



ELSEVIER

Available online at [www.sciencedirect.com](http://www.sciencedirect.com)

SCIENCE @ DIRECT®

Computers & Operations Research 33 (2006) 820–835

computers &  
operations  
research

[www.elsevier.com/locate/cor](http://www.elsevier.com/locate/cor)

# Hybrid meta-heuristics algorithms for task assignment in heterogeneous computing systems

Sancho Salcedo-Sanz<sup>a,\*</sup>, Yong Xu<sup>b</sup>, Xin Yao<sup>b</sup>

<sup>a</sup>*Department of Signal Theory and Communications, Universidad Carlos III de, 28911 Leganes, Madrid, Spain*

<sup>b</sup>*Centre for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, The University of Birmingham*

Available online 21 September 2004

## Abstract

In this paper we tackle the task assignment problem (TSAP) in heterogeneous computer systems. The TSAP consists of assigning a given distributed computer program formed by a number of tasks to a number of processors, subject to a set of constraints, and in such a way a given cost function to be minimized. We introduce a novel formulation of the problem, in which each processor is limited in the number of task it can handle, due to the so called *resource constraint*. We propose two hybrid meta-heuristic approaches for solving this problem. Both hybrid approaches use a Hopfield neural network to solve the problem's constraints, mixed with a genetic algorithm (GA) and a simulated annealing for improving the quality of the solutions found. We test the performance of the proposed algorithms in several computational TSAP instances, using a GA with a penalty function and a GA with a repairing heuristic for comparison purposes. We will show that both meta-heuristics approaches are very good approaches for solving the TSAP.

© 2004 Elsevier Ltd. All rights reserved.

*Keywords:* Task assignment; Heterogeneous computer systems; Genetic algorithms; Simulated annealing; Meta-heuristics

## 1. Introduction

Task assignment is an important issue in distributed computing systems, which provides a better exploitation of the system parallelism and improves its performance [1,2]. The so called *task assignment problem* (TSAP hereafter) is a combinatorial optimization problem which consists of assigning a given

\* Corresponding author. Tel.: +34-91-624-5973; fax: +34-91-624-8749.

E-mail address: [sancho@tsc.uc3m.es](mailto:sancho@tsc.uc3m.es) (S. Salcedo-Sanz).

computer program formed by a number of tasks to a number of processors/machines, subject to a set of constraints, and in such a way a given cost function to be minimized. The constraints of the TSAP are usually related to the resources available for the processors in the system (the number of tasks that a given processor is able to handle is limited by its characteristics, memory constraints etc., in order to avoid overloading). If a temporal constraint is added to the execution of the tasks, the problem is generally known as a *task scheduling problem*, we do not consider this case in this paper. The cost function for a TSAP can be different depending on the designer necessities, but it usually involves the minimization of completion time of the entire program, the minimization of the communication time among tasks or the minimization of the processors' load. The TSAP can be defined in several ways, in [1] and [3] for example, it is defined as a two graphs matching problem, and it is proven to be NP-complete. In this paper we will provide an alternative definition similar to the terminal assignment problem [4], which is also a NP-complete problem.

Several approaches to task assignment in distributed computer systems have been described in the literature. Specially important are the heuristics techniques based on mathematical programming and graph theory [1–3,5–8]. The majority of these approaches consider two graphs for defining the TSAP, first a graph where each node represents a task, and each edge between two nodes represents the amount of communication of the two tasks; and second a network graph in which a node represents a machine and each edge connecting two nodes represents the communication cost between the two machines. Using these two graphs, the TSAP can be seen as a graph matching problem, NP-complete in general, where very different heuristics have been applied [1,2].

In the last few years, however, there have been an increasing interest for meta-heuristics approaches, as robust approaches for tackle the TSAP and task scheduling problems. Genetic algorithms (GA), [9,10], Evolutionary Algorithms GA [11] and simulated annealing (SA) [12] are the most used meta-heuristics techniques, but also other important approaches have been reported using neural networks [13,14] mean field annealing [15] or tabu search [16]. Among these meta-heuristics approaches, hybrid approaches formed by the mixing of two algorithms have provided very good results for the TSAP. The most important hybrid algorithms for task assignment and task scheduling in the literature are the works by Park [15], where a mixed genetic algorithm-mean field annealing is presented, Lin and Yang [17] where several heuristic operators are mixed with traditional GA operators, and the work by Zhu et al. [13] where a hybrid Hopfield neural network-annealing algorithm is analyzed.

In this paper we present two different hybrid meta-heuristic algorithms for the TSAP. Both of them are based on a fast binary Hopfield neural network, which solves the problem's constraints. We hybridize this network with two meta-heuristic approaches for improving the quality of the solutions found by the network, a GA and a SA. We compare our approach against a GA which manages the problem's constraints by means of a penalty function, and a GA with a repairing algorithms for obtaining feasible solutions to the problem. We will show that our algorithm is able to obtain very good results in the TSAP in a reasonable computation time comparing with other approaches, for a hard combinatorial optimization problem like TSAP.

The rest of the paper is structured as follows: In the next section we provide the problem definition, the model of the computing system we consider and the cost function we use in this work. In Section 3 we present the proposed approaches. It is divided in three parts: first we describe the binary Hopfield neural network which solves the problem's constraints, second we present the GA and third the SA, that are used as global search meta-heuristics. Section 4 shows the results obtained by our approaches in several computational experiments, compared with the results obtained by a GA using a penalty

function for managing the problem's constraints. We conclude the paper giving some final remarks in Section 5.

## 2. Problem definition

Consider an heterogeneous distributed system formed by a set of processors/machines  $P = \{P_1, \dots, P_M\}$ , of different speed but with the same architecture, in such a way that a set of different tasks  $T = \{T_1, \dots, T_N\}$  of a distributed application can be executed on them. The TSAP model we consider consists of assigning each task to one of the processors, in such a way that a given cost function (usually the execution time and the communications cost of the system) to be minimized. Let  $W = \{w_1, \dots, w_N\}$  the amount of resources required by the task  $T_i$ , and  $R = \{r_1, \dots, r_M\}$  the maximum resources available for a given processor  $P_i$ , then the task  $T_i$  may be assigned to the processor  $P_j$  if and only if the sum of the weights from all the previous tasks assigned to the processor  $P_j$ , plus  $w_i$  is less or equal than the corresponding maximum resource constraint of the processor  $P_j$ ,  $r_j$ . The resources required by each task, as well as the maximum resources available for a given processor, must be estimated in advance, taking into account that the overloading of processors should be avoided. Other approaches in the literature use different approaches for solving this point, for example in [1] a *interference cost* is implemented in the cost function for avoiding processor overload, and [13] takes into account the differences in completion time among tasks, using them to measure the load of the processors. Using our approach, any feasible solution ensures that no processor in the system will be overloaded after the task assignment.

Mathematically, the TSAP can be defined as follows:

Let  $\mathbf{X}$  be a binary matrix such that every element on it  $x_{ij} = 1$  if task  $i$  has been assigned to processor  $j$ , and  $x_{ij} = 0$  otherwise.

Find  $\mathbf{X}$  which minimizes a cost function  $f(\mathbf{X})$

subject to

$$\sum_{j=1}^M x_{ij} = 1 \quad i = 1, 2, \dots, N \quad (1)$$

and

$$\sum_{i=1}^N w_i x_{ij} \leq r_j \quad j = 1, 2, \dots, M. \quad (2)$$

Note that the first constraint, from Eq. (1), ensures that each task must be associated with one and only one processor, and the second constraint from equation (2) implies that the resource constraint on each processor cannot be violated.

### 2.1. Cost function

Following previous approaches to tasks assignment in computer networks [2,13], the cost function of a TSAP should take into account the minimization of the total execution time and the minimization of the communication time between tasks in different processors. This paper considers the

following model:

- (1) All the tasks may be executed in all the processors. Two tasks executing in different processors incur in a communications cost. This work does not consider communication cost between tasks executing in the same processor (intra-processor communication).
- (2) We define a matrix of communication costs  $\mathbf{K}$ , where each element  $k_{ijpq}$  represents the communication cost between a task  $i$  executing in processor  $j$  and a task  $p$  executing in a different processor  $q$ .
- (3) Let  $v_j$  be the relative speed of a given processor  $j$  to the slowest processor in the system, being the speed of the slowest processor 1. Let  $t_i$  be the time of execution of a given task  $i$  in the slowest processor of the system. We define  $t_j^e = \sum_{i:x_{ij}=1} t_i/v_j$  as the total amount of time needed by the processor  $j$  in order to finish the tasks assigned to it.

Using the definitions above, we write the cost function  $f(\mathbf{X})$  for the TSAP as

$$f(\mathbf{X}) = \alpha_1 \sum_{j=1}^M t_j^e + \alpha_2 \sum_{i=1}^N \sum_{j=1}^M \sum_{p=1}^N \sum_{\substack{q=1 \\ q \neq j}}^M k_{ijpq} x_{ij} x_{pq}, \tag{3}$$

where  $\alpha_1$  and  $\alpha_2$  stand for two parameters which control the importance of each term in the cost function, and  $\alpha_1 + \alpha_2 = 1$ .

### 3. Proposed approaches

In this Section we describe the meta-heuristic approaches for the TSAP that we propose. First we describe the hybrid approaches, presenting the HNN used and the global search heuristics (a GA and a SA).

#### 3.1. Hybrid approaches for the TSAP

##### 3.1.1. The Hopfield neural network

The Hopfield network we use as a local search algorithm for solving the TSAP constraints belongs to a class of binary Hopfield networks [18] where the neurons can only take values 0 or 1. The dynamics of this network depends on a matrix  $C$  which defines the minimum distance between two 1 s in the network for each row, and on the initial state of the neurons. See [18–20] for further details. The structure of the HNN can be described as a graph, where the set of vertices are the neurons, and the set of edges define the connections between the neurons. We map a neuron to every element in the solution matrix  $\mathbf{X}$ . In order to simplify the notation, we shall also use matrix  $\mathbf{X}$  to denote the neurons in the Hopfield network. The HNN dynamics can then be described in the following way: after a random initialization of every neuron with binary values, the HNN operates in a serial mode. This means that only one neuron is updated at a time, while the rest remain unchanged. Denoting by  $x_{ij}(t)$  the state of a neuron at time  $t$ , the updating rule is described by

$$x_{ij}(t) = \text{isgn} \left( \sum_{\substack{p=1 \\ p \neq i}}^N \sum_{\substack{q=\max(1, c_i, p+1) \\ q \neq j}}^{\min(M, j+c_i, p)} x_{pq} \right) \quad \forall i, j, \tag{4}$$

where the *isgn* operator is defined by

$$isgn(a) = \begin{cases} 0 & \text{if } a > 0 \text{ or } \sum_{i=1}^N w_i x_{ij} > r_j \quad \forall j. \\ 1 & \text{otherwise.} \end{cases} \tag{5}$$

Note that the updating rule only takes into account neurons  $x_{pq}$  with value 1 within a distance of  $c_{ip}$ . The matrix  $C$  is an  $N \times N$  matrix which encodes the problem’s constraints, and it is defined as follows:

$$c_{ij} = \begin{cases} M & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Note that this matrix forces one and only one 1 per row (each task must be assigned to one and only one processor), whereas there may be several 1s in the same column (the same processor can manage several tasks).

In the updating rule defined above, the neurons  $x_{ij}$  are updated in their natural order, i.e.,  $i = 1, 2, \dots, N$ ,  $j = 1, 2, \dots, M$ . We introduce a modification of this rule by performing the updating of the neurons in a random ordering of the rows (variable  $i$ ). This way the variability of the feasible solutions found will increase. Let  $\pi(i)$  be a random permutation of  $i = 1, 2, \dots, N$ . The new updating rule of the HNN is

$$x_{\pi(i)j}(t) = isgn \left( \sum_{\substack{p=1 \\ p \neq \pi(i)}}^N \sum_{\substack{q=\max(1, c_{j, \pi(i), p+1}) \\ q \neq j}}^{\min(M, j+c_{j, \pi(i), p})} x_{pq} \right) \quad \forall i, j. \tag{7}$$

The resulting updating rule runs over the rows of  $\mathbf{X}$  in the order given by the permutation  $\pi(i)$ , but the columns are updated in natural order  $j = 1, 2, \dots, M$ .

A *cycle* is defined as the set of  $N \times M$  successive neuron updates in a given order. In a cycle, every neuron is updated once following the given order  $\pi(i)$ , which is fixed during the execution of the algorithm. After every cycle, the convergence of the HNN is checked. The HNN is considered converged if none of the neurons have changed their state during the cycle. The final state of the HNN dynamics is a potential solution for the TSAP, which fulfils the problem’s constraints given in Section 2. Note, however, that the solution found may be infeasible if any of the tasks is not assigned.

### 3.1.2. Implementation example

In this section we provide an example of how the binary Hopfield neural net work used in this paper works. Consider a small TSAP example formed by 4 tasks and 4 processors. Let us imagine that each task requires a certain amount of resources, which can be represented as  $w_i = 4$ ,  $i = \{1, 2, 3, 4\}$ , whereas the maximum resources available for each processor is  $r_i = 6$ ,  $i = \{1, 2, 3, 4\}$ . With these values for  $w_i$  and  $r_i$ , a feasible solution should assign one task to each processor. Following Eq. (6), the constraints matrix  $C$  associated to this problems is then:

$$C = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}.$$

We are interested in showing the process of reaching a feasible solution starting from an infeasible one. Let us suppose that the initial infeasible solution, generated at random, is the following:

$$\mathbf{X}_0 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

In order to apply Eq. (7) to the initial solution  $\mathbf{X}_0$ , we need to select an updating ordering in rows and columns. We choose the natural order in rows and columns for simplicity, so in this case the permutation  $\pi$  which defines the order of updating will be  $\pi = \{1, 2, 3, 4\}$ . For coming up with a feasible solution, we apply Eq. (7) and the *isgn* operator to  $\mathbf{X}_0$ , obtaining the following solution:

$$\mathbf{X}_{\text{final}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

It is easy to check that in this case it takes only two cycles to converge to the final solution  $\mathbf{X}_{\text{final}}$  from the initial one  $\mathbf{X}_0$ . Note that, once the initial state of the network  $\mathbf{X}_0$  and the order of updating ( $\pi$ ) are fixed, the final state  $\mathbf{X}_{\text{final}}$  is defined. Different initial states or updating orderings will produce different final solutions. It is also important to note that this Hopfield neural network does not take the objective function into account, but only the problem's constraints, defined by matrix  $C$  and *isgn* operator, in this case.

### 3.1.3. Hybrid approach HNNGA: The Genetic Algorithm

GAs are robust problem's solving techniques based on natural evolution processes. They are population-based techniques which codifies a set of possible solutions to the problem, and evolve it through the application of the so called *genetic operators* [21]. The standard genetic operators in a GA are:

- Selection, where the individuals of a new population are selected from the old one. In the standard implementation of the selection operator, each individual has a probability of surviving for the next generation proportional to its associated fitness value (roulette wheel).
- Crossover, where new individuals are searched starting from couples of individuals in the population. Once the couples are randomly selected, the individuals have the possibility of swapping parts of themselves with its couple, the probability of this happens is usually called *crossover probability*,  $P_c$ .
- Mutation, where new individuals are searched by randomly changing bits of current individuals with a low probability  $P_m$  (*probability of mutation*).

In this paper we propose a GA to be hybridized with the Hopfield network in order to improve the quality of its solutions. Our GA codifies a population of  $\chi$  potential solutions for the TSAP, as binary strings of length  $N \times M$ . Each string represents a different assignment matrix  $\mathbf{X}$  which is passed through the Hopfield network in order to get a feasible assignment. The population is evolved through successive generations by means of the application of the standard genetic operators selection, crossover and mutation described above.

We implement an elitist strategy, which consists of passing the individual with the highest fitness to the next generation. This way, our algorithm always preserve the best solution found in the evolution. The complete algorithm for the TSAP, formed by the GA and the HNN described in Section 3.1.1, is summarized below:

#### HNNGA algorithm

---

Initialize GA population at random

**while** (max. number of generations has not been reached) **do**

**for** (every individual  $\mathbf{X}$ )

    Run the HNN to obtain a feasible  $\mathbf{X}$ .

    Calculate the fitness value of the individual  $f(\mathbf{X})$ .

    if  $\mathbf{X}$  is not feasible, apply a penalty to  $f(\mathbf{X})$ .

    Substitute the GA individual by the new  $\mathbf{X}$  obtained through the HNN.

**endfor**

**selection**

**crossover**

**mutation**

**end(while)**

---

#### 3.1.4. Hybrid approach HNNSA: the simulated annealing

SA has been widely applied to solve combinatorial optimization problems [22–26]. It is inspired by the physical process of heating a substance and then cooling it slowly, until a strong crystalline structure is obtained. This process is simulated by lowering an initial temperature by slow stages until the system reaches to an equilibrium point, and no more changes occur. Each stage of the process consists of changing the configuration several times, until a thermal equilibrium is reached, and then a new stage starts, with a lower temperature. The solution of the problem is the configuration obtained in the last stage. In the standard SA, the changes in the configuration are performed in the following way: a new configuration is built by a random displacement of the current one. If the new configuration is better, then it replaces the current one, and if not, it may replace the current one probabilistically. This probability of replacement is high in the first stages of the algorithm, and decreases in every stage. The solution found by SA can be considered a “good enough” solution, but it is not guaranteed to be the best [25].

In this paper we consider the hybridization of a SA and the Hopfield neural network presented in Section 3.1.1 for solving the TSAP. The main idea behind this is that configurations involved in the SA are feasible solutions for the TSAP. The SA will then search for the best feasible solution with respect to a given cost function, in this case a non-standard cost function for the TSAP.

The most important parts in a SA algorithm are: the objective function to be minimized during the process, the chosen representation for solutions and the mutation or configuration change operator. The objective function to be minimized is the objective function defined in Section 2. The representation of the problem is the assignment matrix  $\mathbf{X}$  and the change from one configuration to another is performed by means of the standard flip mutation of a given number  $N_f$  of bits in  $\mathbf{X}$ .

The complete algorithm for the TSAP, formed by the SA and the HNN described in Section 3.1.1, performs in the following way:

HNNNSA algorithm:

---

```

k = 0;
T = T0;
Initialize a potential solution at random;
  Run the HNN to obtain X;
  evaluate(X, f(X));
  if X is not feasible, apply a penalty to f(X).
repeat
  for j = 0 to ξ
    Xmut = mutate(X);
    Run the HNN to obtain X;
  evaluate(Xmut, f(X));
  if X is not feasible, apply a penalty to f(X).
  if ((f(Xmut) < f(X)) OR (random(0, 1) < e(-α/T))) then
    X = Xmut;
  endif
endfor
T = fT(T0, k);
k = k + 1;
until(T < Tmin);

```

---

where  $k$  counts the number of iterations performed;  $T$  keeps the current temperature;  $T_0$  is the initial temperature;  $T_{\min}$  is the minimum temperature to be reached;  $\mathbf{X}$  stands for the current configuration and  $\mathbf{X}_{\text{mut}}$  for the new configuration after the mutation operator is applied;  $f$  represents the cost function considered (see Section 2);  $\xi$  is the number of changes performed for a given temperature  $T$ ;  $f_T$  is the freezer function; and  $\alpha$  is a constant. Parameter  $\alpha$  and the initial temperature  $T_0$  are chosen to have an initial acceptance probability about 0.8, a value usually used. The freezer function is defined as

$$f_T = \frac{T_0}{1 + k}. \quad (8)$$

The minimum temperature  $T_{\min}$  is calculated on the basis of the desired number of iterations ( $numIt$ ) as

$$T_{\min} = f_T(T_0, numIt). \quad (9)$$

## 4. Computational experiments and results

### 4.1. Generation of test instances and meta-heuristic parameters

In order to test the performance of our approaches, we tackle a set of TSAP instances of different sizes. The test problems for simulating a distributed computer network were generated using the following model:

Table 1  
Main characteristics of the instances tackled

Problem #	Tasks	Processors
1	30	3
2	30	4
3	30	5
4	40	3
5	40	4
6	40	5
7	50	3
8	50	4
9	50	5
10	75	3
11	75	4
12	75	5
13	100	3
14	100	4
15	100	5

The time a given task takes for finishing in the slowest processor ( $t_i, i = 1, \dots, N$ ) has been randomly generated with values between 1 and 10 from an uniform probability distribution. The speed of the processors in the system ( $v_j, j = 1, \dots, M$ ) have also been randomly generated, with values between 1 and 5, ensuring that at least one processor has the value 1 (slowest processor). The amount of resources needed by each task ( $w_i$ ) is randomly generated from an uniform distribution between 1 and 6. The total resources available for each processor ( $r_j, j = 1, \dots, M$ ) have been generated in such a way that all the processors have the same resources available, and the total processors resources to be between 1% and 5% of the total task needed resources. Finally, each element of the matrix  $\mathbf{K}$  of communication costs between tasks have been randomly generated from an uniform distribution with values between 1 and 10.

Table 1 shows the main characteristics of the generated problems. There are 15 test instances, of different difficulties. In general, the difficulty increases with the problem size. For these problems, we run the meta-heuristic algorithms compared using the following parameters: the GA was run with a population of  $\chi = 50$  individuals, evolving during 300 generations. The crossover probability  $P_c = 0.6$  and the mutation probability was fixed to  $P_m = 0.01$ . The main parameters of the SA are  $\xi = 50$  and  $numIt = 300$ . Note that this way, the number of function evaluations is the same for both meta-heuristics considered. The value of  $N_f$  in the mutation operator is fixed to  $N_f = 20$ .

#### 4.2. GAs for comparison purposes

In order to compare our hybrid approaches, we have implemented two different GAs. The first one uses a penalty function for managing the problem's constraints. This way of managing the problem's constraints is well known for the GA research community, and has provided good results in different problems of combinatorial optimization with constraints [24,27]. The second GA, in addition to the penalty function, implements a *repairing heuristic* [28], in order to get feasible solutions to the problem.

Following [10] instead of a binary representation, in these GAs each solution  $\mathbf{X}$  is encoded as an integer string of length  $\mathbf{N}$ ,  $\tilde{x}$ , such that  $\tilde{x}_i = j$  means that task  $i, i = 1, \dots, N$  has been assigned to processor

$j, j = 1, \dots, M$ . Note that, using this encoding method, there will be no infeasible solutions due to unassigned tasks to processors, but only due to infeasible assignments.

The management of the problem's constraints in the first GA we consider is carried out by means of a term of penalty for solutions with infeasible assignments. We name this approach as  $GA_{\text{penalty}}$ . The penalty function is defined to be proportional to the number of infeasible assignments:

$$\text{Penalty} = a \cdot \text{numInf}, \quad (10)$$

where  $\text{numInf}$  stands for the number of infeasible assignments and  $a$  is the penalty for one infeasible assignment, usually it is a parameter to be tuned for each problem.

In the second GA, in addition to the term of penalty we implement a *repairing heuristic* for managing the infeasible assignments. We call this GA as  $GA_{\text{repair}}$ . The repairing heuristic works in the following way:

#### Repairing heuristic

---

Calculate the load of the processors.

$\gamma = 1$ ;

**while** (max. number of loops  $\gamma_{\text{max}}$  has not been reached) **do**

**for**(each overloaded processor  $j$ )

**if**  $\tilde{x}_i = j$ , **then**

$\tilde{x}_i = k, k \neq j$  randomly chosen.

Recalculate the load of processors.

**end(for)**

$\gamma = \gamma + 1$ ;

**end(while)**

---

The selection and crossover operators in both GAs are the same as in our hybrid HNNGA approach (see Section 3.1.3). The mutation operator consists of changing the value of  $N_g$  randomly chosen genes by a different value in  $\{1, \dots, M\}$ . This process is carried out with a probability  $P_m$ , equal to the probability of mutation of an individual in the HNNGA algorithm. The rest of the parameters in this algorithm are: population  $\chi = 50$  individuals, 300 generations and  $N_g = 10$ . The parameter  $\gamma_{\text{max}}$  of the  $GA_{\text{repair}}$  heuristic was fixed to 20.

#### 4.3. Results and analysis

We run each algorithm 15 times for each problem, keeping the best, average and standard deviation values provided by each algorithm. The parameters of the cost function considered are  $\alpha_1 = 0.5$  and  $\alpha_2 = 0.5$  (see Section 2.1). Other values of  $\alpha_1$  and  $\alpha_2$  are possible depending on whether the execution total time or the communication total cost is considered more important.

Table 2 shows the comparison of the results obtained by the HNNGA and HNNSA approaches. It is interesting that the HNNGA approach outperforms the HNNSA in the majority of problems. However, in the hardest problems #12–#15, it is the HNNSA who obtains better results. Table 3 shows the results of a  $t$ -test performed over the data obtained by the two compared algorithms. We perform this analysis in order to know if the differences in performance showed in Table 2 between algorithms are statistically

Table 2  
Comparison of the results obtained by the different algorithms considered

Problem #	HNNGA			HNNSA		
	Best	Avg.	Dev.	Best	Avg.	Dev.
1	2101.4	2133.7	16.9	2123.3	2154.5	14.2
2	2113.3	2124.4	24.4	2127.2	2160.6	16.8
3	2090.6	2119.7	22.5	2114.3	2162.7	21.6
4	3946.5	3978.2	22.3	3975.9	4003.9	16.7
5	3780.9	3871.2	46.7	3859.4	3904.1	29.5
6	3808.6	3873.5	33.8	3876.4	3929.9	24.7
7	6166.1	6232.0	43.9	6243.9	6281.3	20.6
8	6114.2	6220.4	48.25	6186.7	6264.0	39.1
9	6161.6	6222.7	35.4	6237.5	6281.7	31.8
10	14316.3	14438.4	77.3	14378.8	14425.3	45.9
11	14219.4	14337.2	54.2	14267.5	14376.6	50.3
12	14292.8	14473.0	88.6	14357.4	14458.9	46.8
13	25880.6	26088.7	108.0	25759.5	25918.8	74.5
14	25935.6	26100.4	98.8	25879.1	26014.7	76.5
15	25763.9	26001.1	77.1	25749.8	25921.3	94.3

Best, average and standard deviation values are provided.

Table 3  
 $t$  values obtained by a two-tailed  $t$ -test for Problems 1 to 15

Problem #	HNNGA-HNNSA
1	-3.65 <sup>†</sup>
2	-0.97
3	-4.60 <sup>†</sup>
4	-3.02 <sup>†</sup>
5	-2.12
6	-6.33 <sup>†</sup>
7	-4.49 <sup>†</sup>
8	-3.44 <sup>†</sup>
9	-4.78 <sup>†</sup>
10	-0.82
11	-2.53 <sup>†</sup>
12	-0.52
13	1.8
14	2.81 <sup>†</sup>
15	2.19

<sup>†</sup> stands for values of  $t$  with 14 degrees of freedom which are significant at  $\alpha = 0.05$ .

significant or not. Note that the HNNGA algorithm performs statistically better in 8 out of 15 problems considered, whereas the HNNSA only performs significant better in 1, though it obtains better results than the HNNGA in the hardest problems.

Table 4  
Comparison of the results obtained by the different algorithms considered

Problem #	GA <sub>penalty</sub>			GA <sub>repair</sub>		
	Best	Avg.	Dev.	Best	Avg.	Dev.
1	2117.7	2154.5	23.0	2115.3	2155.3	26.2
2	2139.9	2184.6	27.5	2112.4	2155.5	25.8
3	2200.5	2265.1	43.9	2112.6	2179.7	35.0
4	3992.9	4066.1	33.9	3967.6	4001.1	21.5
5	3786.9	3881.9	33.9	3796.8	3881.7	49.4
6	4033.0	4120.6	33.8	3916.1	3973.6	43.9
7	6208.1	6282.2	49.3	6228.1	6301.9	49.5
8	6273.6	6545.9	97.6	6303.7	6347.5	32.1
9	6347.6	6429.8	63.1	6232.5	6293.0	36.9
10	14369.7	14566.1	89.2	14315.1	14434.6	66.5
11	14308.4	14489.0	83.9	14306.2	14483.3	83.1
12	14525.1	14675.2	86.4	14384.8	14502.7	78.3
13	26097.3	26271.6	79.2	25991.3	26117.0	106.7
14	26161.1	26256.4	81.6	26038.4	26174.0	89.5
15	26147.2	26287.1	102.2	25780.2	26089.6	155.8

Best, average and standard deviation values are provided.

Table 5  
 $t$  values obtained by a two-tailed  $t$ -test for Problems 1–15

Problem #	HNNGA – GA <sub>penalty</sub>	HNNSA – GA <sub>penalty</sub>
1	−2.72 <sup>†</sup>	−0.09
2	−3.73 <sup>†</sup>	−2.37
3	−13.69 <sup>†</sup>	−7.14 <sup>†</sup>
4	−8.62 <sup>†</sup>	−6.68 <sup>†</sup>
5	−0.75	−1.52
6	−13.39 <sup>†</sup>	−11.51 <sup>†</sup>
7	−2.74 <sup>†</sup>	−0.06
8	−11.59 <sup>†</sup>	−8.92 <sup>†</sup>
9	−10.93 <sup>†</sup>	−7.20 <sup>†</sup>
10	−3.55 <sup>†</sup>	−4.85 <sup>†</sup>
11	−4.28 <sup>†</sup>	−3.14 <sup>†</sup>
12	−6.61 <sup>†</sup>	−9.75 <sup>†</sup>
13	−4.92 <sup>†</sup>	−10.54 <sup>†</sup>
14	−4.26 <sup>†</sup>	−7.39 <sup>†</sup>
15	−7.03 <sup>†</sup>	−12.06 <sup>†</sup>

<sup>†</sup> stands for values of  $t$  with 14 degrees of freedom which are significant at  $\alpha = 0.05$ .

Table 4 shows the results obtained by the GAs which we use for comparison purposes. In order to facilitate a direct comparison with our hybrid algorithms, we have also calculated the  $t$ -test over the data obtained by the algorithms. This can be seen in Table 5 for HNNGA, HNNSA and GA<sub>penalty</sub> and in Table 6 for the comparison HNNGA, HNNSA and GA<sub>repair</sub>. The  $t$ -test values shown in both tables corroborate that

Table 6

$t$  values obtained by a two-tailed  $t$ -test for Problems 1–15

Problem #	HNNGA – GA <sub>repair</sub>	HNNNSA – GA <sub>repair</sub>
1	–2.6 <sup>†</sup>	–0.1
2	–0.33	–0.67
3	–5.5 <sup>†</sup>	1.41
4	–3.13 <sup>†</sup>	0.04
5	–0.7	1.7
6	–6.29 <sup>†</sup>	–3.38 <sup>†</sup>
7	–4.03 <sup>†</sup>	–1.47
8	–6.63 <sup>†</sup>	–5.6 <sup>†</sup>
9	–4.78 <sup>†</sup>	–0.94
10	0.15	–0.45
11	–5.17 <sup>†</sup>	–3.95 <sup>†</sup>
12	–0.89	–1.85
13	–0.68	–4.87 <sup>†</sup>
14	–2.14 <sup>†</sup>	–4.96 <sup>†</sup>
15	–1.72	–3.79 <sup>†</sup>

<sup>†</sup> stands for values of  $t$  with 14 degrees of freedom which are significant at  $\alpha = 0.05$ .

the hybrid algorithms we propose always perform better than the GA<sub>penalty</sub> algorithm, and the differences between this algorithm and our hybrid approaches are statistically significant in the majority of cases. The introduction of some *tailoring* by means of the repairing heuristic makes the algorithm stronger, as can be seen in Table 6. Our hybrid algorithms perform better than the GA<sub>repair</sub> algorithm as well. In some cases the differences between them are not statistically significant, but in the majority of problems our approaches obtains better solutions in terms of the objective function.

Regarding the computational time of the analyzed algorithms, it is expected that the hybrid approaches to be more costly than the GA with penalty function and also than the GA with the repairing algorithm, since the HNN convergence is a time consuming process, above all in large instances. In order to give an idea of running times, we have measured the real computational time<sup>1</sup> that the different algorithms needs to complete 15 000 function evaluations. Fig. 1 shows this for each problem considered. Note that adding heuristics for improving the quality of the solutions found increases the computational cost of the algorithm, as expected. However, the hybrid approaches solve all the problems, even the hardest ones, in less than 2 min, which is a reasonable amount of time for a hard combinatorial optimization problem.

The results above show that the hybrid meta-heuristics proposed in the paper are an appealing option for the TSAP. Both of them obtain results which improve the performance of different GAs for this problem. In computational time, both hybrid meta-heuristic approaches have the same performance, since both implement the same HNN. We have also shown that the GA with penalty function and the GA with the

<sup>1</sup> The real computational time measures the elapsed real time between invocation and termination of the algorithm, reading of data and writing of results in files included. The simulations were run in a simulation platform *Dual Xeon/2.8 GHz*.

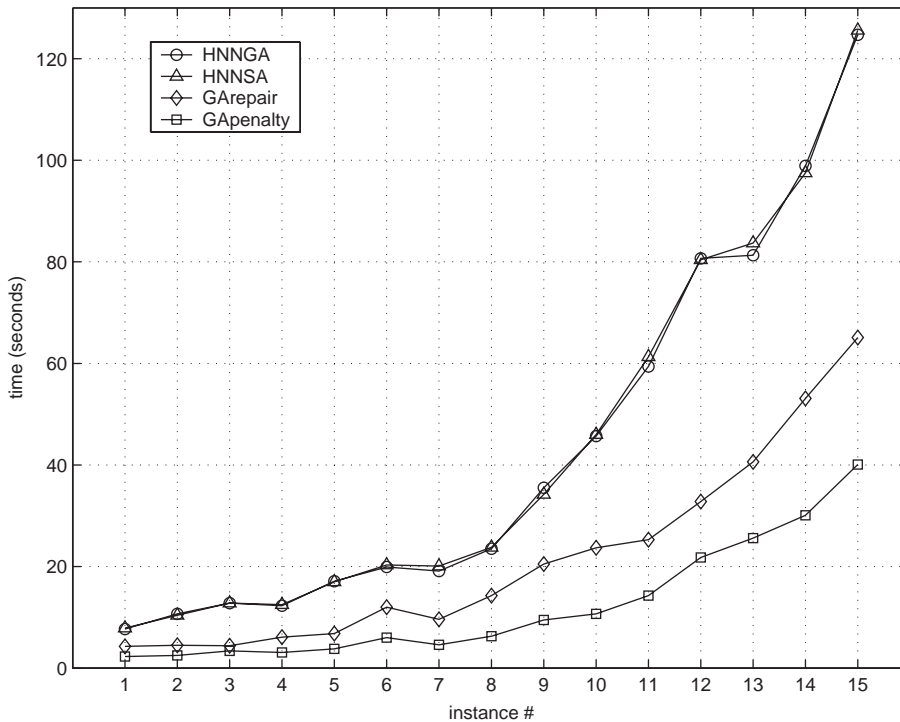


Fig. 1. Real time of the different algorithms compared for the different problems considered.

repairing algorithm obtain solutions of lower quality in terms of the objective function, but in a faster way.

## 5. Conclusions

In this paper we have presented two hybrid meta-heuristic approaches to the TSAP in heterogeneous computer networks. We consider a novel formulation of the problem, in which each processor is limited in the number of task it can handle. Our approach consists of a Hopfield neural network (HNN) which manages the problem's constraints, and we use two different global search algorithms for improving the quality of the solutions found: a GA and a SA.

We have tested the performance of our approaches in several computational test instances, comparing it with the performance of a GA with a penalty function and a GA with a repairing heuristic. We have found that both hybrid meta-heuristics proposed outperform the GA with penalty function and the GA with repairing heuristic in terms of minimization of the cost function. The approach HNN using the GA as a global search algorithm obtains better results in the majority of test performed, however, the approach HNN with the SA has found better solutions in the most difficult instances.

The test performed show that both meta-heuristic approaches are an interesting option for tackling the TSAP in heterogeneous computer networks.

## Acknowledgements

Dr. Sancho Salcedo-Sanz is supported by a postdoctoral fellowship of Ministerio de Educación Cultural y Deporte of Spain, fellowship number EX2003-0463. The authors would like to thank Prof. G. Laporte and one anonymous referee for their assistance and help in the revision of this paper.

## References

- [1] Lo VM. Heuristic algorithms for task assignment in distributed systems. *IEEE Transactions on Computers* 1988;37(11):1384–97.
- [2] Kafil M, Ahmad I. Optimal task assignment in heterogeneous distributed computing systems. *IEEE Concurrency* 1988;6(3):42–51.
- [3] Fernández-Baca D. Allocating modules to processors in a distributed system. *IEEE Transactions on Software Engineering* 1989;15(11):1427–36.
- [4] Khuri S, Chiu T. Heuristic algorithms for the terminal assignment problem. In: *Proceedings of the 1997 ACM Symposium on Applied Computing*. ACM Press; 1997. p. 247–51.
- [5] Chaudhari V, Aggarwal JK. A generalized scheme for mapping parallel algorithms. *IEEE Transactions on Parallel and Distributed Systems* 1993;4(3):328–46.
- [6] El-Rewini H, Lewis TG. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing* 1990;9:138–53.
- [7] Chiu CC, Yeh YS, Chou JS. A fast algorithm for reliability-oriented task assignment in a distributed system. *Computer Communications* 2002;25:1622–30.
- [8] Lee CH, Shin KG. Optimal task assignment in homogeneous networks. *IEEE Transactions on Parallel and Distributed Systems* 1997;8(2):119–29.
- [9] Woo SH, Yang SB, Kim SD, Han TD. Task scheduling in distributed computing systems with a genetic algorithm. In: *Proceedings of High Performance on Information superhighway*. 1997. p. 301–5.
- [10] Alaoui SM, Frieder O, El-Ghazawi T. A parallel genetic algorithm for task mapping on parallel machines. *Lecture Notes in Computer Science*, vol. 949. 1999.
- [11] Zomaya A, Ward C, Macey B. An evolutionary approach for scheduling in parallel processor systems. In: *Proceedings of High Performance on Information superhighway*. 1997. p. 301–5.
- [12] Beck JE, Siewiorek DP. Simulated annealing applied to multicomputer task allocation and processor specification. In: *Proceedings of IEEE Symposium on Parallel and Distributed Processing*. 1996. p. 232–9.
- [13] Zhu W, Liang TY, Shieh CK. A Hopfield neural network based task mapping method. *Computer Communications* 1999;22:1068–79.
- [14] Cooling JE, Korousic-Seljak B. Task scheduling using neural networks within hardware co-processor. In: *Proceedings of Seventh Mediterranean Electrotechnical Conference*. 1994. p. 317–20.
- [15] Park K. A heuristic approach to task assignment optimization in distributed systems. In: *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*. 1997. p. 1838–42.
- [16] Porto SCS, Ribeiro CC. Tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *International Journal of High-Speed Computing* 1993;7:45–71.
- [17] Lin M, Yang LT. Hybrid genetic algorithms for scheduling partially ordered tasks in a multi-processor environment. In: *Proceedings of Sixth International Conference on Real Time Computing Systems and Applications*. 1999. p. 382–6.
- [18] Shrivastava Y, Dasgupta S, Reddy SM. Guaranteed convergence in a class of Hopfield networks. *IEEE Transactions on Neural Networks* 1992;3(6):951–61.
- [19] Salcedo-Sanz S, Bousoño-Calzón C, Figueiras-Vidal AR. A mixed neural-genetic algorithm for the broadcast scheduling problem. *IEEE Transactions on Wireless Communications* 2003;2(2):277–83.
- [20] Salcedo-Sanz S, Santiago-Mozos R, Bousoño-Calzón C. A hybrid Hopfield network-simulated annealing approach for frequency assignment in satellite communications systems. *IEEE Transactions on Systems, Man, and Cybernetics B* 2004;34(2):1108–16.
- [21] Goldberg D. *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley; 1989.

- [22] Kirpatrick S, Gerlatt CD, Vecchi MP. Optimization by simulated annealing. *Science* 1983;220:671–80.
- [23] Kirpatrick S. Optimization by simulated annealing—Quantitative studies. *Journal of Statistical Physics* 1984;34:975–86.
- [24] Yao X. A new simulated annealing algorithm. *International Journal of Computer Mathematics* 1995;56:161–8.
- [25] González J, Rojas I, Pomares H, Salmerón M, Merelo JJ. Web newspaper layout optimization using simulated annealing. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 2002;32(5):686–91.
- [26] Wang G, Ansari N. Optimal broadcast scheduling in packet radio networks using mean field annealing. *IEEE Journal of Selected Areas Communications* 1997;15(2):250–9.
- [27] Richardson J, Palmer M, Liepins G, Hilliard M. Some guidelines for genetic algorithms with penalty functions. In: *Proceedings of Third International Conference Genetic Algorithms and Applications*. California: Morgan Kaufmann; 1989.
- [28] Yao X, Wang F, Padmanabhan K, Salcedo-Sanz S. Hybrid evolutionary approaches to terminal assignment in communications networks. In: Krasnagor N, Smith J, Hart E, editors. *Recent Advances in Memetic Algorithms and related search technologies*, invited book chapter, 2004, in press.