Parallel Ant Systems applied to the Multiple Knapsack Problem

Marcelo Cena, Guillermo Leguizamón
Proyecto UNSL N° 338403*
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
5700 - San Luis - Argentina
e-mail: {mcena,legui}@unsl.edu.ar
Fax: +54 652 30224

Keywords: parallel programming, distributed systems, ant colony systems, combinatorial optimization, multiple knapsack problem.

Abstract

Interesting real world combinatorial problems are NP-complete and many of them are hard to solve by using traditional methods. However, several heuristic methods have been developed in order to obtain timely suboptimal solutions. Most of those heuristic methods are also naturally suitable for a parallel implementation and consequently, an additional improvement on the response time can be obtained. One way of increasing the computational power is by using multiple processors operating together on a single problem. The overall problem is split into parts, each of which is operated by a separate processor in parallel. Unfortunately problems cannot be divided perfectly into separate parts and interaction is necessary between the parts like data transfer and process synchronization. However, substantial improvement can be achieved, depending on the problem and the amount of parallelism in the problem. Our work aims to exploit the capability of a distributed computing environment by using PVM and implementing a parallel version of an Ant System for solving the Multiple Knapsack Problem (MKP). An Ant System (a distributed algorithm) is a set of agents working independently and cooperating sporadically in a common problem solving activity. Regarding the above characteristics, an Ant System can be naturally considered as a *nearly embarrassingly parallel computation*. The proposed parallel implementations of an Ant System are based on two different approaches, *static and dynamic task assignment*. The computational study involves processors of different velocities and several MKP test cases of different sizes and difficulties (tight and loose constraints). The performance on the response time is measured by two indexes, *Speedup Factor and Efficiency* when is compared to a serial version of an Ant System. The results obtained show the potential power of exploiting the parallelism underlying in an Ant System regarding the good quality of the results and a remarkable decreasing on the computation time.

*The research group is supported by U.N.S.L and ANPCyT (Agencia Nacional para la Promoción de la Ciencia y la Tecnología)

Parallel Ant Systems applied to the Multiple Knapsack Problem

1 Introduction

Parallel programming uses multiple computers, or computers with multiple internal processors, to solve a problem at a greater speed than using a single computer. Areas requiring great computational speed include optimization, numerical modeling and simulation of problems in science and engineering, which often need huge repetitive calculations on large amounts of data to give valid results. However, apart from having an algorithmic solution and the amount of memory required, the execution time is a key issue. One way of increasing the computational power is by using multiple processors operating together on a single problem. The overall problem is split into parts, each of which is operated upon by a separate processor in parallel. Unfortunately problems cannot be divided perfectly into separate parts and interaction is necessary between the parts like data transfer and process synchronization. However substantial improvement can be achieved, depending on the problem and the amount of parallelism in the problem. Moreover, for obtaining the potential for increased speed on an existing problem, the use of multiple computers/processors often allows a larger or more precise instance of a problem be solved in a reasonable time.

There exist several types of computer platforms suitable for implementing parallel applications. Our work aims to exploit the capability of a distributed computing environment by using PVM package (Parallel Virtual Machine), and implementing a parallel version of an Ant System for solving the Multiple Knapsack Problem (MKP). The proposed parallel implementations of an Ant System are based on two different approaches, *static and dynamic task assignment*. The computational study involves processors of different power and several MKP test cases of different sizes and difficulties (tight and loose constraints). The performance on the response time is measured by two indexes, *Speedup and Efficiency* when is compared to a *serial version* of an Ant System. The results obtained show the potential power of exploiting the parallelism underlying in an Ant System regarding the good quality of the results and a remarkable decreasing on the computation time.

The remainder of the paper is organized in the following way. In the next sections the classical and adapted version of an Ant System are given. Next, two approaches for task assignment in a distributed system, the experiments performed and results obtained are shown. Finally, the conclusions are exposed.

2 A Brief Description of an Ant System

An Ant System (AS)[2,3,7,8,9] is a new meta-heuristic for hard combinatorial optimization problems. This meta-heuristic is a new member in the class of meta-heuristic derived from nature[4] that includes Genetic Algorithms, Neural Networks, Simulated Annealing, Evolution Strategies, etc. AS is an approach based on the result of low-level interaction among many cooperating simple agents that are not aware of their cooperative behavior[7]. Each simple agent is called *ant* and the Ant System (a distributed algorithm) is a set of ants working independently and cooperating sporadically in a common problem solving activity. Regarding the above characteristics, an Ant System can be naturally considered as a *nearly embarrassingly parallel computation*[14]. Since earlier applications of Ant Systems [7], plenty of work has been done in this area by applying Ant Systems to solve ordering problems like Traveling Salesman Problem (TSP), Bin Packing Problem and Quadratic Assignment Problem [2,3,4,7,8,9,11]. In [5] an Ant System was adapted in order to solve non-ordering or subset problems. The adapted AS[5] shown to be efficient to solve MKP. In this section, the *original* AS for solving TSP (an ordering problem) and the *adapted* AS for solving MKP (a subset problem) are presented.

2.1 The Ant System for Ordering Problems

Given a set of n cities, the Traveling Salesman Problem [12,13] is to find a closed path that visits every city exactly once *(tour)* with minimal total length. i.e.

$$mininize\ COST(i_1, ..., i_n) = \sum_{j=1}^{n-1} d(C_{i_j}, C_{i_{j+1}}) + d(C_{i_n}, C_{i_1})$$

where $d(C_x, C_y)$ is the distance between city x and city y.

The Ant-cycle approach for solving TSP proposed in [7] is briefly presented here.

Let $b_i(t)$ $(i = 1, ..., n)$ be the number of ants in city i at time t and let $Na = \sum_{i=1}^{n} b_i(t)$ be the total number of ants in the system. Let $\tau_{ij}(t + n)$ be the intensity of trial on $path_{ij}$ at time $t + n$, given by

$$\tau_{ij}(t + n) = (\rho\tau_{ij}(t) + \Delta\tau_{ij}(t, t + n)) \quad (\mathbf{2.1.1})$$

where ρ is such $(1 - \rho)$ is the coefficient of evaporation $(0 < \rho < 1)$.

$\Delta\tau_{ij}(t, t+n) = \sum_{k=1}^{Na} \Delta\tau_{ij}^{k}(t, t+n)$, where $\Delta\tau_{ij}^{k}(t, t+n)$ is the quantity per unit of length of trial substance (pheromone in real ants) laid on $path_{ij}$ by the k^{th} ant between time t and $t+n$ and is given by the following formula:

$$\Delta\tau_{ij}^{k}(t, t+n) = \begin{cases} \frac{Q}{L_k} & \textit{if } k^{th} \textit{ ant uses edge } (i,j) \textit{ in its tour} \\ 0 & \textit{otherwise} \end{cases} \quad \textbf{(2.1.2)}$$

where Q is a constant and L_k is the tour length of the k^{th} ant. The intensity of trial at time 0, $\tau_{ij}(0)$, is set to a randomly chosen value.

During the next $(t + n)$ tour the probability to visit city j when being at city i is

$$P_{ij}(t, k) = \begin{cases} \frac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}}{\sum_{h \in allowed_k} \tau_{ih}^{\alpha}(t)\eta_{ih}^{\beta}} & j \in allowed_k \\ 0 & \textit{otherwise} \end{cases} \quad \textbf{(2.1.3)}$$

where $allowed_k$ is a set of cities not visited for that particular tour and η_{ij} is a local heuristic. For TSP the parameter η_{ij}, called visibility, is $\frac{1}{d(C_i, C_j)}$ [7].

The parameters α and β allow control on the relative importance of trail versus visibility. Hence, the transition probability is a trade-off between visibility, which says that close cities should be chosen with high probability, and trail intensity, that says that if on $path_{ij}$ there is a lot of traffic then is it highly profitable .

A data structure, called *tabu list*, is associated to each ant in order to avoid that ants visit a city more than once, i.e. $tabu_k$ list maintain a set of visited cities up to time t by the k^{th} ant. Therefore $allowed_k$ set can be defined as follows: $allowed_k = \{j/j \notin tabu_k\}$. When a tour is completed the $tabu_k$ *list* $(k = 1..Na)$ is emptied and every ant is free again to choose an alternative tour for the next cycle.

By using the above definitions, we describe the Ant-cycle algorithm:

Initialize
for t=1 to number of cycles do
  for k=1 to Na do
    Repeat Until k has completed a tour
      - Select city j to be visited next with probability P_{ij} given by equation **(2.1.3)**
    end
    Calculate the length L_k of the tour generated by ant k
  end
  Save the best solution so far
  Update the trail levels τ_{ij} on all paths according to equation **(2.1.1)**
end
Print the best solution found

2.2 The Ant System for Subset Problems

The Multiple Knapsack Problem which is an example of a subset problem can be formulated [1,12,13] as follows:

$$maximise \sum_{j=1}^{n} p_j x_j$$

$$subject\ to\ \sum_{j=1}^{n} r_{ij} x_j \leq c_i \quad i = 1, ..., m \quad (\mathbf{2.2.1})$$
$$x_i \in \{0, 1\} \quad j = 1, ...n$$

Each of the m constrains described is called a knapsack constrain, so the MKP is also called the *m-dimensional Knapsack Problem*. Let $I = \{1, ..., m\}$ and $J = \{1, ..., n\}$, with $c_i \geq 0$ for all $i \in I$, $j \in J$. A *well-stated* MKP assumes that $p_j > 0$ and $r_{ij} \leq c_i \leq \sum_{j=1}^{n} r_{ij}$ for all $i \in I$, $j \in J$, since any violation of these conditions will result in some x_j being fixed to zero and/or some constrains being eliminated. Note that the $(r_{ij})_{mxn}$ matrix and $(c_i)_m$ vector are both non-negative which distinguishes this problem from general 0-1 linear integer programming problem. Many practical problems can be formulated as a MKP, for example, the capital budgeting problem where project j has profit p_j and consumes r_{ij} units of resource i. The goal is to find a subset of the n projects such that the total profit is maximized and all resource constrains are satisfied. For solving MKP, the ants[5] look for a subset of n items or projects (see MKP formulation) such that the total profit is maximized and all resource constrains are satisfied.

Let $bi\ (i = 1, ..., n)$ be the number of ants incorporating in the solution the item i at time $t = 0$ and let $Na = \sum_{i=1}^{n} b_i$ be the total number of ants in the system. Since in MKP there are not paths, the *intensity of trial* and *local heuristic* are computed in a slightly different way. Let $\tau_i(t + N_{max})$ be the *intensity of trial* on item i at time $t + N_{max}$, given by

$$\tau_i(t + N_{max}) = \rho\tau_i(t) + \Delta\tau_i(t, t + N_{max}) \quad \textbf{(2.2.2)}$$

where ρ is such $(1 - \rho)$ is the *coefficient of evaporation* and N_{max} is the maximum number of items qualified to be added to some solution by some ant.

$$\Delta\tau_i(t, t+N_{max}) = \sum_{i=1}^{Na} \Delta\tau_i^k(t, t+N_{max}),$$

where $\Delta\tau_i^k(t, t+N_{max})$ is the quantity per unit of length of trial substance (pheromone in real ants) laid on item i by the k^{th} ant between time t and $t+N_{max}$ and is given by the following formula:

$$\Delta\tau_i^k(t, t+n) = \begin{cases} \frac{L_k}{Q} & \textit{if } k^{th} \textit{ ant incorporates item } i \\ 0 & \textit{otherwise} \end{cases} \qquad \textbf{(2.2.3)}$$

where Q is a constant and L_k is the profit *(objective function in [Eq. 2.2.1]* obtained by the k^{th} ant. The intensity of trial at time 0, $\tau_i(0)$, is set to a randomly chosen value. During the next $(t+N_{max})$ item incorporation the probability for selecting item i by the k^{th} ant, in order to complete the $solution_k$ is:

$$P_i(t,k) = \begin{cases} \frac{\tau_i^\alpha(t)\eta_i^\beta(k)}{\sum_{j\in allowed_k} \tau_j^\alpha(t)\eta_j^\beta(k)} & i \in allowed_k \\ 0 & \textit{otherwise} \end{cases} \qquad \textbf{(2.2.4)}$$

where $allowed_k$ is a set of items still not considered by the k^{th} ant and the $solution_k$ satisfies all constraints if some of them are added. The parameter $\eta_i(k)$, called *pseudo-utility*, is the local heuristic. We chose $\eta_i(k)$ as follows:

$$\begin{aligned} &\eta_i(k) = \tfrac{p_i}{\bar{\delta}_i(k)}; \qquad \bar{\delta}_i\,(k) = \tfrac{\sum_{j=1}^{m} \delta_{ij}(k)}{m} \\ &\delta_{ij}(k) = \tfrac{r_{ji}}{(c_j - u_j(k))}; \quad u_j(k) = \textstyle\sum_{l\in solution_k} r_{jl} \end{aligned} \qquad (\mathbf{2.2.5})$$

Where $(c_j - u_j(k))$ is the remaining amount to reach the boundary of constraint j, $r_{ji} \leq (c_j - u_j(k))$ and $\delta_{ij}(k) \in (0,1]$, is the $tightness$ of item i on constraint j when item i is added to $solution_k$. Consequently the *pseudo-utility* $\eta_i(k)$ turns larger as $\bar{\delta}_i\,(k)$ *(tightness average)* turns smaller. The parameters α and β, as for TSP, allow control on the relative importance of *trail* versus the local heuristic *(pseudo-utility for MKP)*. Hence, the transition probability is a trade-off between pseudo-utility, which says that more profitable items that uses less resources should be chosen with high probability, and trail intensity, that says that if item i is part of a lot of solutions, then is it highly desirable.

A data structure, called *tabu list*, is also associated to each ant in order to avoid that ants choice an item more than once, i.e. $tabu_k$ list maintain the set of added items up to time t by the k^{th} ant. This list also maintains $u_j(k)$ $(j = 1..m)$ in order to reduce the required computational time. The $allowed_k$ set can be defined as follows:

$$allowed_k = \{j / j \notin tabu_k \textit{ and } solution_k \textit{ with item } j \textit{ added satisfies all constraints}\}$$

When all ants add to the solutions as many items as they can, $tabu_k$ *list* $(k = 1..Na)$ is emptied and every ant is free again to choose an alternative *subset of items* for the next cycle.

The outline of the adapted Ant-cycle algorithm for subset problems follows:

Initialize
for t=1 to number of cycles do
  for k=1 to Na do
    Repeat Until $allowed_k$ *is empty*
      - Select item i to be incorporated with probability P_i *given by equation (2.2.4)*
    end
    Calculate L_k*, the profit obtained by ant* k
    Save the best solution so far
  end
  Update the trail levels τ_i *on all items according to equation (2.2.2)*
end
Print the best solution found

3 Parallel Ant System for solving MKP

An Ant System is a distributed algorithm where multiple independent agents cooperate with each others for solving a common problem. The algorithm runs for a fixed number of cycles, after each cycle the agents interchange some information (cooperate) in order to *learn* which is the more promising area of the search space. During each cycle, all agents can execute independently since no interaction between them is needed at all. Based on the above features, two approaches, *static and dynamic task assignment*[14] were considered in order to accomplish a Parallel Ant System (a nearly embarrassingly parallel program). In the *static task assignment* approach, the problem is divided into a fixed number of processes to be executed in parallel. In addition, the processes are simply distributed among the available processors without any discussion on the effects of the types of processors and their speeds. However, it may be that some of processors will complete their task before others and became idle because the work is unevenly divided or some processors operate faster than others (or both situations). On the other hand, the *dynamic task assignment* approach intend to spread the tasks evenly across the processors in order to maximize the efficiency. Under both of the implemented approaches, there exist a **Master** and p **Slaves** processes, all of them distributed on available processors **(Figure 1)** in the system. The master process is in charge of updating the trail (τ) according the solutions found by the slave processes after each cycle of the algorithm. Every slave process is capable to separately access the instance of MKP to be tested.

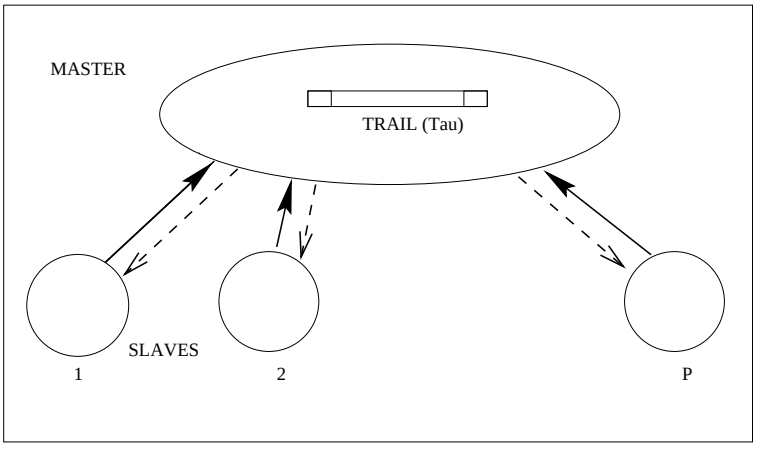

Figure 1: *Layout for Static and Dynamic Task Assignment Approaches*

3.1 Static Task Assignment

Assuming that the Ant System involves Na ants and there exist p processors available in the system, Na/p ants are assigned to each processor. The master and slave processes and a brief explanation concerning the purpose of each message are outlined as follows:

Messages

$INIT\_PROCESS$: - The Ant system parameters
- Request for the first solutions (Na/p)

NEW_τ: - Modified trail *(cooperation stage)*
- Request for (Na/p) solutions

$END\_PROCESS$: - Finish the slave

Master Process

Initialize p slaves
Send to every slave: $INIT\_PROCESS$
do
{ Receive one solution from every slave
  Update $\Delta\tau_i$ regarding p received solutions
  Choose the Best Solution from the p best solutions
  Save the best solution so far
  if ($Ec < MaxCycles$) then /* Ec stands for Elapsed cycles */
      Update $\tau_i(t, t + Nmax)$ $(i : 1..Na)$ [Eq. 2.2.2]
      Send to every slave: NEW_τ
      Ec++
  else
      Send to every slave: $END\_PROCESS$
      Print out the best solution found
      Exit
  endif
} while ($FOREVER$)

Slave Process

do {Receive order from Master
    switch (order) {
    case $INIT\_PROCESS$:
        - Recover the Ant system parameters
        - read_MKP_instance();
        - Generate Na/p solutions
        - Send back the best out of Na/p solutions
    case NEW_τ:
        - Recover new trail
        - Generate Na/p solutions
        - Send back the best out of Na/p solutions
    case $END\_PROCESS$:
        - Finish the slave
    }
} while ($FOREVER$)

3.2 Dynamic Task Assignment (Work-Pool)

By this approach, the master process maintains a Work Pool of agents to be processed *(Centralized Dynamic Load Balancing)*[14]. Thus, the Work Pool represents the inactive agents waiting for an idle processor. The master and slave processes and a brief explanation concerning the purpose of each message are oulined as follows:

Messages

$INIT\_PROCESS$: - The Ant system parameters
- Request for the first solution

$PROCESS\_SOL$: - Solution request

NEW_τ: - Modified trail (cooperation stage)

$END\_PROCESS$: - Finish the slave

Master Process

Initialize Work Pool with Na ants (tasks)
Initialize p slaves /* Each slave processes one ant */
Send to every slave: $INIT\_PROCESS$
do {
  $Working\_Slaves = \{1, ..., p\}$ /* Set of slaves still processing */
  while ($Working\_Slaves$ is not empty)
  {
    Receive a solution (from some slave, say k)
    Update $\Delta\tau$ regarding the Received Solution
    Choose the best solution so far
    if (Work Pool is empty)
      $Working\_Slaves = Working\_Slaves - \{k\}$
    else
      Send to slave k: $PROCESS\_SOL$ /*activate agent i from Work Pool */
      Delete agent i from Work Pool
    endif
  }
  if ($Ec < MaxCycles$) then /* Ec stands for Elapsed cycles */
    Update $\tau_i(t, t + N_{max})$ $(i : 1..Na)$ [Eq. 2.2.2]
    Initialize Work Pool with $Na - p$ ants (tasks)
    Send to every slave: NEW_τ
    Ec++
  else
    Send to every slave: $END\_PROCESS$
    Print Out the Best Solution so far
    Exit
  endif
} while($FOREVER$)

Slave Process

do
{
  Receive order from Master
  switch (order) {
  case $INIT\_PROCESS$:
    - Recover the Ant system parameters

- read_MKP_instance()
- Generate one solution
- Send back the solution

case $PROCESS\_SOL$:
- Generate one solution
- Send back the solution

case NEW_τ:
- Recover new trail
- Generate one solution
- Send back the solution

case $END\_PROCESS$:
- Finish the slave

}
} while ($FOREVER$)

4 Computational Study

Six instances of MKP taken from [1] were considered in our experiments. The Ant Parallel Systems were tested, at the begining, on two processors P_1 and P_2 (Sun Sparc workstations having a similar capacity) by using the optimal parameter setting found in [5]. ***Table I*** shows the results of our experiments expressed in terms of the average of *Speedup Factor* (regarding a Serial Ant System running on processor P_1), *Efficiency* ***(Eqs. 4.1)***[14] and Percentage of Hits out of 10 runs for each instance.

$$Speedup(n) = \frac{T_s}{T_n} \qquad \begin{array}{l} T_s = \text{Execution time using a single processor system} \\ T_n = \text{Execution time using a system with } n \text{ processors} \end{array} \qquad \textbf{(4.1)}$$

$$Efficiency(n) = \frac{Speedup(n)}{n} \qquad \text{i.e the fraction of time the processors are being used on the computation}$$

Columns *PAS-S* and *PAS-D* stand for Static and Dynamic Task Assignment in a Parallel Ant System respectively.

	PAS-S			*PAS-D*		
Instance	Speedup	Efficiency	%hits	Speedup	Efficiency	%hits
1	1.81	0.905	80	1.70	0.85	80
2	1.63	0.815	70	1.67	0.835	70
3	1.77	0.885	100	1.65	0.825	90
4	1.84	0.92	70	1.6	0.8	80
5	1.68	0.84	90	1.72	0.86	90
6	1.89	0.945	100	1.8	0.9	100

Table I. *PASs running on processors P_1 and P_2*

As we observe in ***Table I***, both approaches showed similar behaviour. The parallel systems performed very well regarding the *Speedup*, *Efficency* and additionally, the quality of the results (*%hits*) compared to those results found in [5,6,10]. However, an additional study was necessary in order to establish some difference between *PAS-S* and *PAS-D* approaches. In a second experiment, a third processor P_3 (slower than P_1 and P_2) was incoporated to the Parallel Virtual

Machine. ***Table II*** shows the *Speedup* obtained by the two approaches. It is important remarking that the Serial Ant System was run on processor P_1 (one of the faster processors). Columns $PVM_i = \{P_j\}$ stand for the set of processors conforming the Parallel Virtual Machine.

The values in ***Table II*** indicate that *PAS-D* performed much better than *PAS-S* running on both environments, PVM_1 and PVM_2 respectively. Although the the work is evenly divided, the relative velocities of each processor are no considered by *PAS-S* approach. For example, *PAS-S* obtained for each instance considered a *Speedup* less than 1 running on environment PVM_1 since processor P_3 is the "bottle neck" of the system and turning P_1 idle most part of the time. On the other hand, PAS-D took advantage of the Work Pool of tasks when some processor turns idle. A similar situation is observed in PVM_2 where the inclusion of processor P_2 (the other faster processor) produced only a little improvement on the *Speedup* obtained by *PAS-S*.

	$PVM_1 = \{P_1, P_3\}$		$PVM_2 = \{P_1, P_3, P_2\}$	
Instance	*PAS-S*	*PAS-D*	*PAS-S*	*PAS-D*
1	0.66	1.5	0.82	1.88
2	0.6	1.37	0.755	2.47
3	0.7	1.56	0.86	2.56
4	0.72	1.33	1.11	2.42
5	0.76	1.44	1.13	2.84
6	0.7	1.41	1.145	2.48

Table II. *Values for environments PVM_1 and PVM_2*

It is remarkable that for this kind of highly distributed algorithm (an Ant System) the profit achieved by using PAS-D approach, is evident. However, *PAS-S* also achieve a good performance running on an environment of processors having a similar power (see ***Table I***).

Although the results obtained show clearly the difference between *PAS-S* and *PAS-D* running on different environments, it is not evident to carry out a straightforward analysis of the performance of the parallel ant systems, either *PAS-S* or *PAS-D*, due to they are stochastic algorithms where their computation time strongly depend on the seed given as input to generate random numbers. For example, in image processing, a task can be divided in a small number of tasks and each one processes a *fixed number* of pixels (some image partition). On the other hand, an Ant System is a set of small tasks (independent agents) conforming a distributed algorithm and the purpose is to distribute those small tasks on available processors in a particular parallel platform. However, when any parallel approach is applied to some instance of MKP, the *Speedup* achieved by augmenting the number of processors varied slightly (varying the seed) due to the variation on the number of items incorporated by each ant per cycle of the algorithm which performs accordingly the initial seed. It is also possible that for some instances of MKP, the parallel systems running on an heterogeneous platform are able to obtain a *Speedup Factor* very close to the optimal one .

5 Conclusions

An Ant System is a class of distributed algorithm which can be naturally considered as a *nearly embarrassingly parallel computation*. The two proposed approaches showed that the explicit parallelism involved in an Ant System can be easily exploited by using networked workstations. However, there exist some considerations to take in account when a particular approach will be used: *Work division and processors power conforming the parallel environment*. Also, it is worth remarking that the applications developed in PVM are portable enough to run on different parallel platform without major changes.

References

[1] Beasley, J. - ”OR-Library: Distributing Test Problems by Electronic Mail” - e-mail: o.rlibrary@ic.ac.uk

[2] Bilchev, G. - ”Evolutionary Metaphors for the Bin Packing Problem” - Proceedings of the Fifth Annual Conference on Evolutionary Programming. San Diego, California - USA, 1996.

[3] Bilchev, G.; et al. - ”The Ant Colony Metaphor for Searching Continuous Design Spaces” - Published in Evolutionary Computation, selected papers from AISB Workshop, Sheffield UK, April 1995. Edited by T.C. Fogarty.

[4] Bullnheimer, B.; et al. - ”A New Rank Based Version of the Ant System - A Computational Study”. University of Vienna. April 1997.

[5] Cena, M.; et al. - ”The Ant Colony Metaphor for Multiple Knapsack Problem” 3th CACiC. La Plata, Argentina. October 1997

[6] Chu, Paul; et al. - ”A Genetic Algorithm for the Multi-constraint Knapsack Problem”. http://mscmga.ms.ic.ac.uk/pchu/pchu.html

[7] Dorigo, M. ; et al. - ”Distributed Optimization by Ant Colonies” - Proceedings of ECAL91. Elsevier Publishing, pp 134-142.

[8] Dorigo, M.; et al. - ”An investigation of some properties of an Ant algorithm” - Proceedings on the Parallel Problem Solving from Nature Conference. Elsevier Publishing, 1992.

[9] Dorigo, M.; et al. - ”A study of some properties of ANT-Q” - Published in Proceedings of PPSN IV. Springer-Verlag, 1996.

[10] Khuri, Sami; et al. - ”The Zero/One Multiple Knapsack Problem and Genetic Algorithms”. ACM Symposium of Applied Computation ’94.

[11] Maniezzo, V.; et al. - ”The Ant System Applied to the Quadratic Assignment Problem”. Technical Report 94/28. IRIDIA, Universite Libre de Bruxelles, Belgium.

[12] Nemhauser, G.; et al. - ”Integer and Combinatorial Optimization”. John Wiley & Sons, Inc. 1988.

[13] Papadimitriou, C. - ”Combinatorial Optimization: Algorithms and Complexity”. Prentice Hall. 1982..

[14] Wilkinson, Barry; et al. ”Parallel Programming Techniques and Application Using Networked Workstations”. Preliminary Draft. Prentice Hall, 1997.