PERGAMON

Computers & Operations Research 27 (2000) 287–303

computers & operations research

www.elsevier.com/locate/orms

A hybrid heuristic to solve a task allocation problem

Wun-Hwa Chen[a], Chin-Shien Lin[b,*]

[a]*Department of Business Administration, National Taiwan University, Taiwan, People's Republic of China*
[b]*Department of Business Administration, Providence University, Taiwan, People's Republic of China*

Received 1 September 1997; received in revised form 1 October 1998

Abstract

In this paper, we propose a hybrid method to solve a special version of task allocation problems. This hybrid method combines Tabu search for finding local optimal solutions and noising method for diversifying the search scheme to solve this problem. An experiment is conducted to test the hybrid method against other methods. Experimental results indicate that the hybrid method is efficient so far as the run time is concerned. Besides, it produced much better solutions. Out of 30 problem instances, the hybrid method obtained 23 best solutions for total cost and 27 for fixed cost among the tested algorithms.

Scope and purpose

This paper considers the problem of assigning tasks to processors such that the communication cost among the processors and the fixed costs of the processors are minimized. A hybrid method is proposed to solve this problem by combining Tabu search and noising method. The proposed hybrid method is proved to outperform the random method, Tabu search, noising method, and HBM algorithm in terms of run time and the quality of solutions.

Keywords: Task allocation; Tabu search; Noising method

1. Introduction

The purpose of task allocation problem is to assign tasks to processors such that some objective function can be maximized or minimized subject to some constraints. These are typical

* Corresponding author. Fax: 011-886-4-633-4191.
E-mail address: cslin@simon.pu.edu.tw (C.-S. Lin)

0305-0548/00/$ - see front matter
PII: S0305-0548(99)00045-3

problems arising in the design of distributed computing systems, and have many applications in the industry.

Billionnet et al. [1] and Sinclair [2] use branch-and-bound algorithms to solve the task allocation problem in a heterogeneous multi-processor system with no capacity constraints. Dutta et al. [3], Lo [4], and Ma et al. [5] use branch-and-bound techniques to address problems with various types of constraints such as preference constraints, exclusion constraints, bounds on the number of tasks assigned to each processor and capacity constraints. Sarje and Sagar [6] use an allocation technique to solve the uncapacitated homogeneous systems. Shen and Tsai [7] use a graph matching approach to solve the heterogeneous system problem under various constraints. Price and Krishnaprasad [8] use the clustering algorithm and the Banded Q Heuristic to solve the uncapacitated heterogeneous system design problem.

Recently Hadj-Alouane et al. [9] proposed a hybrid of Lagrangian relaxation and genetic algorithm (HBM) to solve a task allocation problem that arose in General Motor's assembly line [10] with capacity constraint and fixed cost associated with each processor. According to [11], HBM can find an optimal solution as long as it is run long enough. However, it usually takes a lot of time to get an optimal solution when the problem size is large. Since this problem is important for practical purposes, and the existing algorithm is time consuming, we propose another hybrid with local search to solve this problem.

Note that every searching scheme typically consists of two mechanisms. The first one is to find a local optimal solution. The second one is called the diversification mechanism, which is to find another local optimal solution after one has already been found. Our experience shows that the Tabu search [12,13] is good for the first mechanism and the noising method [14] is good for the second mechanism. In this paper, we present a method that combines these two searching algorithms into a hybrid one.

This paper is organized as follows. The task allocation problem along with its constraints is presented in Section 2. Basic issues related to the local search are discussed in Section 3. Section 4 describes implementation of random method, Tabu search, and noising method. The proposed hybrid method and its implementation is presented in Section 5. Section 6 contains experimental results. Finally, some concluding remarks are made in Section 7.

2. The task allocation problem

The task allocation problem investigated in this paper is defined as follows: We are given m tasks and k processors. The throughput requirement in KOP (thousand operations per second) of task i is denoted by v_i. The fixed cost and the capacity (in KOP) associated with processor j are denoted by f_j and c_j, respectively. e_{ij} is the cost of communication between task i and task j. This cost only occurs when tasks are assigned to two different processors and is only dependent on the amount of traffic between tasks i and j. Assume that $e_{ij} = e_{ji}$. We define $x_{ij} = 1$ if task i is assigned to processor j, and $x_{ij} = 0$ otherwise; $y_j = 1$ if processor j is used, and $y_j = 0$ otherwise. The task allocation

problem (P1) can be formulated as follows.

$$(P_1) \quad Z = \min \sum_{i=1}^{m-1} \sum_{i'=i+1}^{m} e_{ii'}\left(1 - \sum_{j=1}^{k} x_{ij}x_{i'j}\right) + \sum_{j=1}^{k} f_j y_j \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} v_i x_{ij} \leqslant c_j y_j, \quad j = 1, \ldots, k, \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} \leqslant m y_j, \quad j = 1, \ldots, k, \tag{3}$$

$$\sum_{j=1}^{k} x_{ij} = 1, \quad i = 1, \ldots, m, \tag{4}$$

$$x_{ij},\ y_j \in \{0, 1\} \quad \text{for all } i = 1, \ldots, m,\ j = 1, \ldots, k. \tag{5}$$

Constraint (2) guarantees that the total KOP requirements of all the tasks assigned to processor j does not exceed its capacity c_j. Constraint (3) ensures that a processor is purchased if it is allocated at least one task. Constraint (4) guarantees that each task is assigned to one and only one processor. Our job is to find the assignment of different tasks to different processors such that the above total cost is minimized. When the number of processors is only two, this problem can be transformed into a minimum cost cut problem [15] and solved efficiently by network flow techniques. However, for three or more processors, the problem has been shown to be NP-hard [16].

The task allocation problem in this paper was first formulated by Rao [10]. Hadj-Alouane et al. [9] have solved this problem by combining the Lagrangian relaxation method and genetic algorithm. In this paper, we present another hybrid of search methods to solve this problem. Essentially, there are three major steps in our approach. First, a relaxed initial solution is created. Then a local search is conducted. This search is a combination of Tabu search [12,13] and noising method [14]. Finally, a substitution technique is applied to improve the solutions.

3. Basic issues related to the local search

No matter what searching method is used, we must consider the following four issues:

1. How do we define a neighborhood?
2. How do we update the objective function values? Actually, we have to update the following three parameters:
   (a) communication cost
   (b) fixed cost
   (c) violation of capacity constraint
3. Outline of the local search.
4. How do we produce an initial solution?

3.1. Neighborhood

As the neighborhood significantly affects the solution quality, it is necessary to clarify how the neighborhood is defined. Let π be the current partition, and S the set of all defined moves. We use $N(\pi)$ to denote the neighborhood of π, i.e., the subset of moves in S applicable to π. For any move $s \in N(\pi)$, the new solution obtained by applying move s to π is called a neighbor of π. Only two different moving schemes concerned with the neighborhood, i.e., one-way move S_1 and two-way exchange S_2, are used in this research, where S_1 is the set of all moves which reassign one task from its current processor to another processor, and S_2 is the set of all moves which exchange two tasks assigned to two different processors.

There are more complicated moving schemes. As the number of tasks associated with a move increases, the efficiency of this moving scheme decreases. However, the quality of the solution obtained by using complicated moving scheme may be better. There is usually a trade-off between efficiency and the quality of the solution.

3.2. Updating the values of objective functions

The original problem (P_1) involves three constraints as pointed out in Section 2. To solve this problem, we adopt the same relaxing method as that in [9]. This is as follows:

$$(P_2) \quad Z = \min \sum_{i=1}^{m-1} \sum_{i'=i+1}^{m} e_{ii'} \left(1 - \sum_{j=1}^{k} x_{ij} x_{i'j}\right) + \sum_{j=1}^{k} f_j y_j + h(x, y), \tag{6}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} x_{ij} \leqslant m y_i, \quad j = 1, \ldots, k, \tag{7}$$

$$\sum_{j=1}^{k} x_{ij} = 1, \quad i = 1, \ldots, m, \tag{8}$$

$$x_{ij},\ y_j \in \{0, 1\} \quad \text{for all } i = 1, \ldots, m,\ j = 1, \ldots, k, \tag{9}$$

where $h(x, y) = \sum_{j=1}^{k} \lambda_j [\min(0, c_j y_j - \sum_{i=1}^{m} v_i x_{ij}]^2$ and λ_j is the multiplier for processor j. In other words, we use Eq. (6) as the objective function to guide our search, and the other constraints are satisfied in the algorithm.

Let (j, p, q) denote the one-way move which reassigns task j from its current processor p to processor q, where $p \neq q$, and $\Delta(j, p, q)$ denote the difference of Z between the partitions after and before the move (j, p, q). Then

$$\Delta(j, p, q) = g(j, p, q) + F(j, p, q) + V(j, p, q), \tag{10}$$

where $g(j, p, q)$, $F(j, p, q)$ and $V(j, p, q)$ are the differences of Z associated with (j, p, q) with respect to communication cost, fixed cost and violation of capacity constraint, respectively. Instead of recalculating Z after move (j, p, q), we can update Z by using $\Delta(j, p, q)$.

3.2.1. Communication cost

Let $\pi(j)$ denote the processor to which task j is assigned $g(j, p, q)$ can be calculated as follows [17]:

$$g(j, p, q) = \sum_{\{i|\pi(i)=p\}} e_{ij} - \sum_{\{k|\pi(k)=q\}} e_{kj}. \tag{11}$$

For each partition, we need to calculate the g for each task when it is reassigned to another processor before the next move is made, i.e., we need to calculate $\Delta(j, p, q)$, where $j = 1, 2, \ldots, m$, $p = \pi(j)$, $q \in \{1, 2, \ldots, k\}$ before we decide which move to take for the next step.

With task j being reassigned from processor p to processor q, the g's, which have been verified, can be incrementally updated as follows [17]:

$$g(j, q, p) = -g(j, p, q), \tag{12}$$

$$g(j, q, r) = g(j, p, r) - g(j, p, q), \quad r \neq p, q, \tag{13}$$

$$g(i, p, q) = g(i, p, q) - 2e_{ij}, \quad \forall \pi(i) = p, \tag{14}$$

$$g(i, p, r) = g(i, p, r) - e_{ij}, \quad \forall \pi(i) = p, r \neq p, q, \tag{15}$$

$$g(i, q, p) = g(i, q, p) + 2e_{ij}, \quad \forall \pi(i) = q, \tag{16}$$

$$g(i, q, r) = g(i, q, r) + e_{ij}, \quad \forall \pi(i) = q, r \neq p, q, \tag{17}$$

$$g(i, r, p) = g(i, r, p) + e_{ij}, \quad \forall \pi(i) \neq p, q, \tag{18}$$

$$g(i, r, q) = g(i, r, q) - e_{ij}, \quad \forall \pi(i) \neq p, q. \tag{19}$$

For a clear derivation of these equations, please refer to the appendix. The above g's on the left-hand side of the equal signs are the g's for all the tasks after move (j, p, q) and they can be updated from the g's before move (j, p, q) at the right-hand side of the equal signs. This updating scheme can save a lot of run time in calculating the communication cost.

3.2.2. Fixed cost

Any one-way move leading to the opening of a new processor q will increase the fixed cost by f_q, and any one-way move leading to the closing of a processor p will decrease the fixed cost by f_p. Let a_p and a_q denote the numbers of tasks already assigned to processor p and processor q, respectively. $F(j, p, q)$ is defined as follows:

$$F(j, p, q) = \begin{cases} f_q & \text{if } a_p > 1, \text{ and } a_q = 0, \\ -f_p & \text{if } a_p = 1, \text{ and } a_q > 0, \\ f_q - f_p & \text{if } a_p = 1, \text{ and } a_q = 0, \\ 0 & \text{if } a_p > 1, \text{ and } a_q > 0. \end{cases} \tag{20}$$

3.2.3. Violation of capacity constraint

The violation of capacity constraint $V(j, p, q)$ can be divided into two parts, $V(j, p, q) = V(p) + V(q)$, where $V(p)$ and $V(q)$ are violations of capacity constraints associated with processor p and processor q, respectively. Let b_p and b_q denote the summation of KOP of all tasks already assigned to processor p and processor q, respectively. All violations can then be calculated as follows:

$$V(p) = \begin{cases} -(b_p - c_p)^2 & \text{if } b_p > c_p \text{ and } b_p - v_i \leqslant c_p, \\ -((b_p - c_p)^2 - (b_p - v_i - c_p)^2) & \text{if } b_p > c_p \text{ and } b_p - v_i > c_p, \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

$$V(q) = \begin{cases} -(b_q + v_i - c_q)^2 & \text{if } b_q \leqslant c_q \text{ and } b_q + v_i > c_q, \\ (b_q - v_i - c_q)^2 - (b_q - c_q)^2 & \text{if } b_q > c_q \text{ and } b_q + v_i > c_q, \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

Let (i, j, p, q) denote the two-way exchange which exchanges task i in processor p and task j in processor q, and $\Delta(i, j, p, q)$ denote the difference of Z associated with (i, j, p, q). Since any two-way exchange can be viewed as two consecutive one-way moves, it can then be calculated as follows:

$$\Delta(i, j, p, q) = \Delta(i, p, q) + \Delta(j, q, p). \tag{23}$$

3.3. Outline of the local search scheme

Our local search consists of two phases: In Phase I, we try to keep the solutions feasible as much as possible. In Phase II, we try to refine the solutions with larger neighborhood. We use one-way move in Phase I and two-way exchange in Phase II.

3.4. Basic initial solutions

In the following, we shall introduce basic initial solutions that can be used in any searching method. It will be pointed out later that in our algorithm, we also use a relaxed version of these initial solutions.

A basic initial solution is generated as follows: First, processors are sorted in ascending order according to the unit cost, f_p/c_p. Then tasks are permuted into a sequence randomly. Next, tasks are assigned in this sequence to the processors until capacity constraints are violated. After this, the next processor in the sequence is chosen and the procedure is repeated until all the tasks are assigned. This method can always generate feasible solutions if the number of available processors is much larger than the number of processors required.

4. The basic implementation schemes of the random method, Tabu search and noising method

In addition to the HBM [9], we use three other searching methods as benchmarks to test our hybrid method. These three methods are the random method, Tabu search and noising method.

Random method is the simplest, Tabu search has been successfully applied to many problems [18–23], and noising method has a surprisingly good performance on solving clique partitioning of a graph [14].

4.1. Random method

The random method to solve our problem is presented as follows:

Step 1: Generate initial solutions.

Step 2: Use the deepest descent search with neighborhood S_1 in Phase I, and the deepest descent search with neighborhood S_2 in Phase II on solutions obtained in Phase I.

Step 3: Return the best-solution found.

The deepest descent search chooses the best move among the neighborhood until the solution found cannot be further improved. This method has the advantage of diversification if the number of initial solutions is large enough.

4.2. The Tabu search

Tabu search, developed by Glover [12,13], is a general purpose optimization technique designed to overcome local optimality. It has been successfully applied to a variety of combinatorial problems, which include maximum clique problem [18], traveling salesperson problem [21], quadratic assignment problem [22,23], the bandwidth packing problem [19], and the bin packing problems [20,24].

An important feature of Tabu search is the Tabu list (also called the short-term memory) which records those solution states that are not permitted at the current iteration. Restricting the next move to only non-Tabu state solutions has the role of preventing cycling and helping to overcome local optimality. However, this may result in rejecting some worthwhile moves. Therefore, a solution state remains Tabu only for a number of iterations. Besides, an aspiration level function is also introduced to override the Tabu status of a move if the move is considered "good enough" by the criterion implicit in the function. The search is continued until certain consecutive non-improving moves have passed, which is called the local intensification. After this, global diversification is implemented by use of a long-term search to visit regions not explored yet. The long-term search is implemented by starting another local search from a new partition obtained from disturbing the best-solution found by reassigning one task from its current processor to another processor.

Let *iter* denote the current iteration number. We use $list(j, p)$ to represent the Tabu list which records the latest iteration number when task j is assigned to processor p. It is initialized by setting $list(j, p) = 1$ for $j = 1, \ldots, m$, $p = \pi(j)$, and $list(j, p) = 0$ otherwise. Let *size* denote the Tabu size. The one-way move (j, p, q) is considered Tabu if and only if

$$iter - list(j, p) < size. \tag{24}$$

The two-way exchange (i, j, p, q) is considered Tabu if and only if

$$iter - list(j, p) < size \quad \text{and} \quad iter - list(i, q) < size. \tag{25}$$

The Tabu list is updated whenever a one-way move (j, p, q) is made by setting $list(j, q) = iter$, for $j = 1, \ldots, m$, and $q = \pi(j)$. A Tabu move becomes admissible if the aspiration level is attained. A simple aspiration level function is to override the Tabu status of a move if it gives an objective value strictly better than the best found so far. Our preliminary tests have shown that a better aspiration level function (adopted in this research) is to override the Tabu status of a move if it results in a smaller Z value than the best Z value obtained during that particular local search.

Let T_π denote the set of defined moves at π with Tabu status without satisfying the aspiration level criterion. Then the admissible moves at π form the set $N(\pi)$-T_π. Let Z_{best} denote the local optimum found during some particular local search, and Z^*_{best} denote the best local optimum found so far. The Tabu search used in our algorithm is presented as follows:

Step 1: Generate an initial solution.
Step 2: Initialize Tabu list.
Step 3: Set Z_{best} and Z^*_{best} to the initial objective function value.
Step 4: Repeat the best move in S_1 until $Z_{\text{best}} < Z^*_{\text{best}}$ for m times in Phase I.
  4.1. Calculate Δ for all the admissible one-way moves.
  4.2. Perform the best admissible one-way move.
  4.3. Update Tabu list and update the best solution if necessary.
Step 5: Repeat the best move in S_2 until $Z_{\text{best}} < Z^*_{\text{best}}$ for m times in Phase II.
  5.1. Calculate Δ for all the admissible two-way exchange.
  5.2. Perform the best admissible two-way exchange.
  5.3 Update Tabu list and update the best solution if necessary.
Step 6: Diversification:
  6.1. If long-term search has been implemented once go to step 7.
  6.2. Randomly reassign a task from its current processor to another processor on Z^*_{best}, go to step 4 (start another local search).
Step 7: Return the best solution found Z^*_{best}.

4.3. The noising method

The noising method was first proposed by Charon and Hudy [14] to solve clique partitioning of a graph. This method starts with an initial solution and repeats the following steps:

1. Add noise to the original data and run the local search.
2. Use the local optimum obtained from the noised data as the initial solution, and run the local search for the original data.

At each iteration, the amount of the added noise decreases until it reaches 0 at the last iteration. Local search of this method consists of the deepest descent search with neighborhood S_1 in Phase I and the deepest descent search with neighborhood S_2 in Phase II. The final solution is the best-solution found during the process. Instead of exploring areas randomly like the random method, noising method focuses on the neighborhood of the best-solution found.

Let Nb_Cycles denote the number of cycles, $Nb\_Iter\_Per\_Cycle$ the number of iterations for each cycle, and Max_Val the highest communication cost. A perturbation rate *Rate* is chosen at

each step. It decreases between two extreme values, *Rate_Max* and *Rate_Min*, with step size calculated as $stepsize = (Rate\_Max - Rate\_Min)/Nb\_Iter\_Per\_Cycle \times Nb\_Cycles$. The disturbed communication cost e'_{ij} is derived as follows:

$$e'_{ij} = \begin{cases} e_{ij} + r \times Rate \times Max\_Val & \text{if } e'_{ij} \geqslant 0, \\ 0 & \text{otherwise,} \end{cases} \tag{26}$$

where r is a random number generated between -1 and 1.

Let $G'(Rate)$ denote the noised problem after the original problem G is disturbed with perturbation rate *Rate*. The partition obtained from the local search on G' from current solution π is denoted by $Descent(\pi, G')$. The partition obtained from the local search on G from current solution π is denoted by $Descent(\pi, G)$. Let $F(\pi, G)$ represent the Z value of partition π on G, π^* the best partition obtained so far, and Z^* the corresponding Z value of π^*, i.e., $Z^* = F(\pi^*, G)$. The algorithm for the noising method is presented as follows:

Step 1: Generate initial solution π.
Step 2: $Rate = Rate\_Max$.
Step 3: $Stepsize = (Rate\_Max - Rate\_Min)/Nb\_Iter\_Per\_Cycle \times Nb\_Cycles$.
Step 4: $\pi^* = \pi$, $Z^* = F(\pi^*, G)$.
Step 5: Repeat the deepest descent search for *Nb_Cycles* times from π.
  5.1. Repeat the deepest descent search for *Nb_Iter_Per_Cycle* times from π.
    5.1.1. $G' = G'(Rate)$.
    5.1.2. $\pi = Descent(\pi, G')$.
    5.1.3. $\pi = Descent(\pi, G)$.
    5.1.4. $Z = F(\pi, G)$.
    5.1.5. Update Z^* and π^* if necessary.
    5.1.6. $Rate = Rate - Stepsize$.
  5.2. Set $\pi = \pi^*$.
Step 6: Return the best-solution found π^*.

The parameter values used in this algorithm were set as follows: $Rate\_Max = 0.8$, $Rate\_Min = 0.2$, $Nb\_Iter\_Per\_Cycle = 5$, and $Nb\_Cycles = [n/10]$, where $[n/10]$ is the rounded integer number of $n/10$. The total number of local search equals to $N/2$.

5. The hybrid method

Every searching scheme consists of two mechanisms: the local search to find a local optimum and the diversification to explore a new area. We like the local search to produce a good solution. That is, in our case, we like the local search to give a solution with the value Z as small as possible. Yet, we like our diversification scheme to be able to improve the solution found by the local search.

It is our experience that the Tabu search is efficient so far as the local search is concerned. In other words, it usually produces solutions with small Z values. The disadvantage is that it is not efficient in the diversification mechanism because the improvement by using the Tabu search is usually not significant. We also note that the noising method is good for diversification.

Our method therefore is a hybrid method. The Tabu search is used as the local searching mechanism and the noising method is used as the diversification mechanism.

5.1. The relaxed initial solutions

As indicated in Section 3.2, we have relaxed our original problem by incorporating capacity constraint into the objective function Z. We generate an initial solution that is not necessarily feasible. Our relaxed initial solution is found by modifying the method discussed in Section 3.4. Instead of satisfying the capacity constraints, we put all the tasks in the cheapest processor first. In the searching process, the solutions generated are not necessarily feasible either. Our experimental results however, indicated that all of the final solutions obtained were feasible. The relaxation method allows the search to visit more solution space. Consequently, it has a bigger chance to reach solutions of quality.

5.2. The improvement of a solution by processor substitution

Even though the processors chosen from the hybrid local search usually have smaller unit cost f_p/c_p than the others, the idle capacity of each chosen processor can make the utilization cost f_p/b_p large. This points to an inefficient utilization of cheap processors. Therefore, for each processor opened, its used capacity and fixed cost need to be checked against every available empty processor to look for cheaper alternative. If any empty processor can accommodate all the tasks in the particular used processor with smaller fixed cost without any violation of the capacity constraint, the used one will be replaced with the alternative one in our algorithm. This step, i.e., processor substitution, is used after the local search.

5.3. The implementation of hybrid method

The hybrid method algorithm is as follows.

Step 1: Put all the tasks in the cheapest processor first.

Step 2: Conduct one-way move (S_1) (Phase I) and two way exchange (S_2) (Phase II) by using the Tabu search on the initial solution.

Step 3: Conduct one-way move (S_1) (Phase I) and two way exchange (S_2) (Phase II) by using the noising method on the solution obtained in Step 2 to produce the local optimal solution.

Step 4: If no feasible solution is found, the next processor in sequence which has not been used is chosen. Put all the tasks into the new processor and go to Step 2.

Step 5: Improve the solution obtained in Step 3 by use of the processor substitution scheme.

Step 6: Return the solution.

6. Experimental results

For each test problem, 500 initial solutions were constructed for random method, 20 identical ones for both Tabu search and noising method. To test the feasibility of our methods, we ran one of the data sets in [9] with $m = 20$ and $k = 6$. The computational results are listed in Table 1, where

Table 1
Test result of the algorithms on a problem from the automobile microcomputer system

Algorithms	Solution
HBM	11,946
OSL	13,097
Random method	11,946
Tabu search	11,946
Noising method	11,946
Hybrid method	11,946

HBM represents Hadj-Alouane, Bean, and Murty's algorithm and OSL indicates the use of branch-and-bound routine of IBM's OSL package to find the solution. It can be seen that all algorithms, except OSL, found the same solution.

6.1. Randomized problem set

Our test data were generated randomly as follows. Let r denote a random number such that $0 < r < 1$. Since small density is good for algorithm testing, the density was set equal to 0.25, indicating a communication requirement was needed between task i and task j only if $r < 0.25$.

Let $c_{\min} = 40$ denote the minimum communication cost, and $c_{\text{int}} = 50$ the difference between the minimum and the maximum communication cost. Communication cost between tasks i and j was set as $c(i, j) = c_{\min} + [r \times c_{\text{int}}]$, where $[r \times c_{\text{int}}]$ denotes the rounded integer of $r \times c_{\text{int}}$.

Let $v_{\min} = 80$ denote the minimum value of KOP requirement among the tasks, and $v_{\text{int}} = 100$ the difference between the minimum and the maximum KOP requirement among the tasks. The KOP requirement of each task was $v(i) = v_{\min} + [r \times v_{\text{int}}]$.

Let n_p denote a random number which was determined as $n_p = 1 + [r \times m]$, and c_p denote the (KOP) capacity for processor p, based on the summation of KOP capacity for n_p times as the following:

$$c_p = \sum_{i=1}^{n_p} \{v_{\min} + [r_i \times v_{\text{int}}]\}, \tag{27}$$

where r_i is the random number for iteration i. The fixed cost was based on the KOP capacity and another variable, *ratio*,

$$f_p = c_p \times (1 + [ratio \times r]), \tag{28}$$

where *ratio* is the unit cost of the capacity (KOP). In our experiment, we set $m = 50$, $k = 20$ and 30. For each k, 15 problems were generated. For each set of 15 problems, 5 problems have ratio equal to 100, 5 equal to 50 and 5 equal to 10.

6.2. Experimental results

Instead of using the oscillating parameters like those used in HBM [9], we set the parameters the same during the search process. All the λ_j, $j = 1, \ldots, k$ were set to 100 based on preliminary tests that need all feasible solutions. Tables 2 and 3 show the experimental results of testing our hybrid method (HYBRID) against the random method (RM), Tabu search (TS), noising method (NM), and HBM.

In these two tables, BEST denotes the number of times a heuristic is the best or tied for the best, WORST denotes the number of times a heuristic is the worst or tied for the worst, DEVI denotes the average deviation from the best-solution found, which is calculated as $(solution - best)/best$, and TIME denotes the average time consumed. It can be seen that the hybrid method has the least percentage of deviation and the least run time for each problem type.

To gain more insights, we tried to show the ability of each algorithm to find the allocation with the least fixed cost. Table 4 shows the number of times, out of 30 problems, each algorithm got the smallest total cost and the least fixed cost among the solutions found. It can be seen that our hybrid method produced much better solutions. Furthermore, we can also see that if fixed costs were emphasized, our hybrid method was even more desirable.

Table 2
Comparison among the algorithms

K	Ratio		RM	TS	NM	HBM	HYBRID
20	100	BEST	1	0	0	0	4
		WORST	0	0	0	5	0
		DEVI	0.0222	0.0286	0.0152	0.2070	0.0001
		TIME	96.1400	96.7340	60.2760	1275.2560	28.5700
	50	BEST	0	0	1	0	4
		WORST	0	0	0	5	0
		DEVI	0.0609	0.0640	0.0606	0.1324	0.0005
		TIME	89.0800	90.5060	79.7420	1035.7440	28.7920
	10	BEST	0	0	1	0	4
		WORST	0	0	0	5	0
		DEVI	0.0146	0.0129	0.0117	0.0570	0.0009
		TIME	107.6620	107.3040	117.8160	687.6180	42.2260
30	100	BEST	1	0	1	0	3
		WORST	0	0	0	5	0
		DEVI	0.0120	0.0117	0.0103	0.2369	0.0006
		TIME	102.0080	108.8940	108.3180	693.7220	36.6900
	50	BEST	0	1	0	0	4
		WORST	0	0	0	5	0
		DEVI	0.0274	0.0243	0.0238	0.1672	0.0009
		TIME	124.0100	129.7660	135.9540	696.2620	40.4540
	10	BEST	0	0	1	0	4
		WORST	0	0	0	5	0
		DEVI	0.0117	0.0084	0.0106	0.0709	0.0009
		TIME	91.2980	100.0400	110.9620	694.9320	34.8520

Table 3
Summary of the computational results

	RM	TS	NM	HBM	HYBRID
BEST	2	1	4	0	23
WORST	0	0	0	30	0
DEVI	0.0248	0.0250	0.0220	0.1452	0.0006
TIME	101.6997	105.5406	102.1780	847.2557	35.2640

Table 4
Further insight summary

	RM	TS	NM	HBM	HYBRID
Total cost	2	1	4	0	23
Fixed cost	5	6	6	0	27

7. Conclusion

In this paper, we have shown how a hybrid method that combines Tabu search and noising method can solve a special version of the task allocation problem efficiently both in terms of run time and quality of solutions. We believe that our algorithm is good not only for this problem but also for many searching problems in general. We have already started applying our method to solve more complicated problems, pertaining to task allocation that included both capacity constraint and number of task constraints in addition to the inclusion of fixed costs.

Appendix A

To demonstrate the derivation of Eqs. (12)–(19) we take an allocation of 7 tasks and 3 processors as an example. Let G_1 be the allocation with tasks 1 and 3 assigned to processor 1, tasks 2, 6, and 7 to processor 2, and tasks 4 and 5 to processor 3. Communication cost among the processors occurs only when tasks having interactions with each other are assigned to different processors. We use C_i to denote the communication cost for allocation G_i. Since the communication cost is assumed to be symmetric, it is counted only once between two tasks. Therefore, $C_1 = e_{12} + e_{16} + e_{17} + e_{14} + e_{15} + e_{32} + e_{36} + e_{37} + e_{34} + e_{35} + e_{24} + e_{25} + e_{64} + e_{65} + e_{74} + e_{75}$, with each element demonstrated as a star in Table (a1) of Fig. 1.

Assume that the move under consideration is (3, 1, 3) which reassigns task 3 from processors 1 to 3. We use G_2 to indicate that the allocation after move (3, 1, 3) is made on G_1. The corresponding communication cost is given by $C_2 = e_{12} + e_{16} + e_{17} + e_{13} + e_{14} + e_{15} + e_{23} + e_{24} + e_{25} + e_{63} + e_{64} + e_{65} + e_{73} + e_{74} + e_{75}$, which is shown in Table (a2) of Fig. 1. Therefore,

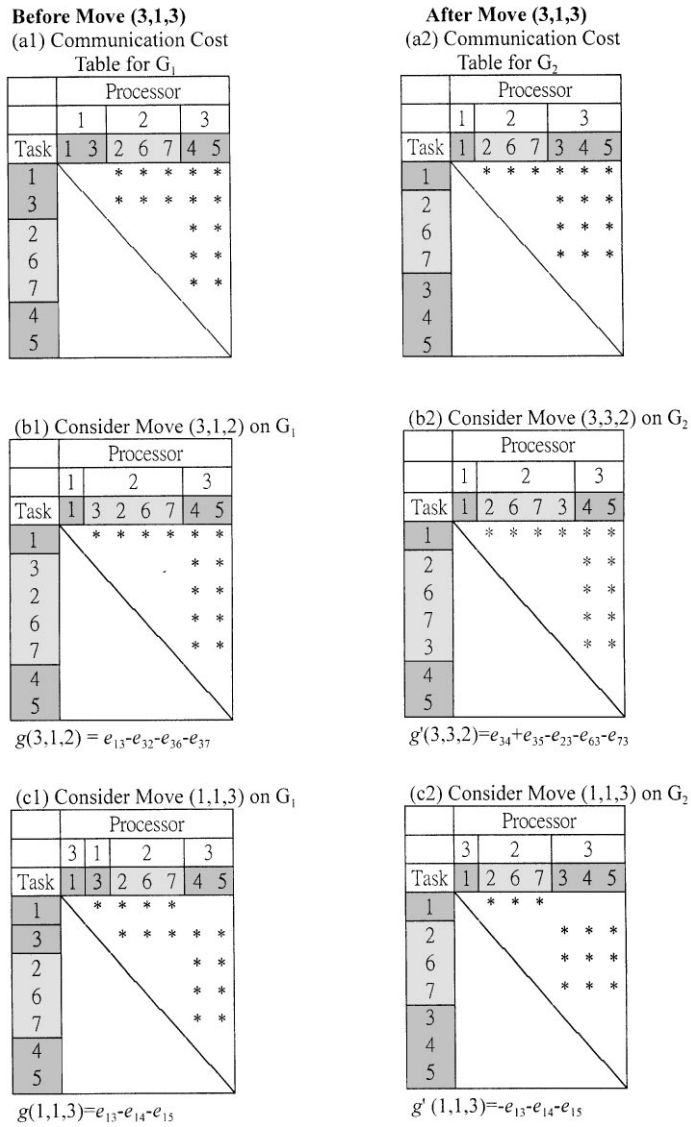

Fig. 1. The comparison of update function between before and after move (3, 1, 3).

$g(3, 1, 3) = C_2 - C_1 = e_{13} - e_{34} - e_{35}$, which can be generalized as Eq. (11), $g(j, p, q) = \sum_{\{t|\pi(t)=p\}} e_{ij} - \sum_{\{k|\pi(k)=q\}} e_{kj}$. Note that $e_{kj} = e_{jk}$.

To show the validity of the update scheme of Eqs. (12)–(19), we will show the calculations of these equations first and then the completeness of the scheme.

A.1. Calculations

To avoid confusion, we use $g'(j, p, q)$ to indicate the difference of communication cost between before and after move (j, p, q) for G_2. The purpose is to update $g'(j, p, q)$ by using $g(j, p, q)$.

Table 5
The detailed calculation of each cell of the update table

Before move (3.1.3)		After move (3.1.3)		Equations	Eq #
$g(j,p,q)$	Value	$g(i,p,q)$	Value		
$g(1,1,2)$	$e_{13}-e_{12}-e_{16}-e_{17}$	$g(1,1,2)$	$-e_{12}-e_{16}-e_{17}$	$g(1,1,2)=g(1,1,2)-e_{13}$	(15)
$g(1,1,3)$	$e_{13}-e_{14}-e_{15}$	$g(1,1,3)$	$-e_{13}-e_{14}-e_{15}$	$g(1,1,3)=g(1,1,3)-2e_{13}$	(14)
$g(3,1,2)$	$e_{13}-e_{32}-e_{36}-e_{37}$	$g(3,3,1)$	$-e_{13}+e_{34}+e_{35}$	$g(3,3,1)=g(3,1,3)$	(12)
$g(3,1,3)$	$e_{13}-e_{34}-e_{35}$	$g(3,3,2)$	$e_{23}+e_{34}+e_{35}-e_{36}-e_{37}$	$g(3,3,2)=g(3,1,2)-g(3,1,3)$	(13)
$g(2,2,1)$	$-e_{12}-e_{23}+e_{26}+e_{27}$	$g(2,2,1)$	$-e_{12}+e_{26}+e_{27}$	$g(2,2,1)=g(2,2,1)+e_{23}$	(18)
$g(2,2,3)$	$-e_{24}-e_{25}+e_{26}+e_{27}$	$g(2,2,3)$	$-e_{23}-e_{24}-e_{25}+e_{26}+e_{27}$	$g(2,2,3)=g(2,2,3)-e_{23}$	(19)
$g(6,2,1)$	$e_{26}+e_{67}-e_{16}-e_{36}$	$g(6,2,1)$	$e_{26}+e_{67}-e_{16}$	$g(6,2,1)=g(6,2,1)+e_{36}$	(18)
$g(6,2,3)$	$e_{26}+e_{67}-e_{46}-e_{56}$	$g(6,2,3)$	$e_{26}+e_{67}-e_{46}-e_{56}-e_{36}$	$g(6,2,3)=g(6,2,3)-e_{36}$	(19)
$g(7,2,1)$	$e_{27}-e_{17}-e_{37}+e_{67}$	$g(7,2,1)$	$e_{27}-e_{17}+e_{67}$	$g(7,2,1)=g(7,2,1)+e_{37}$	(18)
$g(7,2,3)$	$e_{67}+e_{27}-e_{47}-e_{57}$	$g(7,2,3)$	$e_{67}+e_{27}-e_{47}-e_{57}-e_{37}$	$g(7,2,3)=g(7,2,3)-e_{37}$	(19)
$g(4,3,1)$	$-e_{14}-e_{34}-e_{45}$	$g(4,3,1)$	$-e_{14}+e_{34}+e_{45}$	$g(4,3,1)=g(4,3,1)+2e_{34}$	(16)
$g(4,3,2)$	$-e_{24}+e_{45}-e_{46}-e_{47}$	$g(4,3,2)$	$-e_{24}+e_{34}+e_{45}-e_{46}-e_{47}$	$g(4,3,2)=g(4,3,2)+e_{34}$	(17)
$g(5,3,1)$	$e_{45}-e_{15}-e_{35}$	$g(5,3,1)$	$e_{35}+e_{45}-e_{15}$	$g(5,3,1)=g(5,3,1)+2e_{35}$	(16)
$g(5,3,2)$	$e_{45}-e_{25}-e_{65}-e_{75}$	$g(5,3,2)$	$e_{45}+e_{35}-e_{25}-e_{65}-e_{75}$	$g(5,3,2)=g(5,3,2)+e_{35}$	(17)

Consider move (3, 3, 1) for G_2, which is the reverse of move (3, 1, 3) for G_1. Therefore, $g'(3,3,1) = -g(3,1,3)$, which can be generalized as Eq. (12), $g'(j,q,p) = -g(j,p,q)$.

Consider move (3, 1, 2) for G_1 and move (3, 3, 2) for G_2. $g(3,1,2) = e_{13}-e_{32}-e_{36}-e_{37}$, which can be obtained by computing the difference of Tables (b1) and (a1) of Fig. 1. $g'(3,3,2) = e_{34}+e_{35}-e_{23}-e_{63}-e_{73}$, which can be obtained by computing the difference of Tables (b2) and (a2) of Fig. 1. $g'(3,3,2) = g(3,1,2) - g(3,1,3)$, which can be generalized as Eq. (13), $g'(j,q,r) = g(j,p,r) - g(j,p,q)$, where $r \neq p, q$.

Consider move (1, 1, 3) for both G_1 and G_2. $g(1,1,3) = e_{13}-e_{14}-e_{15}$, which can be obtained by computing the difference of Tables (c1) and (a1) of Fig. 1. $g'(1,1,3) = -e_{13}-e_{14}-e_{15}$, which can be obtained by computing the difference of Tables (c2) and (a2). $g'(1,1,3) = g(1,1,3) - 2e_{13}$, which can be generalized as Eq. (14), $g(i,p,q) = g(i,p,q) - 2e_{ij}$, $\forall \pi(i) = p$. The calculations of all the other equations follow the same logic as Eq. (14). All the possible moves, their communication cost values, and their corresponding update equations are listed in Table 5.

A.2. Completeness

All the communication cost update functions before move (3, 1, 3) are shown in Table (a) of Fig. 2. In addition to the communication cost update functions, their corresponding equation numbers are also listed in Table (b) of Fig. 2. It can be seen that the communication cost update functions of all the possible moves are contained in Eqs. (12)–(19).

(a) Communication Cost Update Table before Move (3,1,3)

	Processor		
Task	1	2	3
1	0	g(1,1,2)	g(1,1,3)
3	0	g(3,1,2)	g(3,1,3)
2	g(2,2,1)	0	g(2,2,3)
6	g(6,2,1)	0	g(6,2,3)
7	g(7,2,1)	0	g(7,2,3)
4	g(4,3,1)	g(4,3,2)	0
5	g(5,3,1)	g(5,3,2)	0

(b) Communication Cost Update Table after Move (3,1,3)

	Processor		
Task	1	2	3
1	0 -	g(1,1,2) (15)	g(1,1,3) (14)
2	g(2,2,1) (18)	0 -	g(2,2,3) (19)
6	g(6,2,1) (18)	0 -	g(2,2,3) (19)
7	g(7,2,1) (18)	0 -	g(7,2,3) (19)
3	g(3,3,1) (12)	g(3,3,2) (13)	0 -
4	g(4,3,1) (16)	g(4,3,2) (17)	0 -
5	g(5,3,1) (16)	g(5,3,2) (17)	0 -

Note: The numbers in parenthesis below the g's are the corresponding equation number

Fig. 2. Communication cost update table before and after move (3, 1, 3).

References

[1] Billionnet A. Costa MC, Sutter A. An efficient algorithm for a task allocation problem. Journal of the Association for Computing Machinery Quarterly 1984;29:147–50.
[2] Sinclair JB. Efficient computation of optimal assignments for distributed tasks. Journal of Parallel and Distributed Computing 1987;4:342–62.
[3] Dutta A., Koehler G, Whinston A. On optimal allocation in a distributed processing environment. Management Science 1982;28(8):839–53.
[4] Lo VM. Heuristic algorithm for task assignment in distributed systems. Proceedings of the International Conference on Distributed Computing Systems. San Francisco, CA, 1984:30–9.
[5] Ma PR, Lee EYS, Tsuchiya M. A task allocation model for distributed computing systems. IEEE Transactions on Computers 1982;31(1):41–7.
[6] Sarje AK, Sagar G. Heuristic model for task allocation in distributed computer systems. IEE Proceedings. Part E, Computers and Digital Techniques 1991;138:313–8.

[7] Shen CC, Tsai WH. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. IEEE Transactions on Computers 1985;34:197–203.
[8] Price C, Krishnaprasad S. Software allocation models for distributed computing systems. Proceedings of the International Conference on Distributed Computing Systems, San Francisco, CA, 1984;40–8.
[9] Hadj-Alouane AB, Bean JC, Murty KG. A hybrid genetic/optimization algorithm for a task allocation problem. Technical Report 93-30, Department of Industrial and Operations Engineering, University of Michigan, 1993.
[10] Rao KN. Optimal synthesis of microcomputers for GM vehicles. Technical Report, 1992.
[11] Hadj-Alouane AB, Bean JC,. A genetic algorithm for the multiple-choice integer program. Technical Report 92-50, Department of Industrial and Operations Engineering, University of Michigan, 1992.
[12] Glover F. Tabu search – Part I. ORSA Journal of Computing 1989;1:190–206.
[13] Glover F. Tabu search – Part II. ORSA Journal of Computing 1990;2:4–32.
[14] Charon I, Hudry O. The noising method: a new method for combinatorial optimization. Operations Research Letters 1993;14(3):133–7.
[15] Stone HS. Multiprocessor scheduling with the aid of network flow algorithms. IEEE Transactions on Software Engineering 1977;3:85–93.
[16] Murty KG. Operations research: deterministic optimization models. Englewood Cliffs, NJ, Prentice-Hall, 1994.
[17] Tao L, Zhao Y. Multi-way graph partition by stochastic probe. Computers and Operations Research 1993;20(3):321–47.
[18] Gendreau M, Salvail L, Soriano P. Solving the maximum clique problem using a tabu search approach. Discrete Applied Maths 1990.
[19] Glover F, Laguna M. Bandwidth packing: a tabu search approach. Presented at the First Workshop on Combinatorial Optimization in Science and Technology, DIMACS Technical Report 91-18, RUTCOR Report 3–91, 1991.
[20] Glover F, Hubscher R. Bin packing with tabu search. Technical Report, Graduate School of Business Administration, University of Colorado at Bolder, 1991.
[21] Knox J. An application of Tabu search to the symmetric travelling salesman problem. Ph.D. Thesis, 1988.
[22] Skorin-Kapov J. Tabu search applied to the quadratic assignment problem. ORSA Journal of Computing 1990; 2: 33–45.
[23] Taillard E. Robust taboo search for the quadratic assignment problem. Parallel Computing 1991;17:443–5.
[24] Johnson DS. Fast Algorithms for Bin-Packing. Journal of Computer and System Science 1974;8:272–314.

Wun-Hwa Chen is an Associate Professor in Business Administration Department of National Taiwan University. He holds the degree of B.S. in Management Science from National Chiao-Taung University, and an MBA and Ph.D. in Management Science from New York State University at Buffalo. His research interests are in the areas of algorithm design, financial engineering, and production and operation management.

Chin-Shien Lin is an Associate Professor in Business Administration Department at Providence University presently. He holds the degree of B.S. in Management Science from National Chiao-Taung University, an MBA from Tatung Institute of Technology and a Ph.D. in Decision Science from Washington State University. He has research and teaching interests in algorithm design, financial engineering, and project management.