Distributed Task Scheduling through a Swarm Intelligence Approach

Paulo R. Ferreira Jr.[1] **and Ana L. C. Bazzan**[2]

[1] Departamento de Informática – IFM – Universidade Federal de Pelotas (UFPel)
Caixa Postal 354 – CEP 96010-900 – Pelotas – RS – Brasil

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – CEP 91.501-970 – Porto Alegre – RS – Brazil

paulo.ferreira@ufpel.edu.br, bazzan@inf.ufrgs.br

Abstract. *This paper addresses distributed task scheduling problems as a distributed version of the Resource-Constrained Project Scheduling Problem (RCPSP). We propose and evaluate a novel approach for the distributed RCPSP based on theoretical models of division of labor in social insect colonies. Our approach uses a probabilistic decision-making model based on the social insect tendency to perform certain tasks, and was implemented as an algorithm called Swarm-RCPSP. We show that the results of the Swarm-RCPSP algorithm are better than those obtained with a distributed greedy algorithm, are not very far from the best-known solutions, and have the advantage of being computed in a distributed manner, which is an important issue when dealing with multiagent systems.*

1. Introduction

Several real-world applications often have to solve inherently distributed scheduling problems. Those applications include disaster rescues, grid environments, project management, meeting schedules and sensor networks. They demand distributed solutions that are able to achieve reliable and flexible results. As large-scale scenarios with hundreds of resources and tasks become ubiquitous, new challenges arise. Optimal approaches are not able to tackle large-scale distributed scheduling problems because of the computational and communication requirements associated with the search for optimality. On the other hand, approximate approaches to deal with this type of problem remain to be fully investigated.

We describe an extension of the general scheduling problem called resource-constrained project scheduling problem (RCPSP). Since the RCPSP is a classical optimization problem, several centralized approaches have been proposed to solve it. In the RCPSP, project activities must be scheduled to minimize the total makespan. Tasks have interdependence constraints, such as precedence, that must be satisfied. In addition, tasks demand a minimum number of resources in order to be scheduled. In our extension, called distributed RCPSP (RCPSP), resources are owned by agents. Agents must coordinate their actions to schedule tasks cooperatively.

We propose a novel approximate approach for distributed task scheduling based on the division of labor in social insect colonies. A social insect colony is an example of a self-organizing biological system, in which there is ample evidence of ecological success

despite the lack of centralized coordination. In these colonies, hundreds of thousands of insects adapt to the changes in the environment and to the needs of the colony using plasticity of division of labor [Robinson 1992]. We adopt a multiagent perspective, in which agents have limited perception and autonomous behavior. Cooperative agents running the implementation of our approach are able to schedule tasks using little communication and computation. We empirically evaluated this approach by comparing it with a distributed greedy approach, both benchmarked by solutions available in the literature [Kolisch and Sprecher 1997].

This paper is organized as follows: Section 2 presents the distributed RCPSP. Section 3 introduces the proposed approach and the algorithm that implements it. The empirical evaluation of our approach is described in Section 4, in which the results obtained are discussed. Related work is discussed in Section 5, while Section 6 presents the conclusion and future directions for this work.

2. Problem Statement

The Resource-Constrained Project Scheduling Problem (RCPSP) is an optimization problem in which project tasks, originally called activities, have to be scheduled so that total makespan is minimized. Tasks are interdependent, and there are precedence constraints that have to be satisfied. In addition, task scheduling demands resources, which are limited due to the limitations of total availability of resources.

Considering $\mathcal{J}$ as the set of tasks, and $\mathcal{R}$ as the set of resources available to be used in these tasks, each task $j \in \mathcal{J}$ has a time duration of d_j and requires an amount r_j of each $r \in \mathcal{R}$ resource per time unit. The total quantity associated with each $r \in \mathcal{R}$ resource is given by $Q_r > 0$. The set of direct predecessors of j is given by P_j.

Each scheduling is represented by $\mathcal{S}_n$, in which $\mathcal{S}_n = \{start_n(1), start_n(2), ..., start_n(j), ..., start_n(|\mathcal{J}|)\}$, and $start_n(j)$ is the unit of time at which the j task begins in the $\mathcal{S}_n$ scheduling. Therefore, the time unit at which the task is completed is calculated as $end(j) = start(j) + d_j$. The time unit at which a scheduling begins is the same time unit for the beginning of the task that starts earlier: $min\{start_n(j) \ \forall \ j \ \in \ \mathcal{J}\}$. The time unit at which the scheduling ends is the same time unit for the end of the last task: $max\{start_n(j) + d_j \ \forall \ j \ \in \ \mathcal{J}\}$. Total scheduling makespan is the difference between the time units at which scheduling begins and ends.

To be valid, an $\mathcal{S}_n$ scheduling must satisfy the constraints mentioned in the beginning of this section, which are: The j task cannot begin before all its predecessors have ended: $start_n(j) \geq start_n(h) + d_h \quad \forall h \in P_j$; and the resource constraints must be satisfied, which means that for each t time unit, the sum of the quantity of each r resource demanded by all the scheduled tasks cannot exceed the total availability of these resources. Considering $\mathcal{J}_n^t = \{j \in \mathcal{J} | start_n(j) \leq t < start_n(j) + d_j\}$ as the set of the simultaneously scheduled tasks: $\sum_{j \in \mathcal{J}_n^t} r_j <= Q_r$.

The distributed RCPSP is a modification of the RCPSP that changes it into a distributed problem. In the distributed RCPSP, each resource unit r belongs to one and only one agent. In other words, each agent has only one unit of a $r \in \mathcal{R}$ resource, and may schedule only one task at each time unit. Therefore, an agent i, which has one unit of the $r \in \mathcal{R}$ resource, has a k_{ij} competence to schedule a task j according to the availability of the resource r demanded by this task, $k_{ij} = 1$ if $r_j \neq 0$ or $k_{ij} = 0$ otherwise.

Table 1. Duration d_j, demanded resources $\mathcal{R}_j$ and precedence P for each task in the distributed RCPSP example instance.

	Tasks				
	A	**B**	**C**	**D**	**E**
d	4	9	2	2	2
$\mathcal{R}$	{1,0}	{1,0}	{0,2}	{2,0}	{2,1}
P	-	-	B	-	A

In the RCPSP distributed version, tasks are divided into as many subtasks as the quantity of resources demanded by the tasks. Therefore, each task j is divided into a set of subtasks $\mathcal{B}_j$, whose number is equivalent to the sum of the r_j quantity of each resource $r \in \mathcal{R}$ that the task demands. Therefore, the size of the $|\mathcal{B}_j|$ set of subtasks is equal to the total quantity of resources that the task demands in order to be scheduled.

The distribution of the RCPSP, as proposed here, requires that two other constraints be satisfied for an $\mathcal{S}_n$ scheduling to be valid: Considering $\mathcal{I}_{j_n}$ as the set of agents that allocates the j subtasks in the $\mathcal{S}_n$ scheduling, all the $j_m \in \mathcal{B}_j$ subtasks should be scheduled: $|\mathcal{I}_{j_n}| = |\mathcal{B}_j|$; and if $start_n(j_m)$ is a unit of time at which a subtask $j_m \in \mathcal{B}_j$ was scheduled in the $\mathcal{S}_n$ scheduling, all the other subtasks in j, that is, its sister tasks, should be scheduled at the same time unit t by all agents $i \in \mathcal{I}_{j_n}$: $\{start_n(j_1) = start_n(j_2), start_n(j_2) = start_n(j_3), ..., start_n(j_{|\mathcal{B}_j|-1}) = start_n(j_{|\mathcal{B}_j|})\}$.

Therefore, agents should attempt, in a distributed way, to minimize scheduling time for the tasks perceived, and should satisfy all the constraints described in this section to validate such scheduling. The coordination between agents is, consequently, a complex question. Agents must coordinate actions to allocate tasks simultaneously in a number at least equal to the number of tasks' demanded resources. Moreover, to make scheduling possible, agents must schedule tasks in an order that satisfies task precedence constraints.

In a simple instance of the distributed RCPSP with 4 agents $\mathcal{I} = (V, X, Y, Z)$ and 5 tasks $\mathcal{J} = (A, B, C, D, E)$, whose specifications are defined in Table 1, each task has a specific duration, as shown on the first line of the table. The sets presented on the second line of the table indicate the quantity for each resource $\mathcal{R} = \{R1, R2\}$ demanded by each task. The task precedence constraints are presented on the last line of the table.

Task division into a set of subtasks according to the resources demanded by the tasks, as described above, determine the following: Tasks A and B remain unchanged, and demand one unit of the $R1$ resource each one; task C is divided into two subtasks ($C1$ and $C2$), which demand one unit of the $R2$ resource each; task D is divided into two subtasks ($D1$ and $D2$), which demand one unit of the $R1$ resource each; and task E is divided into three subtasks ($E1$, $E2$ and $E3$); $E1$ and $E2$ demand one unit of the $R1$ resource each, and $E3$ demands one unit of the $R2$ resource. Agents V and X have one unit of the $R1$ resource, and agents Y and Z have one unit of the $R2$ resource.

Figure 1 shows a possible scheduling for the instance under discussion. The interdependence between tasks was satisfied, and the C and E subtasks were scheduled after their predecessors. Simultaneous scheduling of the C, D and E subtasks was also adequately performed by the agents.

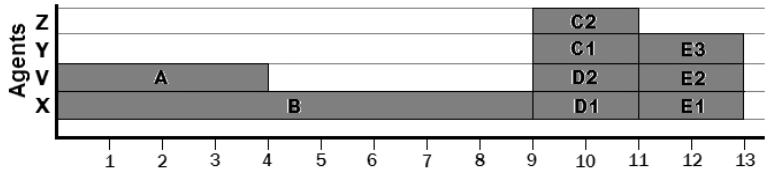

Figure 1. Possible scheduling for the distributed RCPSP example instance.

3. Proposed Approach

3.1. Decision Making

The theoretical basis of the proposed approach is the plasticity of division of labor in social insect colonies [Robinson 1992]. This plasticity emerges from the interaction between individuals and between individuals and environment, and uses simple communication patterns without any central coordination. Tasks for individuals that belong to a colony should be allocated in a way that ensures that environment variations are absorbed and that the amount of energy spent by the colony is the lowest possible.

In our approach, agents decide probabilistically which tasks to schedule according to the theoretical model of response threshold described by [Bonabeau et al. 1999]. The tasks produce a stimulus for the agents that, because of their internal response thresholds, have different tendencies to schedule each task. When the tasks are perceived by the agents, they should analyze them and decide which ones to schedule using this specific tendency. This is achieved by using a roulette system, and the tendency is used as the probability that determines whether the agent will or will not schedule the task under analysis.

The tendency $T_{\theta_{ij}}(s_{ij})$ of the agent i to engage in scheduling task j is determined by the relation between this task stimulus and the agent's internal threshold, as shown in Equation 1. A distributed process is under analysis here; therefore, the task stimuli are agent specific, and the same task may stimulate each of the agents differently.

$$T_{\theta_{ij}}(s_{ij}) = \frac{s_{ij}^2}{s_{ij}^2 + \theta_{ij}^2} \tag{1}$$

Where:

s_{ij} stimulus associated with task j for agent i;
θ_{ij} internal threshold of agent i for task j.

If the values for the stimulus and the agent's internal threshold are within the $[0, 1]$ interval, the tendency values also vary within the same interval. In the equation described above, the agent that has the greatest competence to schedule a certain task is the agent that has the lowest internal threshold for that task.

As the agents have a limited quantity of resources to use in each scheduling, decision making is strongly affected by the order in which the tasks are analyzed. Agents make decisions for all the tasks by analyzing them one by one. This process may be repeated until all the tasks have been selected or until agent resources are used up. In this case, the tasks that remain unanalyzed or were not selected will be disregarded. Agents

organize scheduling as each task is selected, and always choose to place the task in a position as early as possible in its timeline or in agreement with some specific start time for this task.

Therefore, the order of tasks plays a fundamental role in decision making. All the tasks perceived should be analyzed in an order that satisfies the constraints imposed by the distributed RCPSP definition. A certain task that has another task as its predecessor can only be analyzed by the agents if its predecessor was successfully scheduled before. Agents always try to communicate their decisions so that information about allocations can be inferred by the other agents.

Moreover, when a task is divided into subtasks, these subtasks have to be scheduled simultaneously. When agents schedule one of the subtasks, they send a message to the other agents and ask them to give priority to the scheduling of the other subtasks not yet scheduled. The other agents should try to schedule subtasks at the same start time at which the first subtask was scheduled. Agents that receive this message first analyze the tasks received in relation to the tasks that they have perceived. Their order ensures that the analysis is given priority in this type of case.

This mechanism was inspired by how social insects ask for help to perform tasks that are beyond their individual capacity. Many worker ants look for prey or larger and heavier food items that one single individual cannot carry. These ants use cooperative transport to deal with large items. If it cannot retrieve certain objects, a worker ant may recruit its nestmates using simple forms of communication [Kube and Bonabeau 2000]. Nestmates in the vicinity are attracted by a simple call for help and rapidly join efforts with the individual that asked for help in an act of cooperation.

3.2. Stimulus Adaptation

In the approach proposed here, agents schedule tasks in an attempt to minimize the makespan and to satisfy all the constraints and characteristics of different scenarios. Tasks produce a stimulus for the agent, as described above. This stimulus may be fixed, according to the importance of the task in a certain scenario, or variable in order to increase the agent's tendency to choose it for some specific reason. We propose that the stimulus is periodically increased for the tasks that are part of the scheduling and have the earliest end time. The update bias $\Gamma(j)$ is used to determine by how much each task stimulus should be increased given a specific scheduling. This bias is lower as the task's start time. As stimuli increase due to the updating, agents have a greater stimulus to schedule tasks in the same order as they were previously scheduled. $\Gamma(j) = 1 - \frac{start_{n^*}(j)}{|\mathcal{S}_{n^*}|}$ where $start_{n^*}(j)$ is the unit of time in which task j begins in the best scheduling, $\mathcal{S}_{n^*}$; $n \in \mathcal{S}$, and $|\mathcal{S}_{n^*}|$ is total time duration of the best scheduling.

The $s_{ij}(t)$ local stimulus associated with task j and agent i at time t is calculated as: $s_{ij}(t) = s_{ij}(t-\mu) * \delta + \Gamma(j) * \alpha$. Here $s_{ij}(t-\mu)$ is the local stimulus of task j in round $t-\mu$; δ is the constant to decrease the stimulus value along time; $\Gamma(j)$ is the update bias for task j; and α is the discount rate to decrease the impact of the update on stimulus. In this case, stimuli decrease along time according to the constant δ, and only increase according to the update bias $\Gamma(j)$, calculated according to the best scheduling obtained by agents.

This stimulus update occurs with frequency μ. Therefore, during μ schedules,

agents store the solutions obtained and use the best solution as the basis for the calculation of the update bias $\Gamma(j)$.

We assume that $\mathcal{M}_n$ is the set of tasks in scheduling n and that $start_n(j)$ is the function that returns the time at which task j starts. $\mathcal{S}$ is the set of n schedulings where $\mathcal{S}_n = \{start_n(1), start_n(2), ..., start_n(j), ..., start_n(|\mathcal{M}_n|)\}$. The total duration $|\mathcal{S}_n|$ of $\mathcal{S}_n$ is given by the time at which the last task was completed. This is formalized as: $|\mathcal{S}_n| = max_{j=1}^{|\mathcal{M}_n|}[start_n(j) + d_j]$ where: $start_n(j)$ is the time at which allocation started for task j in scheduling n; $|\mathcal{M}_n|$ is the number of scheduled tasks in scheduling n; d_j is the duration of task j.

The performance of a complete scheduling is measured according to the number of tasks allocated and the total makespan.The best scheduling $\mathcal{S}_{n^*}$ in $\mathcal{S}$ is the one that has the shortest time duration in the set of schedules that have the greatest number of tasks (those that have their indices in $\mathcal{N}^*$).

4. Experiments and Results

We have built an algorithm called Swarm-RCPSP which implements the proposed approach. The experiments with the Swarm-RCPSP were conducted in an abstract simulator written in Java. These simulations were configured according to three different instances of the RCPSP. Such instances belong to the data set "j120" of the Project Scheduling Problem Library (PSPLIB) [Kolisch and Sprecher 1997]. These instances were originally designed for centralized problems, but, as discussed in section 2, may be mapped as distributed problems.

The three instances used for the experiments in this study are among the largest: "j1201_1", "j12030_1" and "j12060_1". 200 tasks are used in each one. In the three cases tasks are strongly interdependent. Each task has a different number of successors in the discrete interval $[1, 3]$. The tasks also have the duration of a discrete interval, which is $[1, 10]$ in this case; Four different resources, called $R1$, $R2$, $R3$ and $R4$, are used. Each task demands a quantity of each resource in the discrete interval $[0, 10]$. The instances under analysis also differ in the number of different resources that they demand. In instance "j1201_1", all tasks demand a quantity that is different from zero for only one resource. In instance "j12030_1", the number of different resources varies within the discrete interval $[1, 3]$; and in instance "j12060_1", a task may demand that all the resources be involved in its scheduling.

Considering the process of dividing the RCPSP tasks into subtasks in the distributed RCPSP, discussed in Section 2, there is a total of 646, 1334 and 2554 tasks for the "j1201_1", "j12030_1" and "j12060_1" instances. Each of these tasks demands one unit of a certain resource according to the characteristics of the supertasks of each specific instance. Once each agent has one resource, the number of agents necessary to deal with these instances as a distributed RCPSP is equal to the sum of the available quantities of each resource. This sum is equal to 48, 128 and 191 for "j1201_1", "j12030_1" and "j12060_1", and the last one has the greatest availability in the data set.

The complexity of the problem increases substantially from one instance to another. The number of interdependences between tasks, the total number of agents, and the number of agent classes involved in each task scheduling increase simultaneously.

A scheduling attempt is successful only if the tasks were scheduled according to all the constraints that characterize the distributed RCPSP, as detailed in section 2.

For the sake of comparison with our algorithm (Swarm-RCPSP) we use centralized solutions as well as greedy distributed strategies. Regarding the latter an algorithm equivalent to the Swarm-RCPSP was implemented that is a simple variation of the Swarm-RCPSP, but has neither the stimulus variation mechanism nor the tendency-oriented probabilistic decision making. Agents choose the tasks whenever possible. Similarly to what occurs with the Swarm-RCPSP, these tasks are scheduled in the order that they are chosen according to the time availability in the agent's scheduling. We chose to implement this strategy to be able to analyze and to compare performance associated with the search for distributed solutions. This way it is possible to see the impact of the use of the mechanisms that make up the approach on the quality of the solution obtained for the RCPSP.

Moreover, PSPLIB has the best known solutions for the instances used in this study [Kolisch and Sprecher 1997], which will be used as comparison parameters to evaluate the general performance of the Swarm-RCPSP and of the greedy algorithm. The best solutions for the "j1201_1" and "j12030_1" instances were obtained in the studies conducted by [Brucker and Knust 2000], and for the "j12060_1" instance, in the work conducted by [Luo et al. 2006]. Their solutions, however, were determined in a centralized way, and used different heuristics with the sole objective of searching for a better solution than the best-known solution.

To ensure that the results obtained with some distributed algorithms may be appropriately compared with the best-known solution in a centralized way, all the tasks are perceived by at least one agent in the experiments conducted in our study. Neighbor recruitment to deal with the interrelations between tasks ensures that a task will be shared if it is perceived by one agent and requires that more agents schedule its sister tasks. Therefore, the other agents will have their perception extended to include these tasks, which is important to ensure that the system obtains a complete scheduling. For the same reason, messages are sent to all agents simultaneously (broadcasting).

First, several experiments were conducted with the three instances used in this study to determine the combination of values for the parameters in Equation 1. The same values were used for the internal threshold of all agents. Similarly, the same values were used for the stimulus for all agents in all tasks. These values, determined empirically, lead agents to the determination of the best scheduling. Details of these experiments are not reported here. The following initial optimal values were used: 0.043 for stimulus and 0.03 for internal threshold in the "j1201_1" instance; 0.026 for stimulus and 0.016 for internal threshold in the "j12030_1" instance; and 0.012 for stimulus and 0.008 for internal threshold in the "j12060_1" instance.

The results using the Swarm-RCPSP are compared with those using a distributed greedy approach and with the best-known solutions of the RCPSP instances used in this study [Brucker and Knust 2000, Luo et al. 2006]. Figure 2 shows the comparison between the total scheduling makespan using the Swarm-RCPSP (with and without stimulus adaptation), the centralized approximated algorithms, and the greedy algorithm for the instances in the experiment.

The greedy algorithm had the worst results; its total makespan was two to six

times greater than the best-known total makespan for these schedulings, depending on the complexity of the experiment instance. The Swarm-RCPSP schedulings had results that were at the most twice the best-known makespan. The algorithm was significantly more efficient than the greedy approach, particularly for the most complex instances.

It was expected that the results of the Swarm-RCPSP would not be equivalent to the best-known centralized solution. This happens because we deal with the problem in a distributed manner. It must be emphasized that the differences between results using the Swarm-RCPSP and the best-known solution do not vary substantially as the instances become more complex.

Figure 2 shows that stimulus adaptation was effective for the "j1201_1" instance; its use resulted in a 10% improvement of scheduling makespan in comparison with results of instances without stimulus adaptation. Moreover, as discussed earlier, stimulus adaptation ensured that the algorithm always achieved a valid solution (complete scheduling) for the "j1201_1" instance, which did not occur when it was not used. In the other instances, which were much more complex and involved a larger number of agents, no significant differences were seen, and the improvement reached 5% in average.

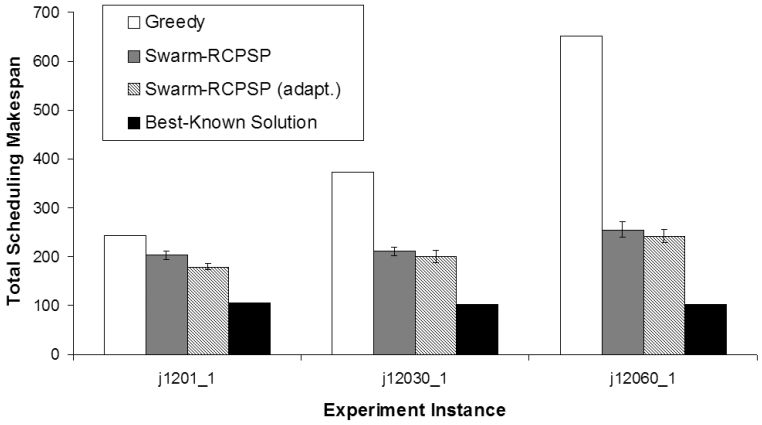

Figure 2. Comparing the total scheduling makespan.

Figure 3 shows the number of messages exchanged by the agents running the Swarm-RCPSP, with and without stimulus adaptation, and the greedy algorithm for the experiment instances. The agents using the greedy algorithm exchange fewer messages than when using the Swarm-RCPSP. The Swarm-RCPSP showed a significant decrease in message exchanges (35% in average) for the "j1201_1" instance when stimulus adaptation was used. No significant differences were found in the other two instances regarding the use of adaptation.

5. Related Work

Several approaches based on Swarm Intelligence (Ant Colony Optimization - ACO) have been proposed in the literature to solve centralized scheduling problems, which can be modeled as a RCPSP. In these approaches, many generations of artificial ants search for a good schedule for tasks. Each ant of a certain generation builds a solution by making probabilistic decisions. In general, ants that find a good solution drop pheromones on

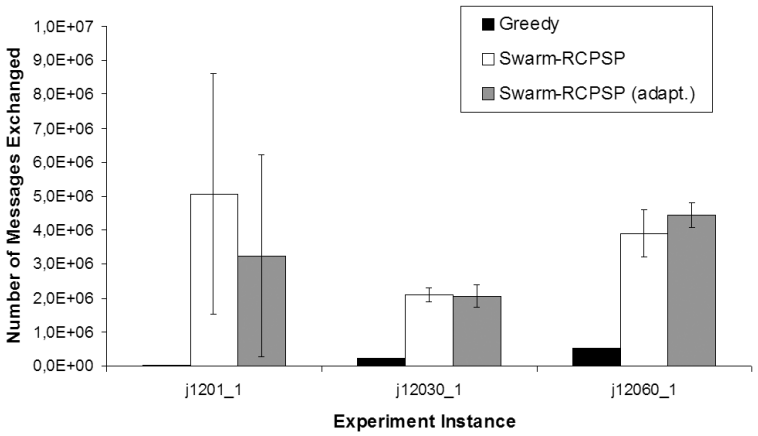

Figure 3. Comparing the number of messages exchanged by the agents.

it. Next generation ants are attracted by these pheromones and tend to search for a solution closer to the one found by the previous generation. Despite the pheromones, it is usual to add specific heuristics associated with this problem to the decision making process. Some ACO-based approaches achieve better schedules than previous approaches in several cases [Merkle et al. 2002]. However they work in a centralized fashion.

A problem related to the RCPSP, called Distributed Meeting Schedule problem (DMS), was mapped to a Distributed Constraint Optimization Problem (DCOP) and solved using DCOP algorithms [Maheswaran et al. 2004]. Agent goals in this scenario is to generate a coordinated schedule for the performance of joint activities or resource use in a multiple-events scenario. The authors tested the efficiency of one prominent DCOP algorithm (Adopt) applied to a complex domain. This so-called complex domain is much simpler than the problem considered here. An approximated algorithm that is also inspired in social insect colonies is proposed in [Ferreira et al. 2008] but deals with pure assignment of task, hence not considering the makespan.

A large number of algorithms with different approaches were recently proposed to solve DCOPs. Studies about the performance of these algorithms for large scale problems show that a search for optimal solutions is expensive in terms of communication and computational time [Davin and Modi 2005].

6. Conclusion

Agents running the Swarm-RCPSP adapt task stimulus according to tasks' start time achieved in the best scheduling. This adaptation process ensures that shorter makespans are obtained and drives the algorithm to achieve complete solutions. The Swarm-RCPSP satisfies all RCPSP constraints using simple message exchanging.

We compared the Swarm-RCPSP with a greedy algorithm and benchmark solutions. Our algorithm performed well considering that it has the advantage of being distributed. The greedy algorithm achieved results much worse than the Swarm-RCPSP. It should be pointed out that the main difference between the two processes is the swarm-like decision making. Our approach is simple and deals with the distributed RCPSP effectively. Our empirical results show that the theoretical models of division of labor in

social insect colonies may be successfully applied to distributed task scheduling.

In the future we intend to extend the experiments applying our algorithm into other instances of the PSPLIB. We are also searching for distributed algorithms in order to obtain more fairly performance comparisons.

Acknowledgments

This research is partially supported by the Air Force Office of Scientific Research (AFOSR), under grant number FA9550-06-1-0517.

References

Bonabeau, E., Theraulaz, G., and Dorigo, M. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, USA.

Brucker, P. and Knust, S. (2000). A linear programming and constraint propagation-based lower bound for the rcpsp. *European Journal of Operational Research*, 127(2):355–362.

Davin, J. and Modi, P. J. (2005). Impact of problem centralization in distributed constraint optimization algorithms. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 1057–1066, New York, NY, USA. ACM Press.

Ferreira, Jr., P. R., Boffo, F., and Bazzan, A. L. C. (2008). Using Swarm-GAP for distributed task allocation in complex scenarios. In Jamali, N., Scerri, P., and Sugawara, T., editors, *Massively Multiagent Systems*, number 5043 in Lecture Notes in Artificial Intelligence, pages 107–121. Springer, Berlin.

Kolisch, R. and Sprecher, A. (1997). Psplib – a project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216.

Kube, R. and Bonabeau, E. (2000). Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1/2):85–101.

Luo, X., Wang, D., Tang, J., and Tu, Y. (2006). An improved pso algorithm for resource-constrained project scheduling problem. *Intelligent Control and Automation*, 1(1):3514–3518.

Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., and Varakantham, P. (2004). Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, volume 1, pages 310–317, New York. Washington, DC: IEEE Computer Society.

Merkle, D., Middendorf, M., and Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333–146.

Robinson, G. E. (1992). Regulation of division of labor in insect societies. *Annual Review of Entomology*, 37:637–665.