4.05976	199.730 ± 16.285	187.033 ± 22.960	0.439149 ± 0.304347

$$+\sum_{k=1}^{N_{icr}}[h_{i_k} * E_{icr} + (1-h_{i_k}) * [E_{dramr} + E_{icw} + (1-WP_i) * DB_{i_k} * (E_{icr} + E_{dramw})]] \quad (10)$$

$$+\sum_{k=1}^{N_{icw}}[WP_i * E_{dramw} + h_{i_k} * E_{icw} + (1-WP_i) * (1-h_{i_k}) * [E_{icw} + DB_{i_k} * (E_{icr} + E_{dramw})]] \quad (11)$$

$$+ N_{dramr} * E_{dramr} \quad (12)$$

$$+ N_{dramw} * E_{dramw}. \quad (13)$$

Equations (8) and (9) represent, respectively, the total energy consumed during reading and writing from/to SPM. Equations (10) and (11) represent, respectively, the total energy consumed during reading and writing from/to instruction cache. When, Equations (12) and (13) represent, respectively, the total energy consumed during reading and writing from/to DRAM. The various terms used in this energy model are explained in Table 6.

As SPM has got a lot of advantages, it is clearly preferable to put as much data as possible in it. In other words, we must maximize terms N_{spmr} and N_{spmw} in the model. Hence, the problem becomes to maximize the number of accesses to the SPM. It is therefore a combinatorial optimization problem like knapsack problem [27]. We want to fill SPM that can hold a maximum capacity of C with some combination of data from a list of N possible data each with $size_i$ and $access\ number_i$ so that the access number of the data allocated into SPM is maximized. This problem has a single linear constraint, a linear objective function which sums

Table 6: List of terms.

Term	Meaning
E_{spmr}	Energy consumed during a reading from SPM.
E_{spmw}	Energy consumed during a writing into SPM.
N_{spmr}	Reading access number to SPM.
N_{spmw}	Writing access number to SPM.
E_{icr}	Energy consumed during a reading from instruction cache.
E_{icw}	Energy consumed during a writing into instruction cache.
N_{icr}	Reading access number to instruction cache.
N_{icw}	Writing access number to instruction cache.
E_{dramr}	Energy consumed during a reading from DRAM.
E_{dramw}	Energy consumed during a writing into DRAM.
N_{dramr}	Reading access number to DRAM.
N_{dramw}	Writing access number to DRAM.
WP_i	The considered cache write policy: WT or WB. In case of WT, $WP_i = 1$ else in case of WB then $WP_i = 0$.
DB_{i_k}	Dirty Bit used in case of WB to indicate during the access k if the instruction cache line has been modified before ($DB_i = 1$) or not ($DB_i = 0$).
h_{i_k}	Type of the access k to the instruction cache. In case of cache hit, $h_{i_k} = 1$. In case of cache miss, $h_{i_k} = 0$.

the sizes of the data allocated into SPM, and the added restriction that each data will be in the SPM or not. If N is the total number of data, then a solution is just a finite sequence s of N terms such that $s[n]$ is either 0 or the size of the n_{th} data. $s[n] = 0$ if and only if the n_{th} data is not selected in the solution. This solution must satisfy the constraint of not exceeding the maximum SPM capacity (i.e., $\sum_{i=1}^{N} s[i] \leq C$).

5.2. Discrete Sequential Hybrid HPSO-SA Algorithm. This section should be considered as an attempt to use hybrid evolutionary algorithms for reducing energy consumption in embedded systems. Here, the focus is on the use of HPSO-SA algorithm designed in previous sections. Since the problem under consideration is dicrete and has specific features, HPSO-SA needs changes.

solution (Particle). A solution can be represented by an array having size equal to the number of the data. Each element from this array denotes whether a data is included in the SPM ("1") or not ("0"). The HPSO-SA algorithm starts with an initial swarm which is randomly initialized.

Evaluate Function. It is the objective function that we want to minimize in the problem. It serves for each solution to be tested for suitability to the environment under consideration

$$Evaluate(solution) = Total\_Number\_Access\_all\_data - Number\_Access(solution). \quad (14)$$

Position Update Equation. Each dimension j of the particle i is updated by using (15):

$$x_{ij} = \begin{cases} 1, & \text{rand} < \text{sigm}\left(v_{ij}\right), \\ 0, & \text{otherwise}, \end{cases} \quad (15)$$

where $\text{sigm}(v_{ij})$ is the sigmoid function, used to scale the velocities between 0 and 1, defined as:

$$\text{sigm}\left(v_{ij}\right) = \frac{1}{1 + \exp\left(-v_{ij}\right)} \quad (16)$$

Generate Function. SA uses a notion of neighborhood relation. Let S be the set of all feasible solutions to the problem and $f : S \to \Re$ the objective function to be minimized. A neighborhood relation is a binary relation $N \subseteq S \times S$ with some desired properties. The interpretation of $N(s, s')$ is that solution s is a neighbor of solution s' in the search space of all solutions S. A neighbor heuristic proceeds in steps. Starting search at some initial solution s_0 and then each step moves from the current solution to some neighbor according to rules specific to the heuristic. At each iteration, SA algorithm generates a random neighbor of the *current_solution* (line 10). The neighborhood relation is defined as follows:

(1) with probability equal to 0.03, the value of each element of *current_solution* is flipped from 1 to 0 or from 0 to 1;

(2) validate solution:
while *Not feasible* (*current_solution* must satisfy the constraint of not exceeding the maximum SPM capacity.) (*current_solution*) do
Remove the data j having a low number of access from *current_solution*: ($current\_solution[j] \leftarrow 0$).

Accept Function. the key idea in the SA approach is the function *Accept* which specifies the probability of accepting the move from *current_solution* to a *Neighbor* solution, which also depends on so called temperature (T). The function *Accept* should satisfy the following conditions:

(1) $p = 1$ if solution *Neighbor* is better than *current_solution* in terms of the cost function f:
(i.e., $f(Neighbor) < f(current\_solution)$ in a minimization problem);

(2) if *Neighbor* is worse than *current_solution* the value of p is positive (i.e., it allows for moving to a worse solution), but decreases with $|f(Neighbor) - f(current\_solution)|$;

(3) for fixed *current_solution* and *Neighbor*, when *Neighbor* is worse than *current_solution* the value of p decreases with time and tends to 0.

The function $Accept(C_{cost}, N_{cost}, T)$ is decided by the probability of accepting configuration *Neighbor*. This probability is given by the following formula:

$$p = \begin{cases} 1, & N_{cost} < C_{cost}, \\ \frac{1}{2}\text{rand} * \left(1 + e^{(C_{cost} - N_{cost})/T}\right), & \text{otherwise}, \end{cases} \quad (17)$$

where T is the temperature and rand is a random number independently generated within the range of $[0, 1]$.

5.3. Discrete Cooperative Distributed Hybrid HPSO-SA Algorithm. For Distributed hybrid HPSO-SA (*HPSO-SA_Dist*) algorithm, we use independent subswarms of particles with their own fitness functions which evolve in isolation, except for an exchange of some particles (migration). A set of $m = 30$ particles is assigned to each of the P processors, for a total population size of $m \times P$. The set assigned to each processor is its subswarm. The processors are connected by an interconnection network with a ring topology. Initial subswarms consist of a randomly constructed assignment created at each processor. Each processor, disjointly and in parallel, executes the *HPSO-SA_Seq* algorithm on its subswarm for a certain number of generations. Afterwards, each subswarm exchanges ($HPSO - SA\_Dist$ runs in *Asynchronous* mode: at the 100th iteration, each processor sends its best solution and continues the improvement of its subswarm and verifies if it does not receive a solution from its neighbor.) its best particle (migrant) with its neighbors. We exchange the particles themselves (i.e., the migrant is removed from one subswarm and added to another). Hence, the size of the subswarm remains the same after migration (the worst particle is removed). The process continues with the separate improvement of each current solution for a maximum number of iterations. At the end of the process the best solution that exists constitutes the final assignment.

5.4. Experimental Results. In order to compute the energy cost of studied memory architecture composed by an SPM, an instruction cache and a DRAM, we proposed an energy consumption estimation model which is explained in [11]. Hybrid HPSO-SA algorithms and TS have been implemented on a cluster of PCs running under Windows XP Professional version 2002. The cluster is composed by 4 Pentium (D) machines running at 3 GHz. Each processor has 1 Gbyte of memory. Table 7 gives a description of the benchmarks used and they also can be downloaded from [28].

In experiments, 30 different executions for each heuristic are performed and the best and average results obtained on these 30 executions are recorded. In this case, the best and the average solutions give similar results. Figure 3 shows that both *HPSO-SA_Seq* and *HPSO-SA_Dist* achieve better performances than TS on energy savings. In fact, hybrid HPSO-SA heuristics consume from 76.23% (StatemateCE) to 98.92% (ShaCE) less energy than TS.

As *HPSO-SA_Seq* and *HPSO-SA_Dist* give similar results, we decide to experiment their behavior when considering their execution time. We recorded the average execution times needed by *HPSO-SA_Seq* and *HPSO-SA_Dist* (running on a cluster of 4 PCs) to achieve the 30 executions. Figure 4 presents the results obtained on the largest (size) benchmarks. From this figure, we see that the Distributed HPSO-SA version (*HPSO-SA_Dist*) is faster than the Sequential HPSO-SA version (*HPSO-SA_Seq*). In fact, *HPSO-SA_Dist* requires 73.16% (AdpcmCE) to 84.65% (CntCE) less execution time than *HPSO-SA_Seq*. The Distributed HPSO-SA version is always faster than the Sequential HPSO-SA version.

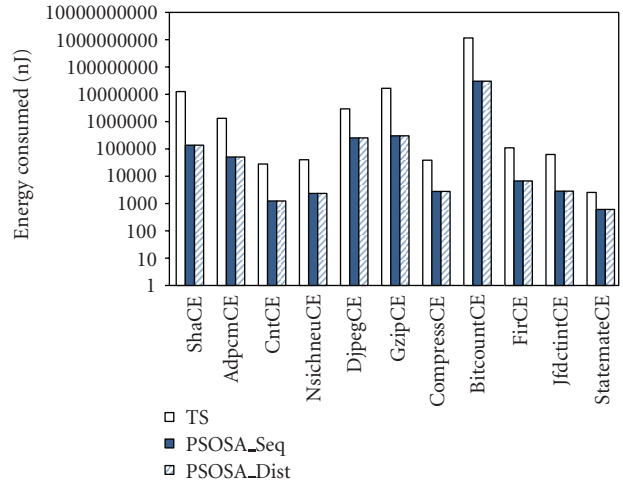

Figure 3: Energy consumed by benchmarks studied in this work.

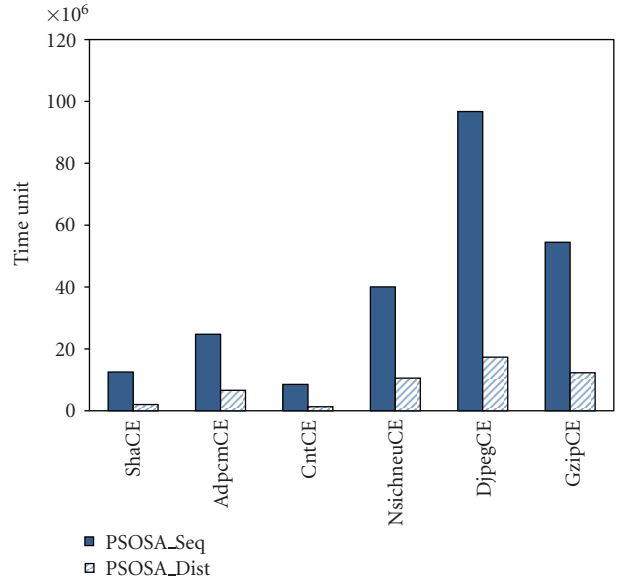

Figure 4: Execution time used by HPSO-SA algorithms on benchmarks studied in this work.

6. Conclusion and Perspectives

In this paper, we have designed a hybrid algorithm (HPSO-SA) that combines the exploration ability of PSO with the exploitation ability of SA, and is capable of preventing premature convergence. Compared with QIPSO, ATREPSO and GMPSO [2], Tl-PSO [8], PSO-SA [9] and SUPER-SAPSO [10] on well-known benchmark functions and for the problem of reducing energy consumption in embedded systems memories, it has been shown that HPSO-SA performs well in terms of accuracy, convergence rate, stability and robustness. In future, we will also compare the performances

Table 7: List of Benchmarks.

Benchmark	Suite	Description
ShaCE	MiBench	The secure hash algorithm that produces a 160- bit message digest for a given input.
BitcountCE	MiBench	Tests the bit manipulation abilities of a processor by counting the number of bits in an array of integers.
FirCE	SNU-RT	Finite impulse response filter (signal processing over a 700 items long sample).
JfdctintCE	SNU-RT	Discrete-cosine transformation on 8×8 pixel block.
AdpcmCE	Mälardalen	Adaptive pulse code modulation algorithm.
CntCE	Mälardalen	Counts nonnegative numbers in a matrix.
CompressCE	Mälardalen	Data compression using lzw.
DjpegCE	Mediabenchs	JPEG decoding.
GzipCE	Spec 2000	Compression.
NsichneuCE	Wcet Benchs	Simulate an extended Petri net. Automatically generated code with more than 250 if-statements.
StatemateCE	Wcet Benchs	Automatically generated code.

of HPSO-SA with the above mentioned algorithms on the embedded systems memory saving problem.

In addition, we will compare HPSO-SA algorithm with other hybrid algorithms (PSO-GA, PSO-MDP, PSO-TS) whose design is in progress by the authors. Comparison will also be done on additional benchmark functions and more complex problems including functions with dimensionality larger than 30.

Acknowledgments

The authors are grateful to anonymous referees for their pertinent comments and suggestions. Dawood Khan helped the authors with the intricacies of the english language. The work of M. Idrissi Aouad is supported by the French national research agency (ANR) in the Future Architectures program.

References

[1] M. Locatelli, "Simulated annealing algorithms for continuous global optimization," in *Handbook of Global Optimization*, P. M. Pardalos and H. E. Romeijn, Eds., vol. 2, pp. 179–230, Kluwer Academic, 2001.

[2] M. Pant, R. Thangaraj, and A. Abraham, "Particle swarm based metaheuristics for function optimization and engineering applications," in *Proceedings of the 7th Computer Information Systems and Industrial Management Applications (CISIM '08)*, vol. 7, pp. 84–90, IEEE Computer Society, Washington, DC, USA, 2008.

[3] J. Kennedy and C. E. Russell, "Swarm intelligence," in *Morgan Kaufmann*, Academic Press, 2001.

[4] A. M. Marco, T. Stützle et al., "Convergence behavior of the fully informed particle swarm optimization algorithm," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pp. 71–78, 2008.

[5] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons, 2005.

[6] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization. An overview," in *Swarm Intelligence*, vol. 1, pp. 33–57, 2007.

[7] P. N. Suganthan, N. Hansen, J. J. Liang et al., "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," Tech. Rep. 2005005, Nanyang Technological University, Singapore; IIT Kanpur, India, 2005.

[8] S. Nakano, A. Ishigame, and K. Yasuda, "Consideration of particle swarm optimization combined with tabu search," *IEEJ Transactions on Electronics, Information and Systems*, vol. 128, pp. 1162–1167, 2008, Special Issue on 'The Electronics, Information and Systems Conference Electronics, Information and Systems Society, I.E.E. of Japan'.

[9] G. Yang, D. Chen, and G. Zhou, "A new hybrid algorithm of particle swarm optimization," in *Lecture Notes in Computer Science*, vol. 4115, pp. 50–60, 2006.

[10] M. Bahrepour, E. Mahdipour, R. Cheloi, and M. Yaghoobi, "Super-sapso: a new sa-based pso algorithm," in *Applications of Soft Computing*, vol. 58, pp. 423–430, 2009.

[11] M. I. Aouad, R. Schott, and O. Zendra, "A tabu search heuristic for scratch-pad memory management," in *Proceedings of the International Conference on Software Engineering and Technology (ICSET '10)*, vol. 64, pp. 386–390, WASET, Rome, Italy, 2010.

[12] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[13] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948, IEEE Computer Society, 1995.

[14] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '98)*, pp. 69–73, IEEE Computer Society, 1998.

[15] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the Congress on Evolutionary Computation (CEC '99)*, pp. 1945–1950, 1999.

[16] W. J. Xia and Z. M. Wu, "A hybrid particle swarm optimization approach for the job-shop scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 29, no. 3-4, pp. 360–366, 2006.

[17] D. Chaojun and Z. Qiu, "Particle swarm optimization algorithm based on the idea of simulated annealing," *International Journal of Computer Science and Network Security*, vol. 6, no. 10, pp. 152–157, 2006.

[18] L. Fang, P. Chen, and S. Liu, "Particle swarm optimization with simulated annealing for tsp," in *Proceedings of the 6th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED '07)*, pp. 206–210, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wis, USA, 2007.

[19] X. Wang and J. Li, "Hybrid particle swarm optimization with simulated annealing," in *Proceedings of the 3rd International Conference on Machine Learning and Cybernetics (ICMLC '04)*, vol. 4, pp. 2402–2405, 2004.

[20] S. Kirkpatrick, "Optimization by simulated annealing: quantitative studies," *Journal of Statistical Physics*, vol. 34, no. 5-6, pp. 975–986, 1984.

[21] L. Morten, K. R. Thomas, and T. Krink, *Hybrid Particle Swarm Optimization with Breeding and Subpopulations*, Springer, Berlin, Germany, 2000.

[22] "ITRS, System Drivers," 2007, http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_SystemDrivers.pdf.

[23] M. I. Aouad and O. Zendra, "A survey of scratch-pad memory management techniques for low-power and -energy," in *Proceedings of the 2nd ECOOP Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems (ICOOOLPS '07)*, pp. 31–38, Berlin, Germany, 2007.

[24] H. B. Fradj, A. El Ouardighi, C. Belleudy, and M. Auguin, "Energy aware memory architecture configuration," *SIGARCH Computer Architecture News*, vol. 33, no. 3, pp. 3–9, 2005.

[25] M. Gendreau, *An Introduction to Tabu Search*, vol. 57, Kluwer Academic, Boston, Mass, USA, 2003.

[26] A. Tanenbaum, *Architecture de l'ordinateur*, 5th edition, 2005.

[27] U. P. H. Kellerer and D. Pisinger, *Knapsack Problems*, Springer, Berlin, Germany, 2004.

[28] "Benchmarks," http://www.loria.fr/~idrissma/benchs.zip.