# A Distributed Approach To Solving Constrained Multiagent Task Scheduling Problems

**Kevin Coúsin and Gilbert L. Peterson**
Department of Electrical and Computer Engineering
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45431
e-mail: {kevin.cousin, gilbert.peterson}@afit.edu

**Abstract**

The constrained multiagent task scheduling problem is a problem class that describes complex scheduling problems that integrate stochastic task allocation with constraint satisfaction for agents that concurrently execute multiple tasks. The combination of elements from constraint satisfaction, task allocation and resource scheduling produces a challenging class of problems. Unfortunately, existing work in multiagent scheduling limits the solution space and therefore task allocations do not completely encompass the complexity and interaction of the full multiagent task scheduling problem.

We propose a problem description and distributed approach designed to directly solve such problems allowing optimal and approximated solution comparison. Experimentation using a simulated alarm handling domain provides empirical evidence of the successful adaptation of our approach to solving constrained multiagent task scheduling problems with scalability towards large problem sizes.

## Introduction

Creating an optimal schedule for a realistic problem involving multiple agents or robots is challenging due to the complexities and dependencies on how a problem can be solved. For highly complex task assignment problems, computer software cannot quickly generate optimal solutions. Instead, tradeoffs are made between optimality and time that approximate the best possible solution or break the problem down into simpler structures.

Today, current task assignment algorithms can effectively solve interesting non-trivial problems generating optimal solutions (Liu & Layland 1973; Shehory & Kraus 1998; Li & Sycara 2001; Lau & Zhang 2003; LaValle 2006) and through approximation (Rybski *et al.* 2002; Torbjørn S. Dahl & Sukhatme 2004; Nair *et al.* 2005; Mailler & Lesser 2006). However, the algorithms are often designed to a specific problem domain and fail to handle alterations in the problem domain without significant modification. For instance, many algorithms work under the assumption that there is a one-to-one relationship between the individual and the task, i.e., an individual is scheduled one task at a time and a task only requires one individual. But these assumptions often do not hold in realistic environments where tasks may require several workers, and workers are able to multi-task differently.

To meet the demands of such domain problems, problem definitions must support notions of multi-capable agents concurrently performing groups of tasks, each task possibly requiring multiple agents. The definitions must also account for constraints on how or when agents can perform a task and uncertainty in what the actual schedule may be. Algorithms need to be adjusted to handle more than one specific variation of a problem—the number of agents needed to perform a task or the number of task requirements should not be a limitation of the algorithm on solving a problem.

Likewise, algorithms must support non-trivially sized task allocation problems. Several ideas in the current multiagent research introduce novel techniques that are applicable to large groups, but are often only demonstrated on very small agent or robot populations. There have also been successful implementations of distributed approaches to solving task assignment problems (Rybski *et al.* 2002; Torbjørn S. Dahl & Sukhatme 2004; Nair *et al.* 2005; Mailler & Lesser 2006) which do scale to large problems, but these also generally solve less complex problems or do not address multiagent-specific constraints.

Other related studies of this problem class introduced methods for estimating error bounds (Sandholm & Lesser 1997; Lau & Zhang 2003; Dang & Jennings 2004; Galstyan, Hogg, & Lerman 2005), incorporating constraints (Lawrence 1984; Jäger & Nebel 2001; Kraus, Shehory, & Taase 2003; Modi *et al.* 2003), measuring group dynamics (Li & Sycara 2004; Lerman *et al.* 2006) and introducing uncertainty (Miyata *et al.* 2002; Vidal *et al.* 2002; Chalkiadakis & Boutilier 2004; Nair *et al.* 2005; Verma & Rao 2005). Robotics reseach commonly uses behavior-based derivation of solutions for real time multi-robot systems (Parker 1998; Botelho & Alami 1999; Werger & Matarić 2000; Goldberg & Matarić 2001; Gage *et al.* 2004). However, a vast majority of the experiments conducted using these algorithms also used relatively simple problems, and often very few agents. In most cases, this valuable research was applied with specialized algorithms that are not suited to solve the entire space of problems without some modification.

Thus, we formulate an encompassing problem defini-

tion to represent instances of constrained multiagent task scheduling (CMTS) problems along with a distributed approach to solving them combining ideas found across a number of related disciplines. Results show that the distributed approach presented here produces reasonable solutions to an alarm handling domain while searching significantly less of the solution space using standard search algorithms.

## Constrained Multiagent Task Scheduling Problems

The constrained, multiagent task scheduling (CMTS) problem definition borrows ideas from multiagent task allocation models, resource scheduling and constraint satisfaction. It covers agents that can concurrently perform one or more tasks, and tasks that may require more than one agent doing different things for successful completion. Constraints are imposed on assignments, a pairing of an agent to a task within a complete allocation schedule. Additional constraints may exist on the ordering of tasks, implying that certain tasks need to be performed before others. The allocation process measures the utility of assigning an agent, or group of agents, to a task based on statistics gathered from the agent, task or environment which they operate in. Then, the performance of a task is scheduled when the utility of an allocation is optimal in accordance with any constraints levied upon the problem.

### CMTS Problem Definition

The CMTS problem is expressed as a set of multi-capable agents ($M$) and a set of tasks ($Y$), each mapped to a set of common domain operations ($D$). Every agent performs some subset of the domain operations and each task requires some subset of domain operations for completion. The solution to the problem is an acyclic directed graph holding sequences of assignments for every task in the problem. An assignment, represented as a node in the solution graph, matches an appropriate group of agents to each problem task. The group of agents assigned to a task in an assignment represents a coalition of agent abilities ($C_y$), or independent single-capable sub-agents representing one of the abilities of a multi-capable agent. The coalition forms by selecting the agent abilities that perform the domain operation(s) required by a task. For any one task requirement, one or more agent abilities may be assigned allowing for combined effort. Figure 1(a) shows an example of a simple CMTS problem.

The problem also expresses two types of constraints. First, every coalition must conform to inhibition constraints, i.e., no member blocks another agent's ability from performing the task. Second, tasks may have prerequisites, which are tasks that must be performed prior to a given task.

An assignment is quantified by a utility value that is derived from ability and task value functions. The value functions, $v_x$ and $v_y$ respectively, contribute domain-specific quantified information given the assignment pairing and an associated domain state $s$ which encapsulates information useful for deriving meaningful values, such as a time. For example, an agent ability that moves objects could supply its inverted distance to an object as its contributed value to the assignment. The assignment's utility value is then measured as a function, $f^A$, of the combined value of the abilities in the assigned coalition, $f^C$, and the task's contributed value for each ability in the coalition, shown in Equation 1. Finally, the utility of the solution is quantified by a function $F$ which combines the utility values of each assignment contained in the solution, Equation 2.

$$u((C_y, y), s) = f^A(f^C_{x \in C_y}(v_x(y, s)), v_y(C_y, s)) \quad (1)$$

$$U(S) \equiv U(S, s) = F_{(C_y, y) \in S} u((C_y, y), s) \quad (2)$$

Algorithms seeking to find the best solution for a problem attempt to optimize the utility value $U(S)$. Furthermore, algorithms that solve by approximating solutions can use an expected utility value $E[U(S)]$, shown in Equation 3. The expected utility of a partial solution measures the utility of the assignments already made plus the expected value of the value function for each remaining task, $E[v_y(X, s)]$ given a full coalition effort of all agent abilities.

$$E[U(S)] = U(S) + \sum_{y \notin S} E[v_y(X, s)] \quad (3)$$

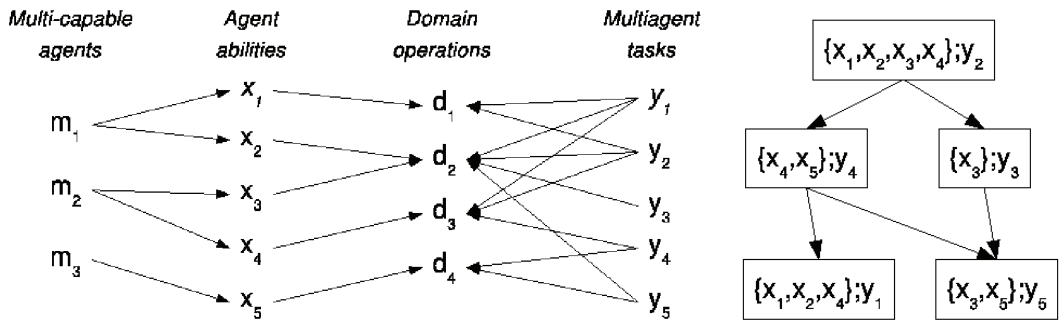
## Distributed Solving

Like other multiagent task allocation approaches, the CMTS problem and its solution lend theirselves to a graph representation. The schedule solutions for CMTS problems are based on an acyclic directed graph allowing multiple parents per node. Nodes in the graph represent a single assignment of a coalition of agent capabilities to a task. The edges in the graph represent a transition of agents moving from one task to the next. The edge itself does not require any value, although there is certainly future application for it, e.g. uncertain or probabilistic transitions. Some edges can cause distinct paths between two nodes, such as a node's parent (or any of its ancestors) referencing its children.

Figure 1(b) illustrates a graph solution showing the pairing of a coalition of abilities to a task for the example problem appearing in 1(a). Each box contains an assignment of a coalition of agent abilities (in brackets) matched with a task. If task $y_1$ were to be a prerequisite of any other task, or if ability $x_2$ interferes with $x_4$, then this solution would not be valid. Note that the graph node containing an assignment for task $y_5$ has two parents.

### Solution Construction

Serial algorithms construct solutions by one of two means: iterative construction or edge selection. In iterative construction, solutions are formed by iteratively assigning a legal, non-inhibiting coalition to a task, then inserting the assignment into the solution. The insertion of a new assignment is restricted by the prerequisites of a task. Clearly, any task that is a prerequisite of another must appear "earlier" in the solution, implying that assignments containing prerequisite tasks are in the ancestry of the constraining task. This insertion process continues until all tasks are assigned to a coalition. This is best suited, for example, for algorithms based on a breadth- or depth-first search.




(a) Example CMTS Problem (b) Valid Solution Graph

Figure 1: Example CMTS Problem with Solution Representation

Alternatively, solutions can be generated by selecting edges of a fully connected graph that form a proper solution. The graph essentially has a node for every possible assignment combination. The subgraph formed from selected edges in this graph must contain an assignment for every task and that appropriate constraints are met. This approach works best for algorithms that solve problems primarily using the solution space, such as a random selection, simulated annealing or evolutionary algorithms.

## Distributing the Solution Process

The size of the search space for a CMTS problem is particularly challenging for increasingly larger problems. Given a set of abilities, $X$, and a set of tasks, $Y$, for a problem, the search space is approximately (disregarding problem-specific constraints) $\left[2^{|X|}-1\right]^{|Y|}$. $\left[2^{|Y|-1}\right]^{|Y|} \sim O\left(2^{|Y|(|X|+|Y|)}\right)$—$\left[2^{|X|}-1\right]^{|Y|}$ counts the maximum number of possible coalitions for each task, and $\left[2^{|Y|-1}\right]^{|Y|}$ for the number of possible task sequences. Unfortunately, computing resources often limit problem sizes for multiagent scheduling algorithms. In order to solve relatively large CMTS problems, a distributed approach becomes necessary. Often, this approach reduces the effects of this limitation by solving smaller, decomposed pieces of a problem and combining the sub-results into a final solution.

The distributed CMTS algorithm approach builds on traditional problem decomposition, especially since the problem description provides several opportunities for decomposition. The most opportunistic decompositions are to solve the problem by assigning subsets of agents to the total task set or by assigning the agent group to subsets of tasks. The latter is somewhat less desirable since the decreasing number of tasks does not greatly reduce the size of a problem search graph, and there is a much higher chance of agents getting assigned to tasks that would execute concurrently when subresults are combined. A very robust error checking and correcting mechanism is then required to properly recombine the results. Decomposing by agents, on the other hand, produces solutions that are easily recombined because any combination of capabilities are allowed on a task, provided each capability does not inhibit another. Yet, a third, untested, decomposition creates subproblems based on which operations are involved. This seems to involve a higher degree of error checking and correction than decomposing by tasks.

The basic approach chosen for decomposing a CMTS problem for distributed processing involves decomposing both agents and tasks. Essentially, to create $k$ subproblems (one for each of $k$ processing nodes) from a problem with agent set $M$ and task set $Y$, $|M|/k + \delta_M$ agents and $|Y|/k + \delta_Y$ tasks, $\delta_M, \delta_Y \geq 0$, are sequentially selected for each subproblem. The $\delta$ bias controls how much overlap there is between problems. A bias greater than 1 ensures that there is some overlap between subproblems which forces some task assignment decisions to be examined by more than one algorithm. This is useful for getting second or more "opinions" about an assignment.

Then, before a subproblem is released to a processing algorithm, all prerequisites of tasks in the subproblem are added if not present. Likewise, if any abilities are missing, an agent that can perform the missing ability is chosen at random to cover the assignment. Clearly, after the additions, the resulting subproblem sizes may not be the same. It is possible to grow a subproblem back to original size, but this is the exception, not the norm. Such a situation primarily occurs when the tasks are completely ordered by prerequisites.

Once the subproblems are solved, the individual results are collected for recombination. The first step is to build a combined solution using assignments that are common in each subsolution. Afterwards, each subsolution is ranked according to the size of the subproblem solved. Then for every task not in the combined solution, the first occurrence of an assignment for the task among the ranked subsolutions is used. The ranking ensures that subproblems with exposure to more information carry a more significant vote in the complete solution. The combined solution preserves the parent/child relationships of an assignment from the subsolutions, but in the event of a conflict, the relationships of the better ranked subsolution are kept.

### Expected Performance

The CMTS problem is solvable using optimal solution generating algorithms, such as depth-first search or A*, or through approximating algorithms, such as beam search or greedy methods. Algorithms generating optimal solutions generally examine a significant portion of the search space, requiring similar amounts of memory resources to keep track of discovered solutions. But, the search space size of relatively small problems is a barrier to these algorithms since the memory requirements scale with an exponential search size. For example, a problem with three agents having two abilities each and five tasks approaches a search space size of $2^{50} \sim 10^{15}$. It is unlikely that optimal solutions will be efficiently found without significant decomposition of the problem space for large problems using these algorithms.

However, using a distributed method designed around decomposing the problem likely introduces errors. The error comes from solving subproblems that may not have the information necessary to generate optimal solutions. For instance, if an optimal solution requires the combined abilities of two specific agents on a certain task, but those agents are not present together in any subproblem, the recombination method would not introduce a solution with the two agents assigned to perform the task. The decomposition method described in this section is subject to introducing this type of error.

Approximation algorithms mitigate the memory issue in various ways, but must still carefully consider how to construct or search for solutions. The algorithms must be able to effectively search the large spaces involved with a CMTS problem. This generally involves tuning parameters to allow for very wide searches around localized solutions, such as a wide beam for a beam search, or moving search points relatively far away from local optima. Unfortunately, searching too broadly may lead to searching suboptimal regions or slow convergence. Since search space is reduced when using this distributed method, the approximation algorithms may use more suitable search parameters. However, the algorithms are still subject to errors introduced by information loss.

Thus, the optimal solution generating algorithm should provide the best possible solution for feasible-sized problems when directly applied, but using the distributed method with such algorithms is not guaranteed to produce an optimal solution. Approximation algorithms should approach the optimal solution where directly used, but perform comparable to any type of algorithm when used in the distributed system. In any case, the number of solutions examined by either type of algorithm should be significantly reduced when using distribution.

## Experimentation

The multiagent problem domain used for experimentation is a multi-capable agent, multi-agent task alarm response where alarms require between one and three distinct domain operations to be deactivated. A problem instance involves a group of agents scattered across a two-dimensional area containing some number of activated alarms, illustrated in Figure 2. Each alarm identifies a task requiring one or more agents to collectively perform. The goal is for the agents to efficiently service all of the alarms.

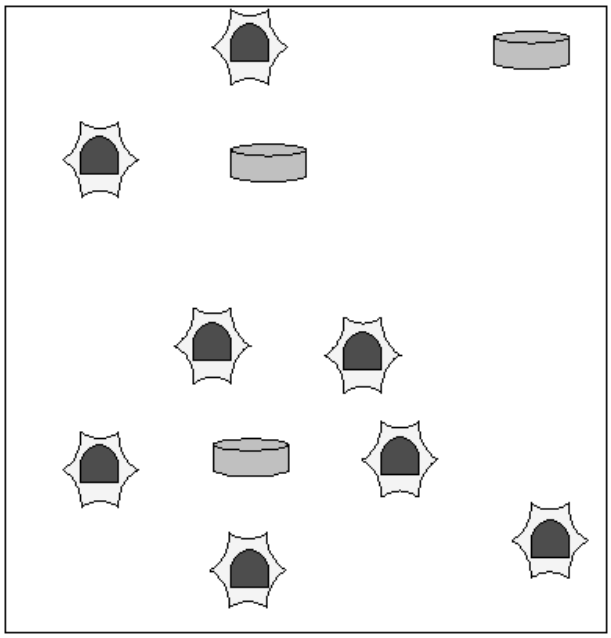


Figure 2: Alarm Domain Problem w/ Three Agents Handling Eight Alarms

Solutions to this problem must optimize two objectives: minimize the total distance traveled to deactivate the alarms and minimize the number of tasks any agent performs. The first objective implies that agents are to complete the problem as quickly as possible while the second encourages coalitions that balance the workload—although having one agent do it all is a valid solution, having several others sit idle is generally not very efficient.

Using the CMTS problem descriptor, the agent ability value function measures the distance of an agent to an alarm task from its last working position (or an initial position if it has not yet performed a task) for objective one and provides its portion of the total contribution towards a task,yielding the function $v_x(y, s) = (dist(x, y), \frac{1}{|C_y|})$, where $C_y$ is the coalition performing task $y$. The task value function quantifies the objectives by denoting a distance value of zero and a step value equal to the depth of the assignment in the solution (one plus the number of tasks that preceed it in the graph) yielding $v_y(C_y, s) = (0, depth(y))$. The state, $s$, used for these functions is the currently built solution.

The assignment utility value $u((C_y, y), s)$ combines the ability and task value functions as a sum over the agent distances to an alarm task and a product of task steps for each ability. The coalition combination function $f^C$ is summation over the agent distances, and the assignment combination function sums the distances of the agents to all tasks in a partial solution with the depth of the task in the solution, represented in Equation 4. The overall utility value

$U(S)$ combines the assignment utility values by producing a single two-dimensional value by summing over each of the distances and taking the maximum of the task depths, simplified in Equation 5.

$$u((C_y, y), s) = (\sum_{x \in C_y} dist(x, y), depth(y)) \quad (4)$$

$$U(S) = (\sum_{x \in C_y \forall (C_y, y) \in S} dist(x, y), depth(S)) \quad (5)$$

The heuristic chosen for the alarm problem domain is to choose the smallest distance to all remaining tasks and assume that all remaining tasks are completed within $\lceil |Y'|/|X| \rceil$ steps of a partial solution, where $Y'$ is the set of unassigned tasks. Using these values to estimate the utility of an assignment consistently underestimates (at most equals) the true assignment utility value. Likewise, the heuristic is consistent since the non-negative distances are summed and the estimated number of remaining steps is monotonically decreasing. Thus the heuristic component to the problem yields $E[v_y(X, s)]$ as shown in Equation 6, per the estimated utility value description defined by Equation 3.

$$E[v_y(X, s)] = (argmin(dist(x, y), \lceil \frac{|\{y \notin A\}|}{|X|} \rceil) \quad (6)$$

## Configuration

Three serial algorithms provide the base algorithms for the distributed tests: A*, beam search, and random selection. The A* algorithm is chosen because, for the given heuristic, it generates optimal solutions. Beam search performs similarly to A* (heuristically driven), but can only approximate solutions since it limits the number of potential solution candidates it evaluates. The experiments test beam search using pool sizes of 50, 100, 150, 200, 250 and 500 candidate solutions. The A* and beam search algorithms use the same heuristic to determine which partial solutions are the most competitive for further expansion. Finally, the random selection mainly provides a way to sample the quality of the solutions generated by the distributed approach.

The distributed tests involve three mixtures of the serial algorithms over three processors. The first configuration has all three processors using the A* algorithm. Although the A* algorithm is generating optimal solutions, the distributed algorithm using all A* is not guaranteed to produce an optimal solution due to the decomposition method described in the Distributed Solving section. The second configuration generates solutions with all processors using the beam search algorithm set to a pool size of 200. The final distributed test mixes the algorithms with one processor running A* and the other two running beam search with 200 candidates.

Each of the CMTS algorithms and distributed approaches are tested on alarm domain problem instances having 2 to 5 agents, with each agent having 2 or more abilities, collectively perform a problem consisting of 2, 4, 6 or 8 alarm tasks, many requiring more than one ability. The A* algorithm serves as the baseline for optimal solutions, and the random selection algorithm provides comparative sample quality.

Table 1: Average Difference From Best Solution.

| Algorithm | 2-Tasks | 4-Tasks | 6-Tasks | 8-Tasks |
|---|---|---|---|---|
| A* | 0.00 | 0.00 | 0.00 | — |
| Beam (50) | 0.00 | 1.97 | 2.00 | 2.21 |
| Beam (100) | 0.00 | 1.97 | 2.00 | 4.85 |
| Beam (150) | 0.00 | 0.00 | 2.00 | 0.00 |
| Beam (200) | 0.00 | 0.00 | 0.00 | 0.00 |
| Beam (250) | 0.00 | 0.00 | 0.00 | 0.00 |
| Beam (500) | 0.00 | 0.00 | 0.01 | 5.00 |
| Dist. A* | 0.00 | 15.58 | 29.61 | 24.87 |
| Dist. Beam | 0.84 | 10.55 | 23.58 | 24.22 |
| Dist. Mix | 0.00 | 9.47 | 27.83 | 29.91 |
| Random | 25.32 | 59.42 | 82.71 | 100.47 |

## Results

Table 1 and Figure 3 chart comparisons of the algorithms to gauge their effectiveness and efficiency in serial and distributed processing. The serial A* algorithm provided baseline optimal results for each of the problems except the eight-task problems which it was unable to complete due to resource limitations. The results in Table 1 show how the quality of solutions deviated in unit steps (metric distance of the distance and number of steps of agents in the problem domain) from the optimal results found by A* for 2, 4 and 6-task problems and the best solution found for the eight-task problems. As expected, the beam searches generally improve as the beam width grows, but the effects of approximating the solution by limiting candidate pool sizes is apparent for a pool size of 500, where the search result is of higher deviation than smaller beams. The distributed algorithms, however, have much more room for improvement, but offer an average 51% improvement over randomly generated solutions which brings some merit to the distributed approach in light of its straight-forward decomposition approach.

The results in Figure 3 shows the efficiency of the algorithms when solving the same problem set. The beam search algorithms generate a notably lower number of solutions for the same set of problems as A*, but each of these algorithms generate significantly more solutions than the distributed approach. The solution generation rates show that the distributed approach requires nearly two orders of magnitudes less generation than stand-alone algorithms. Specifically, the distributed beam search required less than 0.4% of the number of generations to find it solution; the distributed A*, at worst, needed less than 0.7%. This is key to efficiently solving much larger problems.

# Future Work

This research is continuing to be develped in two complementary ways. Additional solution techniques are being developed to solve the CMTS problem using biologically-inspired models such as genetic algorithms and ant colony optimization. Additional algorithms such as reinforcement

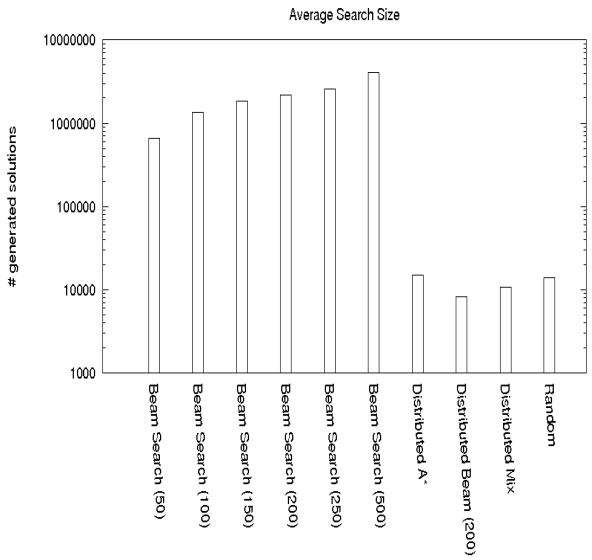



Figure 3: Comparison of Algorithm Performance

learning, particle swarm optimization, auction/agent negotiation and integer linear programming are also being examined. Along with these methods, new problem domains are being developed to fully test different aspects of the CMTS definition, such as the impact of certain types of constraints of the solution search space.

Likewise, different approaches to decomposing the problem and solution space are being examined. This work demonstrates problem decomposition based on generating a certain number of subproblems in this effort. However, there are other viable alternatives such as decomposition by tasks, where problems are decomposed as one per task and only associated agents are included; and similarly for decomposition by agent. There are also methods that decompose the solution space rather than the problem that are more traditional distributed system exploit. This approach would allow for generation of optimal solutions by a distributed system if the entire search space is examined by a collective of thorough search algorithms, but the efficiency tradeoff requires modeling.

## Conclusion

The techniques illustrated in this report demonstrate the basic approach to forming and using scalable, distributed approaches to constrained assignment problems incorporating basic agent behaviors. We introduce a distributed approach to solve constrained multiagent task scheduling problems, a robust description for complex task scheduling problems involving multi-capable agents concurrently performing tasks that can require multiple agents providing different abilities. A small set of algorithms are applied to a typical CMTS problem to serially generate optimal and approximated solutions.

Experimental results show that the distributed process using the serial algorithms provides solutions slightly above average quality, as expected, but requiring significantly less search space exploration to generate those solutions. This demonstrates that this distributed process is a viable algorithmic approach to approximating solutions to large CMTS problems using mixtures of algorithms that serially generate optimal and approximated solutions.

This research opens the door for hybridized algorithm development, distributed solution development with high scalability and real-time scheduling in a dynamic environment using techniques coming from several disciplines such as distributed constraint satisfaction, behavior-based robotics, task allocation and uncertainty modeling.

## Acknowledgments

The authors would like to acknowledge funding through AFRL Lab Task 06SN02COR from the Air Force Office of Scientific Research, Lt Col Scott Wells, program manager, and the Dayton Area Graduate Studies Institute, Dr. Elizabeth Downie, director.

The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense or the U.S. Government.

## References

Botelho, S., and Alami, R. 1999. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. *IEEE International Conference on Robotics and Automation* 2:1234–1239.

Chalkiadakis, G., and Boutilier, C. 2004. Bayesian reinforcement learning for coalition formation under uncertainty. *Third International Joint Conference on Autonomous Agents and Multiagent Systems* 03:1090–1097.

Dang, V. D., and Jennings, N. R. 2004. Generating coalition structures with finite bound from the optimal guarantees. *Proceedings of the Third International Conference on Autonomous Agents and Multi-Agent Systems* 564–571.

Gage, A.; Murphy, R.; Valavanis, K.; and Long, M. 2004. Affective task allocation for distributed multi-robot teams.

Galstyan, A.; Hogg, T.; and Lerman, K. 2005. Modeling and mathematical analysis of swarms of microscopic robots. *IEEE Swarm Intelligence Symposium*.

Goldberg, D., and Matarić, M. J. 2001. *Design and Evaluation of Robust Behavior-Based Controllers for Distributed Multi-Robot Collection Tasks*. AK Peters, Ltd. 315–244.

Jäger, M., and Nebel, B. 2001. Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots. In *Int. Conf. on Intelligent Robots and Systems (IROS)*.

Kraus, S.; Shehory, O.; and Taase, G. 2003. Coalition formation with uncertain heterogeneous information. In *Proceedings of AAMAS*, 1–8.

Lau, H. C., and Zhang, L. 2003. Task allocation via multi-agent coalition formation: Taxonomy, algorithms and complexity. *15th IEEE International Conference on Tools with Artificial Intelligence* 346.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

Lawrence, S. 1984. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement).

Lerman, K.; Jones, C.; Galstyan, A.; and Matarić, M. J. 2006. Analysis of dynamic task allocation in multi-robot systems. *International Journal of Robotics Research* 25(3):225–241.

Li, C., and Sycara, K. 2001. Algorithms for combinatorial coalition formation and payoff division in an electronic marketplace. In *First International Joint Conference on Autonomous Agents and Multiagent Systems(AAMAS)*, 120–127.

Li, C., and Sycara, K. 2004. *A stable and efficient scheme for task allocation via agent coalition formation*. World Scientific. 193–211.

Liu, C. L., and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM* 20(1):46–61.

Mailler, R., and Lesser, V. 2006. Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *Journal of Artificial Intelligence Research* 25:529–576.

Miyata, N.; Ota, J.; Arai, T.; and Asama, H. 2002. Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment. *IEEE Trans. on Robotics and Automation* 18(5):769–780.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2003. Solving distributed constraint optimization problems optimally, efficiently and asynchronously. In *Proceedings of AAMAS*.

Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2005. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *Proceedings of AAAI*.

Parker, L. E. 1998. Alliance: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation* 14(2):220–240.

Rybski, P. E.; Stoeter, S. A.; Gini, M.; Hougen, D. F.; and Papanikolopoulos, N. 2002. Performance of a distributed robotic system using shared communications channels. *IEEE Trans. on Robotics and Automation* 22(5):713–727.

Sandholm, T. W., and Lesser, V. R. 1997. Coalitions among computationally bounded agents. *Artificial Intelligence* 94(1-2):99–137.

Shehory, O., and Kraus, S. 1998. Methods for task allocation via agent coalition formation. *Artificial Intelligence* 101(1–2):165–200.

Torbjørn S. Dahl, M. J. M., and Sukhatme, G. S. 2004. *Complex Engineering Systems*. Perseus Books. chapter A machine learning method for improving task allocation in distributed multi-robot transportation.

Verma, D., and Rao, R. 2005. Graphical models for planning and imitation in uncertain environments. Technical Report 2005-02-01, Department of CSE, University of Washington.

Vidal, R.; Shakernia, O.; Kim, H. J.; Shim, H.; and Sastry, S. 2002. Multi-agent probabilistic pursuit-evasion games with unmanned ground and aerial vehicles. *IEEE Trans. on Robotics and Automation* 18(5):662–669.

Werger, B. B., and Matarić, M. J. 2000. *Broadcast of local eligibility for multitarget observation*, volume 4. New York: Springer-Verlag. 347–356.