HW/SW Partitioning Using Discrete Particle Swarm

Amin Farmahini-Farahani
ECE Department
University of Tehran
Tehran 14395-515, IRAN
a.farmahini@ece.ut.ac.ir

Mehdi Kamal
CE Department
Sharif University of Technology
Tehran 11365-8639, IRAN
kamal@ce.sharif.edu

Sied Mehdi Fakhraie, Saeed Safari
ECE Department
University of Tehran
Tehran 14395-515, IRAN
{fakhraie, saeed}@ut.ac.ir

ABSTRACT
Hardware/Software partitioning is one of the most important issues of codesign of embedded systems, since the costs and delays of the final results of a design will strongly depend on partitioning. We present an algorithm based on Particle Swarm Optimization to perform the hardware/software partitioning of a given task graph for minimum cost subject to timing constraint. By novel evolving strategy, we enhance the efficiency and result's quality of our partitioning algorithm in an acceptable run-time. Also, we compare our results with those of Genetic Algorithm on different task graphs. Experimental results show the algorithm's effectiveness in achieving the optimal solution of the HW/SW partitioning problem even in large task graphs.

Categories and Subject Descriptors
G.1.6 [**Numerical Analysis**]: Optimization.

General Terms
Algorithms, Design, Management.

Keywords
HW/SW Partitioning, Discrete Particle Swarm Optimization.

1. INTRODUCTION
Embedded system on chips (SoCs) consist of one or more processors, which run some software, and another set of hardware blocks implemented with application specific integrated circuit. With the development of IC technology, the scale of modern embedded systems becomes larger and the functions become more complicated. Nowadays, embedded systems are used in a wide range of industrial areas. Many commercial application SoCs are made up of many processors, coprocessors, memories, A/D, D/A, and IP cores.

For instance, in an embedded signal processing application, it is common to use both application-specific hardware accelerator circuits and general-purpose programmable units running appropriate software.

GLSVLSI'07, March 11-13, 2007, Stresa-Lago Maggiore, Italy.

This mixture is beneficial since hardware is usually much faster than software, but it is also significantly more expensive. Software solutions on the other hand are cheaper to create and to maintain, but slow [20]. Hence, normally performance-critical components of a system should be realized in hardware, and non-critical components in software. By this way, a good tradeoff between cost and performance can be achieved. The process of simultaneous design of hardware and software components of a system is known as codesign [17]. The problem consists of finding a design of the system that meets the performance and cost requirements.

However, this kind of system design creates some challenges. One of the most critical steps of the design process is partitioning, i.e. deciding which components of a system should be realized in hardware and which ones in software. In this step, the system's operation is partitioned in smaller functional tasks, always having in mind a set of constraints, and optimal cost-performance trade-off is to be found. As the systems to design have become more and more complex, research efforts have been undertaken to automate partitioning as much as possible and HW/SW codesign including partitioning is becoming a promising solution to modern embedded system [10].

When the number of tasks is increased, finding the best partitioning is becoming more difficult, so using search and optimization methods are appropriate approaches. One group of popular and powerful approaches for search and optimization problems are Evolutionary Algorithms (EAs) [25], [26]. Nowadays, EAs are widely used in many engineering applications. Particle Swarm Optimization (PSO) is a form of evolutionary algorithm that was developed to simulate a simplified social system [1]. A binary-valued PSO is developed by the creators of PSO for using in binary search spaces [3]. In this paper, we propose a discrete PSO technique for hardware software partitioning application.

The remaining sections of this paper are organized as follows: Section 2 reviews the related work so far. The chosen hardware software partitioning algorithm is explained in Section 3. Section 4 gives an overview of the PSO and our methodology. The achieved results are explained in Section 5. Finally, Section 6 concludes the paper and explains future works.

2. RELATED WORK
At present, there are limited numbers of algorithms for HW/SW partitioning. Ref. [9] shows an algorithm to solve the joint problem of partitioning and scheduling. It consists of basically two local search heuristics: one for partitioning and one for

scheduling. The two algorithms operate on the same graph, at the same time. The objective of the technique is to minimize the worst case latency of the task graph subject to the area constraints on the architecture.

The earlier solutions have considered the HW/SW partitioning problem as a mixed Integer Linear Program (ILP). This integer program is a part of a two-phase heuristic optimization scheme which aims at gaining better timing estimations using repeated scheduling phases, and using the estimates in the partitioning phases. This approach was slow and only practical for small problems [11], [12], [13]. Arato *et al.* [20] presented an ILP-based approach that works good for quite big systems.

Following this, Simulated Annealing (SA) was used as a non-deterministic solution to reduce the computational cost of codesign. Recently, Genetic Algorithms (GA) have been employed to solve the problem [14]. Saha *et al.* [15] have modeled partitioning problem as a Constraint Satisfaction Problem (CSP), and have presented a GA-based approach to solve the CSP. Ref. [17] has addressed the functional partitioning problem using GA and have presented a comparative study with SA and modified version of Fiduccia-Matheyse. Ref. [18] has developed an Extended GA (EGA) that implemented a novel selection method with function scaling, adaptive crossover and mutation. They showed EGA outperforms Standard GA and it is easier to apply due to fewer parameters. Ref. [19] has established a HW/SW partitioning model based on system's Basic Scheduling Block (BSB) graph and propose a modified genetic partitioning algorithm. Tan *et al.* [14] have proposed some performance measures, test problem, and have given a comparison of some of the prior work on GAs.

3. PARTICLE SWARM OPTIMIZATION

3.1 Particle Swarm Algorithm

The particle swarm optimization is a new population based optimization algorithm, which has been proposed by Kennedy and Eberhart in 1995 [1]. The PSO begins with a random population and searches for fitness optimum just like the GA, but in the PSO algorithm, particles will evolve by cooperation and competition among the individuals themselves through generations instead of using genetic operators [2].

Instead of using evolutionary operators to manipulate the individual, as in other evolutionary computational algorithms, each individual in PSO flies in the search space with a velocity which is dynamically adjusted according to its own flying experience and its companions' flying experience. Each individual is treated as a volume-less particle in the d-dimensional search space.

Each particle keeps track of its coordinates in the problem space, which are associated with the best solution (fitness) it has achieved so far. This value is called *pbest*. Another best value that is tracked by the global version of the particle swarm optimizer is the overall best value, and its location, obtained so far by any particle in the population. This location is called *gbest*.

At each time step, the PSO algorithm consists of velocity changes of each particle toward its *pbest* and *gbest* locations. Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pbest* and *gbest* locations. The modified velocity and position of each individual particle can be calculated using the current velocity and the distance from $pbest_{id}$ to $gbest_d$, as shown in the following equations:

$$v_{id}^{k+1} = w.v_{id}^{k} + c_1 r_1 (pbest_{id} - x_{id}^{k}) + c_2 r_2 (gbest_d - x_{id}^{k}) \quad (1),$$

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \quad (2),$$

where x_{id}^{k} is current position of individual i at iteration k, which has velocity v_{id}^{k} and $V_d^{\min} \le v_{id}^{k} \le V_d^{\max}$ and the *pbest* is the historical best position of x_{id}^{k} and *gbest* is the global best position in the population's history. Besides, there are five parameters should be defined, w is the inertia weight factor, c_1 and c_2 are acceleration constants and r_1 and r_2 are uniform random numbers between 0 and 1. The evolution process generally begins with random distribution and evolutes as the Equations (1) and (2).

In the above procedures, the parameter V_{max} determines the resolution, or fitness, with which regions between the present position and target position are searched. If V_{max} is too high, particles may fly past the good solutions. If V_{max} is too small, particles may not explore sufficiently beyond local solutions. In previous experience with PSO, V_{max} was often set at 10-20% of the dynamic range of the variable on each dimension.

The constants c_1 and c_2 represent the weighting of the stochastic acceleration terms that pull each particle toward *pbest* and *gbest* positions. Low values allow particles to roam far from target regions before being tugged back. On the other hand, high values result in abrupt movements toward, or past, the target regions. Hence, the acceleration constants c_1 and c_2 are often set to be 2.0 according to the past experiences.

Suitable selection of inertia weight w in Equation (3) provides a balance between global and local exploration and exploitation, and on average results in less iterations required to find a sufficiently optimal solution. As originally developed, w often decreases linearly from about 0.9 to 0.4 during a run. In general, the inertia weight w is set according to the following equation:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{iter_{\max}} \times iter \quad (3),$$

where $iter_{max}$ is the maximum iteration number (generation) and *iter* is the current iteration number.

3.2 Discrete Particle Swarm Algorithm

The original PSO algorithm can only optimize problems in which the elements of the solution are continuous real numbers. Discrete Particle Swarm Optimization can be obtained by replacing (2) with (4).

$$\begin{aligned} &\textit{if (rand()} < S(v_{id}^{k+1})) \quad \textit{then} \quad x_{id}^{k+1} = 1; \\ &\qquad\qquad\qquad\qquad \textit{else} \quad x_{id}^{k+1} = 0; \end{aligned} \quad (4)$$

where $S(v)$ is a sigmoid limiting transformation function $S(v) = 1 / (1 + e^{-v})$, and *rand()* is a random number selected from a uniform distribution in [0.0, 1.0]. V_{max} in the discrete particle swarm is to set a limit to further exploration after the population has

converged; in a sense, it could be said to control the ultimate mutation rate or temperature of the bit vector. Note also that, while high V_{max} in the continuous-valued version increases the range explored by a particle, the opposite occurs in the discrete version; smaller values allow a higher mutation rate [3].

There have been some other explorations of Discrete PSO techniques for discrete optimization. Yang *et al.* [4] developed an algorithm which uses a different method to update velocity. Al-kazemi and Mohan [5] used a technique which had particles alternatively exploit personal best and neighborhood best positions instead of simultaneously.

4. HARDWARE/SOFTWARE PARTITIONING

Hardware software codesign is gaining importance with the advent of CAD for embedded systems. An important phase in such approaches is partitioning the design into hardware and software implementation sets.

4.1 Problem Definition

We have addressed the problem of HW/SW partitioning using PSO, with the aim of achieving a near optimal solution efficiently. Let the sets S = $\{s_1, s_2,\ldots, s_n\}$ and H = $\{h_1, h_2,\ldots, h_n\}$ denote the sets of all possible mappings of the tasks to SW and HW respectively. There are four approaches for solving a HW/SW partitioning [22]:

Q_1: H_0, S_0 are the given constraints. Find a HW/SW partition P so that $H_p \leq H_0$ *and* $S_p \leq S_0$ (hardware cost should be less than H_0 and execution time should be less than S_0).

Q_2: H_0 is the given constraint. Find a HW/SW partition P so that $H_p \leq H_0$ and S_p is minimal (hardware cost should be less than H_0 and have the minimum execution time).

Q_3: S_0 is the given constraint. Find a HW/SW partition P so that $S_p \leq S_0$ and H_p be minimal (execution time should be less than S_0 and have minimum hardware cost).

Q_4: Find a HW/SW partition P so that S_p and H_p be minimal (minimum execution time and minimum hardware cost).

It is proved that Q_1 is NP-Complete and Q_2, Q_3 are NP-hard [23]. Also, Q_4 could be mapped to maximum-flow minimum-cut problem [24] which has algorithms with $O(n^3)$ complexity. In this paper, HW/SW partitioning is performed according to the Q_3 type.

At the beginning, the problem should be modeled. There are three predetermined characteristics for the model as follows:

Each software component has a time related to its execution time. Software cost has been ignored under the supposition of having enough available memory in system to accommodate the whole implementation.

Each hardware component has a cost related to its hardware demands such as area and power consumption and a time relative to its execution time.

Only if two communicating components are mapped to different contexts, their communication has an associated cost. Otherwise, the expense of communication between them is negligible.

The partitioning formal problem would be expressed as follows: "Given a set of tasks that together represent a design, the problem is to find a partition of hardware and software implementations, such that joining them constructs an equivalent system with a minimal cost, while satisfying performance constraints." Thus, we want to determine which task should be implemented in hardware and which in software in a manner that the overall design meets cost and time constraints.

4.2 Problem Representation

To partition a design, first it must be mapped into a task graph. Applications can be easily broken down into distinct tasks at a coarse level of granularity and can be specified by a data-dependence-based task graph.

The behavior of the design is presented to the partitioning algorithm through a directed acyclic task graph. It represents the high level system to be implemented. It consists of a set of nodes and a set of edges and is represented by $G = (V, E)$. Each node $v \in V$ denotes an operation (task) that has a specific cost of implementation on HW and times of execution on HW (e.g., FPGA, ASIC) and SW platforms on general-purpose processor or DSP chip. A node represents a single thread of execution and cannot be preempted, i.e. it is atomic. Each task may be executed in hardware or software. Each edge $e \in E$ denotes data dependencies between nodes. The edges in the data flow graph have communication delays depending on the partitions in which the two nodes incident on the edge are present [6], [8]. We have used task graphs to investigate the correctness of algorithm and run-time obtained by applying the PSO. A sample task graph is shown in Figure 1.

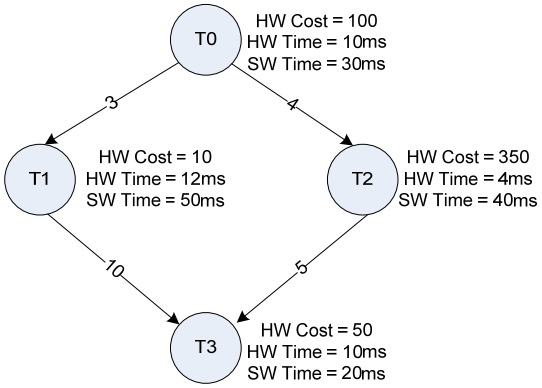

Figure 1. Task graph of a design, values on the edges represent communication costs.

To generate task graphs, we have used a task graph generator called "Task Graph For Free, TGFF" [7]. TGFF generates parameters for each task (HW and SW computation times, HW implementation cost, etc.). We have modified TGFF source code to generate applicable graphs.

The problem representation is one of the most important parameters in the discrete PSO. Each particle is a string of bits

that illustrate a scheme of system partitioning. The length of particle is equals to the number of tasks. Each bit defines that a task of system must be implemented in hardware or developed with software. An example of a particle is shown in Figure 2. In bit string, zeros determine software development and ones hardware implementation.

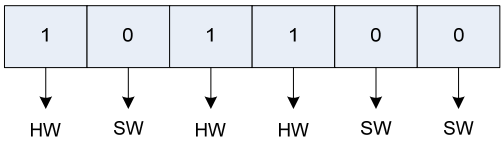

Figure 2. An example of a particle used in partitioning.

4.3 Partitioning Methodology

Figure 3 illustrates our overall methodology. The evaluation of the fitness function (codesign cost & time estimator) needs some information for its computations. The data is given by an input file including this information: number of tasks, estimated hardware execution time, hardware cost, software time of each task, connections of the tasks with others and the associated communication cost. Then, a task graph is generated using the input file.

In the beginning, particles and corresponding velocities are randomly generated. In each generation (iteration), fitness of each particle is evaluated and an estimation of cost and time is achieved, and *pbest* and *gbest* are determined. Then position and velocity of each particle is updated using (1) and (4) respectively. This process is resumed until termination condition is met. Termination conditions are generation number limit, run-time limit, or convergence of algorithm to a predefined fitness value.

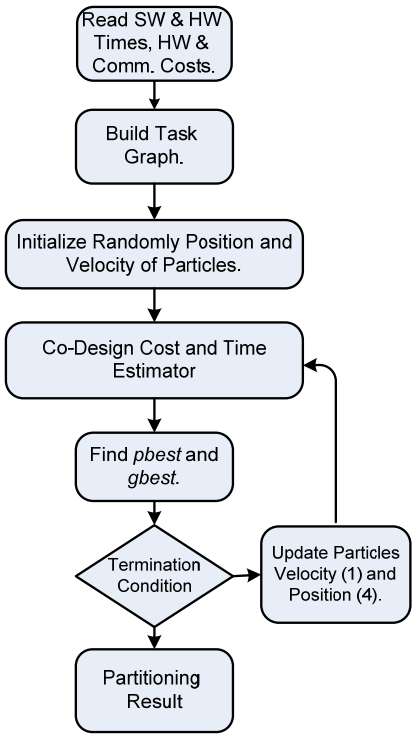

Figure 3. Partitioning methodology.

4.4 Fitness Function

In order to evaluate an obtained solution, we need to know the goodness of a partition, so we require some metrics. A metric is a measurement of an attribute of the partitioned system. Usually some of these metrics are combined and a fitness function (codesign cost and time estimator) is obtained to guide the algorithm through the optimization process. Some of the possible metrics are economical cost, performance time, power consumption, silicon area, number of pins**,** memory size, lines of code, or communications cost.

In our partitioning, the focus is on performance (execution time), area costs and communication costs between software and hardware blocks. The main objective of our solution is minimizing the system cost while maintaining the constraint requirement for worst execution time. At the beginning, all source nodes are traversed to the destination nodes using depth first search (DFS) algorithm [21] and the worst execution time will be determined. The fitness value is defined as infinite if the worst execution time of each individual is bigger than the constraint value. Otherwise, the fitness value is the cost of the resulting system. An optimal solution will have a minimal cost and meet the timing constraint. The pseudo code of the fitness function is as below:

```
If (DFS(Graphi) <= Time_Constraint)
        Fitness_Value = Communication_Cost + Hardware_Cost;
Else
        Fitness_Value = ∞;
```

5. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed method, we have implemented the partitioning algorithm using PSO and GA and then compared the results of the experiments. C programming language is used for the implementation. To generate proper random task graphs, we have modified TGFF program and made some task graphs and experiments with the discrete PSO and GA algorithm. Five task graphs have been generated with different number of nodes and edges. We have entitled each task graph with respect to its number of nodes and edges. The PSO and GA parameters which we have used are shown in Table 1. We have set GA parameters to achieve the best results. Uniform crossover, multipoint mutation, and Tournament selection is used for combination, mutation, and selection operations, respectively.

Table 1. Discrete PSO and GA parameters.

Discrete PSO		**GA**	
Parameter	Value	Parameter	Value
Population	50	Population	50
V_{max}	10	Crossover Rate	0.8
V_{min}	-10	Mutation Rate	0.02
c_1, c_2	2	Elitism	2

We have run PSO and GA on a computer with 1.7 GHz CPU and 512 MB RAM. Each algorithm has been run for 1000 generations

and the results are reported in Table 2. A time constraint is defined for each graph, so the algorithm must find a solution, if any, smaller than this time constraint. In this table, *cost* includes hardware cost and communication cost and *run-time* denotes the CPU time of running the algorithm.

Table 2. Discrete PSO and GA results over 1000 generations and 20 trials.

		Discrete PSO		GA	
Nodes/ Edges	Time constraint	Cost	Run-time (ms)	Cost	Run-time (ms)
182/235	1000	368.7	4101	3240.5	4142
296/344	1700	1746.8	19014	8020.8	21419
402/500	500	662.4	9211	11153	10096
502/585	3300	9037	15861	21864	17236
992/1083	5500	15162	20440	33915.7	18189

The results show that the proposed method outperforms GA in all cases. In particular for 402/500 nodes/edges task graph, our proposed method has achieved better result with less than 6% of GA cost. Run-time of task graphs in GA is moderately equals to those of PSO, because run-time is mostly dedicated to fitness function evaluation. However, for small task graphs, GA takes more time than PSO for a fixed number of generations, and also, in an exceptional case, in 992/1083 nodes/edges task graph, PSO takes more time.

We have also evaluated the performance of the GA and the discrete PSO in a different way in which a limited run-time is assumed. The results are depicted in Figure 4. In this figure, for smaller task graphs run-time is limited to 50 seconds and for 992/1083 nodes/edges task graph is limited to 100 seconds.

Figure 4 shows the discrete PSO can find better partitions than GA in a fixed run time. Although GA has faster convergence and achieves better performances in earlier generations, while discrete PSO outperforms it at the end. For example, in 182/235 nodes/edges task graph, GA converges to a local optima solution. However, discrete PSO explores the search spaces gradually and can find optimal or near optimal solution, if the algorithm is given enough time.

6. CONCLUSION

In this paper, we have proposed an approach for hardware/software partitioning using discrete particle swarm optimization. We have shown the discrete PSO performance is competitive with those of GA and is able to find near optimal solution for hardware/software partitioning. The proposed solution finds better partitioning than GA in all cases. However, in some cases, PSO takes more time for a fixed number of generations. Experimental results show that the proposed method not only can obtain low-cost solution, but can also avoid the phenomenon of premature convergence of the GA, resulting in robustness and effectiveness in solving partitioning of task graphs with hundreds of nodes.

7. REFERENCES

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Intl. Conf. Neural Networks,* vol. 4, 1995, pp. 1942-1948.

[2] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE World Cong. on Computational Intelligence*, 1998, pp. 96-73.

[3] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. IEEE Conf. Syst., Man, and Cybernetics*, Orlando, FA, 1997, pp. 4104-4109.

[4] S. Yang, M. Wang, and L. Jiao, "A quantum particle swarm optimization," in *Cong. Evolutionary Computing*, vol. 1, Jun. 2004, pp. 320-324.

[5] B. Al-kazemi and C. K. Mohan, "Multi-phase discrete particle swarm optimization," in *Proc. Intl. Workshop Frontiers in Evolutionary Algorithms*, 2002.

[6] V. Srinivasan, S. Radhakrishnan, and R. Vemuri, "Hardware software partitioning with integrated hardware design space exploration," in *Proc. Design Automation and Test in Europe*, 1998, pp. 28-35.

[7] R. P. Dick, D. I. Rhodes, and W. Wolf, "TGFF: task graphs for free," in Proc. *Int. Workshop Hardware-Software codesign*, Mar. 1998, pp. 97-101.

[8] K. Bhasyam and K. Bazargan, "HW/SW codesign incorporating edge delays using dynamic programming," in *Proc. Euromicro Symp. Digital System Design*, Sept. 2003.

[9] K. S. Chatha and R. Vemuri, "MAGELLAN: multiway hardware software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs," in *Proc. Intl. Conf. Hardware-Software Codesign and System Synthesis*, 2001.

[10] R. Ernst, "Codesign of embedded systems: status and trends," in P*roc. IEEE Design & Test of Computers*, 1998, pp.45-54.

[11] W. Wolf, "A decade of hardware/software codesign," in *Computer*, pp. 38-43, Apr. 2003.

[12] R. Niemann and P. Marwedel, "An algorithm for hardware/software partitioning using mixed integer linear programming," in *Proc. Design Automation for Embedded Systems, special issue: Partitioning Methods for Embedded Systems*, vol. 2, Mar. 1997, pp. 165-193.

[13] R. Niemann, "Hardware/software codesign for data flow dominated embedded systems," Kluwer Academic Publishers, 1998.

[14] R. A. Wildman, J. I. Kramer, D. S. Weile, and P. Christie, "Multi-objective optimization of interconnect geometry," in *IEEE Trans. On Very Large Scale Integration Syst.*, pp. 15-23, Feb. 2003.

[15] K. C. Tan, T. H. Lee, and E. F. Khor, "Evolutionary algorithms for multi-objective optimization: performance assessments and comparisons," in *Proc. Cong. Evo1utionary Computation*, May 2001, pp. 979-986.

[16] D. Saha, R. S. Mitra, and A. Basu, "Hardware software partitioning using genetic algorithm," in *Proc. Int. Conf. VLSI design*, Jan. 1997, pp. 155-160.

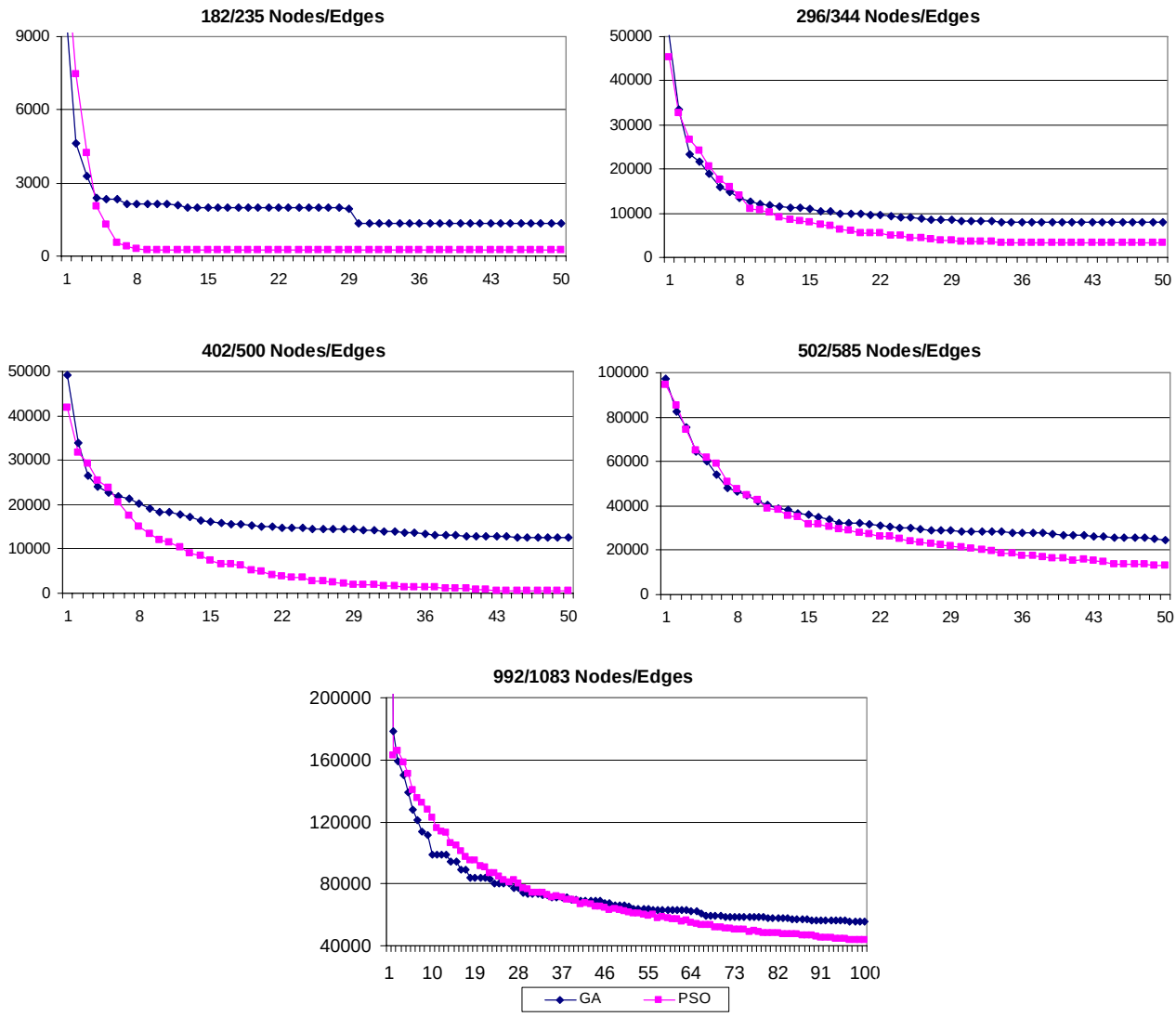

Figure 4. Average performances for the GA and the PSO against randomly generated task graphs. For each graph, the *x* axis shows the run-time in second and the *y* axis represents the average fitness computed over 20 trials. In all graphs, a lower *y* value represents partition with a lower cost.

[17] J. I. Hidalgo, and J. Lanchares, "Functional partitioning for hardware/software codesign using genetic algorithm," in *Proc. Euromicro Conf.*, 1997.

[18] M. J. W. Savage, Z. Salcic, G. Coghill, and G. Covic, "Extended genetic algorithm for codesign optimization of DSP syst. in FPGAs," in *Proc. IEEE Intl. Conf. Field-Programmable Technology*, Dec. 2004, pp. 291-294.

[19] Y. Zou, Z. Zhuang, and H. Chen, "HW/SW partitioning based on genetic algorithm," in *Proc. Cong. Evolutionary Computation*, vol. 1, Jun. 2004, pp. 628-633.

[20] P. Arato, S. Juhasz, Z. A. Mann, A. A. Orban, and D. Papp, "Hardware software partitioning in embedded system design," in *Proc. Intelligent Signal Processing*, Sept. 2003.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, MIT Press, 2001.

[22] K. S. Chatha, and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," in *IEEE Trans. Very Large Scale Integration Syst.*, vol. 10, pp. 193–208, Jun. 2002.

[23] P. Arato, S. Juhasz, Z. A. Mann, A. A. Orban, and, D. Papp, "Hardware software partitioning in embedded system design," in *Proc. Intelegent Signal Processing*, 2003.

[24] M. Sgroi, L. Lavagno, and A. Sangiovanni-Vincentelli, "Formal Models for Embedded System Design," in *IEEE Design & Test of Computers*, vol. 17, no. 2, Jun. 2000, pp.14-27.

[25] A. E. Eiben, and J. E. Smith, *Introduction to evolutionary computing*, Berlin Heildberg: Springer-Verlag, 2003.

[26] T. Mitchell, *Machine learning*, New York, McGraw-Hill, 1997.