

# Coordinated aggressive bidding in distributed combinatorial resource allocation

Jinbo Chen, Alejandro Bugacov, Pedro Szekely,  
Martin Frank, Min Cai, Donghan Kim, Robert Neches  
USC/Information Sciences Institute  
4676 Admiralty Way  
Marina del Rey, CA 90292

{jinbo,bugacov,szekely,frank,mcai,donghank,rneches}@isi.edu

## ABSTRACT

Large NP-hard combinatorial resource allocation problems are best solved via approximation techniques which produce acceptable solutions in the time available. The best of such “good-enough/soon-enough” techniques handle large problems, run in distributed environments, adapt rapidly to changes to the problem while solving, and exhibit good anytime performance.

The Dynamic Marble Size (DMS) algorithm appears promising as an approach offering all those traits. It is a market-inspired distributed multi-agent scheme, in which task agents aggressively bid for their preferred resource bundles through single-resource auctions, coordinating their interdependent bids by bid adjustment.

In the DMS algorithm, tasks maximally bid their value. This aggressive bidding strategy maximizes other bidders’ information about their prospects of succeeding, and thereby also help them focus on resources they can win. An oscillation-avoiding bid adjustment algorithm utilizes a binary search technique to prevent those adjustments from introducing cycles or deadlocks. A “stubbornness-detection monitor” (a re-start limit) limits how many sets of resources a task will pursue. Aggressive bidding, oscillation-avoidance and stubbornness-detection promote rapid convergence on good solutions – bidders avoid highly-contended resources and focus upon more promising alternatives.

To evaluate the DMS algorithm, we analyzed characteristics of randomly-generated problems, using results obtained with a Pseudo-Boolean encoding of the problem as the gold standard for quality obtainable from centralized solutions. For the types of problems that our problem generator can produce, DMS produces comparable solution quality, using significantly less time, as well as exhibiting a good anytime performance. Furthermore, a variant of DMS scales linearly in the problem size.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

## Categories and Subject Descriptors

C.2.4 [Computer Systems Organization]: Distributed Systems—*distributed applications*; D.2.8 [Software Engineering]: Metrics—*complexity measures, performance measures*; I.2.11 [Computing Methodologies]: Distributed Artificial Intelligence—*coherence and coordination, multiagent systems*

## General Terms

Cooperative negotiation, distributed resource allocation, market mechanisms for multi-agent systems, combinatorial auctions

## Keywords

decentralized resource allocation, auction protocols, bidding strategies, distributed constraint optimization

## 1. INTRODUCTION

A resource allocation problem consists of a set of tasks to be performed and a set of resources that can be used to perform the tasks. The goal is to determine an assignment of resources to tasks so that the utility of the tasks is maximized. In many real-world domains (e.g. flight crew training scheduling [2], electricity markets [4], transportation exchanges [8]), resource utilization efficiency could be enhanced if tasks are allowed to express preferences over bundles of resource items (i.e. some resources can be complementary or substitutes).

Because of resource complementarities, bids from a task are not independent. For example, a task has a bundle resource preference: A, B and C. So bidding on a resource (B) could depend on the following:

1. whether the task is winning on resource A;
2. how much the task has bid on resource A;
3. what is the task’s speculation or counter-speculation on other tasks’ bidding on A and C.

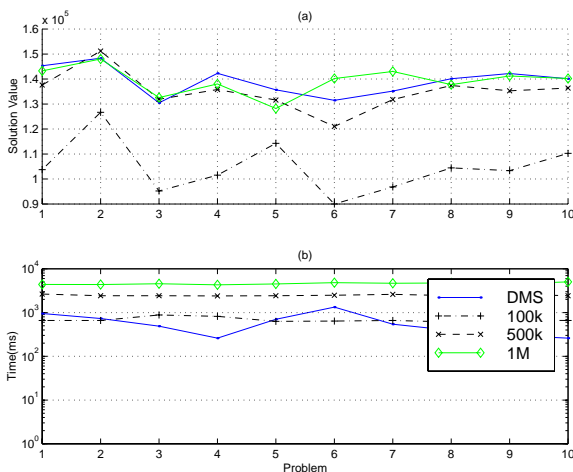
These complications cause the traditional single-resource auction to be inefficient for the resource allocation problems with bundled resource preferences.

To address inefficiencies of the traditional single-resource auction, combinatorial auctions were proposed, in which

multiple resources could be allocated in a single auction [6]. Winner determination in combinatorial auctions is NP-Complete [9]. There has been a recent surge of interest on combinatorial auctions (Single-unit and multi-unit) [6, 1, 7, 3, 9] But these proposed combinatorial auctions could be potentially computationally complex for large real-world problems (e.g. long term resource allocation and scheduling for crew flight training).

For many real-world applications, NP-Hard combinatorial resource allocation problems are best solved via approximation techniques which produce acceptable solutions in the time available. The best of such “good-enough/soon-enough” techniques handle large problems, run in distributed environments, adapt rapidly to changes to the problem while solving, and exhibit good anytime performance.

As a step toward a technique that approximates the combinatorial resource allocation problems and satisfies the desiderata of the preceding paragraph, we have developed a market-inspired approximation technique called the Dynamic Marble Size (DMS) algorithm. The DMS algorithm extends the traditional single-resource auction by introducing coordinated aggressive bidding in tasks. From the evaluations running on randomly-generated problems, the DMS algorithm appears promising as an approach offering all those desired traits from an approximation technique.



**Figure 1: Comparison between DMS and a Pseudo-Boolean solver: comparable solution quality achieved approximately an order of magnitude faster**

To quantitatively measure “good enough/soon enough”, the DMS algorithm was compared on solution quality and speed against a centralized algorithm based on a Pseudo-Boolean encoding [11].

For 10 randomly-generated problems with 30 resources and 30 tasks, the DMS algorithm produces comparable solution quality against the pseudo-boolean encoding algorithm using 500K and 1M “flips” [11] (Figure 1 (a)), and the DMS algorithm is about one order of magnitude faster (Figure 1 (b)).

Comparisons on different sized problems also showed that the DMS algorithm compares favorably to the centralized solver.

## 2. PROBLEM

One way to formulate combinatorial resource allocation problems is as a table where rows represent tasks and columns represent resources. Table 1 shows a problem instance with five tasks (Q, R, S, T and U), and 10 resources labelled  $A$  through  $J$ . Task Q is worth 200, R is worth 192, and so on.

Each task has one or more requirements, each represented as a row in the table. The cells formed by the requirements and the resources contain a star ( $\star$ ) when a resource is qualified to fulfill a requirement. For example, resource A is qualified to fulfill requirements 1 and 2 of Q, requirement 1 of R and requirement 3 of S.

A black circle ( $\bullet$ ) indicates that the corresponding assignment is part of the optimal solution while a white circle ( $\circ$ ) indicates that it is part of the solution found by DMS.

A resource allocation for a task consists of an assignment of qualified resources to *all* requirements of a task. In our problem formulation, partially filled tasks are worthless. The objective is to maximize the sum of the value of the completely filled tasks.

The solution to even small problems such as the one shown in Table 1 is hard to determine visually. The optimal solution for this problem allocates all five tasks for a solution value of 837. DMS finds a solution worth 750 by allocating the first four tasks (see Section 4.5).

In this work we study problems of different sizes. The small ones contain about 30 resources and tasks, and the large ones contain 8192 resources and tasks. Even a 30 by 30 problem is large enough that computing an optimal solution using a complete solver is not feasible using many hours of computation time in a contemporary personal computer. In fact, Zhang proved that marble problems are NP-Hard when tasks can have more than two requirements [13].

Given the complexity issues, approximation algorithms are highly desirable. Because there are many applications which are inherently distributed, and therefore not amenable to centralized solutions, the ideal approximation approach would be decentralized as well. Other market-inspired, distributed approximation techniques have been explored [2]. However, the Dynamic Marble Size (DMS) algorithm described in the next section appears to scale better to large problems.

## 3. APPROACH

The Dynamic Marble Size (DMS) algorithm is a market-inspired distributed resource allocation algorithm. It is one member of a family of resource allocation algorithms called “Marbles algorithms” [2]. Marbles algorithms represent both tasks and resources as agents: one agent for each task and for each resource. Each resource agent runs a single-unit auction to allocate its resource to the highest bidder. A task agent sends separate bids for its preferred resource bundle, but it coordinates its bids by bid adjustment.

In the DMS algorithm, concepts of “marble” and “marble size” are introduced for performing bid adjustment. A marble has a size that denotes a certain amount of money, and it could be further cut into several small sized marbles.

The basic idea of the DMS algorithm is that, a task agent always starts by choosing the cheapest resource set out of its possible combinations and bidding all its money on the selected resources. If outbid, the task agent will adjust its bids by transferring money from a winning bid to a losing

		Resources									
Task (value)		A	B	C	D	E	F	G	H	I	J
Q (200)	1	★●		★	★			★○			
	2	★○								★●	★
R (192)	1	★				★●			★		★○
	2			★			★		★●○		★
S (240)	1						★●	★		★○	
	2		★●	★○			★	★			
	3	★	★○					★●			
T (118)	1		★		★●		★○				
U (87)	1			★●			★			★	

**Table 1: An example problem. A star indicates that the resource could fill the task’s requirement. The filled circle indicates an optimal resource assignment for this problem. The empty circle indicates the resource assignment found by the DMS algorithm for this problem.**

bid until all its bids are winning. If a task can not win all its bids by bid adjustment, it re-starts by choosing the cheapest resource set based on the current resource prices, and aggressively bids again. If a task can not even afford its cheapest resource set, or re-starts too many times, it will withdraw from auctions.

The motivation for this bidding strategy is that, by maximally increasing resource prices, tasks will more quickly realize that they cannot compete on some resources, then focus on other alternatives. Some tasks will also withdraw quickly if their cheapest resource sets rapidly becomes unaffordable. This bidding strategy accelerates the closure of the negotiation compared to incremental-bidding schemes.

The flip side of bidding aggressively is that a task could select a resource to bid on from its choices arbitrarily, and because the bid is high it may cause another task to withdraw inappropriately. The reason that a task may inappropriately select a resource to bid on is that resource prices only approximately embody the demands from tasks. For example, some lower value tasks could fail to reveal their demands if their bids are already lower than the current resource prices.

The rest of this section describes the DMS algorithm in greater detail.

### 3.1 Resource Agents

In the DMS algorithm, resource agents are very simple. Their state is defined by two variables: the current price of the resource, and the name of the highest bidder.

Resource agents accept three types of messages:

1. *Bid(task, amount)*: if the amount is lower or equal than the current price, the resource agent responds with a *lose* message. If the price is higher than the current price, the resource agent sends a *lose* message to the previous winner (if different from the new winner), and sends a *win* message to the new winner.
2. *Withdraw(task)*: if the task is the same as the current winner, the current winner is erased and the price is set back to zero.
3. *Inquiry(task)*: the resource agent will report the current price to the sender.

### 3.2 Task Agents

DMS task agents perform the following steps:

1. *Inquiry*: ask all qualified resources for their current price.
2. *Select*: select a candidate resource for each requirement so that the total price of the candidates is minimal.
3. *First bid*: divide the task value equally among all requirements, thus creating one marble for each requirement, and bid that marble.
4. *Adjust bids*: If needed, cut marbles in half and move them among requirements to try to win all the selected resources.

The *adjust bids* step can fail because even after cutting the marbles in half several times, and moving them around, the task may not be able to win all the selected resources. In that case the task will give up on the selected resources (sending withdraw messages for any resources it may hold) and repeat the steps. It will inquire for prices again, likely select a different set of resources and try the bidding again. By restarting the bidding and re-sampling the prices, the algorithm can recover from previous decisions based on resource prices that didn’t accurately represent demand. Because each bidding round takes several message exchanges, by the time a task re-starts its bidding, other tasks will have also bid, thus increasing the accuracy of prices.

The algorithm has two control parameters. One is the number of times a task agent is allowed to cut marbles in half during the *adjust bids* step. The other one is the number of times a task agent is allowed to start over with the steps, which is currently utilized as a simple way to prevent tasks from repeating the steps fruitlessly (the stubbornness-detection monitor).

We present the details of the algorithm for tasks below.

Tasks receive two types of messages from resource agents:

1. *Win(resource)*: sent in response to a *bid* message to announce that the task agent’s bid on a resource was the highest and the task is now winning that resource.
2. *Lose(resource)*: sent in response to a *bid* message to announce that the bid is not high enough to win the resource. A resource may send a *lose* message after previously having sent a *win* message. This happens when a task is out-bid at a later time.

In order to simplify the description of the algorithm we will omit details having to do with lower-level message han-

dling (timeouts to deal with message losses, etc.), and describe the algorithm in terms of a high level event called *All-bid-responses*. This event will be triggered when a task receives all the *win* and *lose* responses to all outstanding *bid* messages.

The handler for *All-bid-responses* and *lose* messages is defined in line 19. Before describing the details of the handler we describe the variables used in the pseudo-code.

```

1 constant C, N
2 global n = 0
3 global a, w, m[], s

```

Line 1 defines the control parameters of the algorithm. *C* stands for the number of times that marbles can be cut in half, and *N* stands for the number of times the task agent can start new bidding rounds. Line 2 initializes the current number of restarts *n* to 0; Line 3 declares *a*, the variable that holds the current assignment of resources to requirements; *w*, the current list of winning requirements; *m*[], an array that records the number of marbles bid on each requirement; and *s*, the current marble size;

Task agents are initialized by sending *inquiry* messages to all qualified resources and calling *start-bidding* when they receive replies to all *inquiry* messages (we ignore unresponsive resources and/or message loss and delay here for the sake of brevity).

```

4 procedure start-bidding()
5   n = n + 1
6   if n > N then
7     withdraw
8   else
9     a = select-resources()
10    if cost(a) > task.value then
11      withdraw
12    else
13      w = empty
14      m[] = 1
15      s = task.value / # requirements
16      for each i in a bid m[i] * s
17    endif
18  endif

```

Procedure *start-bidding* works as follows. It first increments the number of bidding starts *n* (line 5), and then tests whether it has reached the limit of restarts allowed by the control parameter *N*. In that case the task withdraws from competition and reclaims all its marbles. In line 9 it selects the cheapest set of candidate resources for all requirements. Given the current price information, the selected allocation is the cheapest to satisfy all requirements. Of course, it may turn out that the task doesn't have enough marbles to afford it, or prices may have changed by the time the bids from this task arrive at the resource agents, or the price of resources may rise in the future. The rest of the algorithm is designed to cope with these contingencies.

If the total cost of the cheapest set is more expensive than the task value, then the task withdraws from competition (line 10). Otherwise, it proceeds to initialize the global variables (lines 13 to 15). It initializes the current set of winners to the empty set (line 13), the number of marbles spent on each requirement to 1 (line 14), and the marble size to the task value divided by the number of requirements (line 15). Then it sends a bid for each selected resource (line 16).

After sending all the bids, the task waits for the responses to come back, which will trigger the *event-handler* shown

below. The same handler is also triggered when the task receives asynchronous *lose* messages after previously having received a *win* message from the same resource.

```

19 event-handler(all-bid-responses, lose)
20   w = append winners to w
21   if length(w) < # requirements then
22     if w != empty then
23       adjust-bids()
24     else
25       start-bidding()
26     endif
27   endif

```

In line 20 the task computes the winning requirements by scanning the reply messages. It selects the winning ones and appends them to *w*. In line 21 the task determines whether it is winning on all requirements, in which case it does nothing. Otherwise, it has to try to win on the losing requirements by moving marbles from winning resources to the losing ones. In line 22 it tests whether there are winning resources from which to steal a marble, and if so calls *adjust-bids* to move marbles. If it is not possible to move the marbles, then the current allocation should be abandoned, and a new one tried by calling *start-bidding*.

Below we show the algorithm to determine whether the current allocation can be adjusted by moving marbles from winning to losing requirements.

```

28 procedure adjust-bids()
29   l = randomly select losing requirement
30   x = first of w
31   case
32     m[x] > 1:
33     m[x] = m[x] - 1
34     m[l] = m[l] + 1
35     send bids for x and l
36   m[x] = 1:
37     if c = C then
38       start-bidding()
39     else
40       c = c + 1
41       m[] = m[] / 2
42       for each i in a m[i] = m[i]*2
43     m[x] = m[x] - 1
44     m[l] = m[l] + 1
45     send bids for x and l
46   endif

```

The *adjust-bids* procedure works as follows. Line 29 sets *l* to a randomly selected losing requirement. Line 30 sets *x* to the first element in list *w*. Then two cases arise. If the selected resource has more than one marble (line 32) then the task transfers a marble from *x* to *l*, and sends the bids for both resources. The harder case is when *x* has only one marble (line 36). In this case it is necessary to cut the marbles so that the task can transfer one from *x* to *l*. If the task has reached the cut limit *C* (line 37), then its marbles cannot be cut further. The current allocation cannot be adjusted and the task should compute a new allocation and restart the bidding (line 38).

If the marbles can be cut then the task records the number of cuts and the new marble size. In line 42 it records that each requirement in the allocation has double the marbles it had before. The following lines simply transfer a marble from *x* to *l* and send the new bids for the corresponding resources.

The remaining part of the algorithm is the procedure `select-resources` used in line 9, which selects the cheapest allocation based on the current price of all resources. We do not show pseudo-code for this procedure. It works by sorting the eligible resources for each requirement by price, from low to high. It selects the cheapest resource for each requirement. In cases where the same resource can be used for multiple requirements, it breaks the tie by eliminating the resource from the requirement with cheaper subsequent resources. The computation time is linear in the number of qualified resources for the task.

## 4. EMPIRICAL EVALUATION

In Figure 1 in the introduction we compared the DMS algorithm to a centralized solver (Pseudo-Boolean) and showed that the solutions are of similar quality, but the DMS algorithm is significantly faster.

In this section, we present preliminary evaluation on anytime performance, adaptivity and scalability.

### 4.1 Problem Generator

We implemented a problem generator to evaluate the DMS algorithm (and our other algorithms, see [2]) which takes the following input parameters. The values in parentheses indicate a setting that produces problems of the same size and characteristics as the problem in Table 1. The random values are independently selected from Gaussian distributions.

The parameters are: number of tasks, number of resources, average task value (200), average task value standard deviation (0.2), average number of requirements per task (3), average number of requirements per task standard deviation (0.2), average resources per requirement (4), average resources per requirement standard deviation (0.005).

We felt that the chosen values for these parameters would provide a reasonable test of the algorithm’s capabilities because the average number of requirements per task (3) will make the randomly generated problems computationally complex (the problems are NP-Hard when tasks can have more than two requirements [13]).

The empirical evaluation results in this paper are all based on problems of the former characteristics.

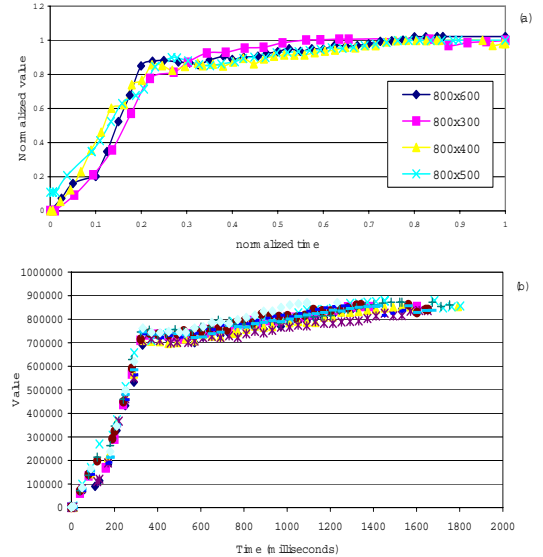
### 4.2 Anytime performance

One desired attribute from a solver is that good anytime performance (the solution quality always increases so that users can quit optimizing at any time).

From the empirical results (Figure 2), the DMS algorithm demonstrates a good anytime performance for very large size problems with different characteristics (the problem with 800 resources and 300 tasks (800x300) is under-constrained, the 800x400 ones are about lightly-constrained, and all other ones are highly-constrained).

Figure 2 shows that the DMS algorithm can produce good quality solution quickly. In about 25% of the time it achieves about 85% of the solution quality.

There is a distinctive knee in the solution profiles. The rate of increase of solution quality in the first phase is about 18 times that of the second phase. The sharp difference in the slope of the performance curve could enable users to quickly determine the amount of time and quality increase that can be achieved by allowing the algorithm to run to completion.



**Figure 2: Anytime performance of the DMS algorithm: (a) Solution profiles for 4 different size problems, (b) Solution profiles for 10 same size problems of 800 resources 400 tasks.**

### 4.3 Scalability

In experiments with randomly generated tasks of uniform characteristics, we found that the convergence time of our algorithm increased linearly with the size of problems, and that the percentage of tasks that it could satisfy remained constant (Figure 3). The first finding holds because the problem generator generates a constant number of possible resources per requirement (around a Gaussian distribution), independently of the total number of resources. The second finding is encouraging in that the quality of the solution does not seem to degrade with increasing problem size. (We expect the optimal percentage of tasks that can be satisfied to also be constant with increasing problem size because all problems are generated with the same characteristics.)

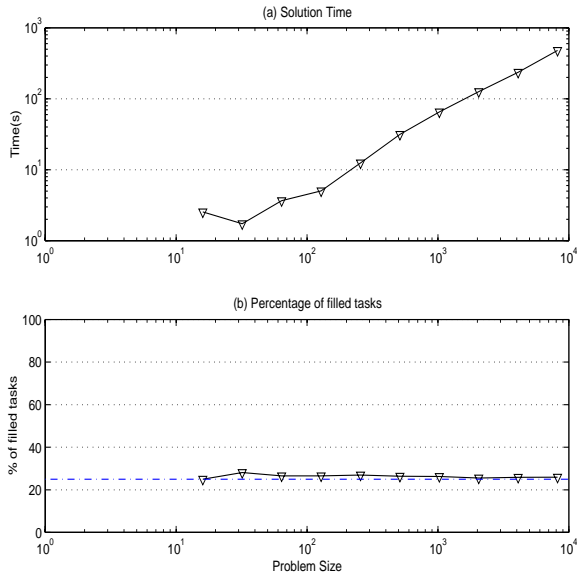
### 4.4 Adaptivity

One of the design desiderata for the algorithm was that it should be able to adapt rapidly to the changes to the problems (resource availability or tasks).

To evaluate the adaptivity of the DMS algorithm, the changes to resource availability were simulated by making some resources unavailable after the solver reached initial convergence. The solution profiles were recorded at a fixed time interval to measure how quickly the solver adapts to the introduced changes.

The preliminary results (shown in Figure 4) shows that the solver takes much less time to adapt to the changes than to reach initial convergence. The adapting time is only 15% of the time of reaching initial convergence. This finding suggested that the changes trigger only localized bidding and counter-bidding (otherwise the adapting process will take the same amount of time as the time for initial convergence).

In the simulations, the changes to resource availability could cause some winning task lose and start bidding again, which triggers other tasks to start counter bidding. Suppose a high-value task does not bid maximally on its resources,



**Figure 3: DMS appears scalable with problem size when evaluated against: (a) Solution time. (b) Percentage of satisfied tasks. Problem size  $n$  refers to a problem with  $n$  tasks and  $n$  resources.**

therefore these resources could appear affordable to a low-value task. By bidding on these resources, the low-value task could trigger a chain of unnecessary bids and counter bids, which slows the adapting process.

We hypothesized that the aggressive bidding strategy accelerates the adapting process by limiting the unnecessary bids and counter bids. Our hypothesis was supported by the preliminary results that the aggressive bidding strategy enables the solver to adapt to changes 4-5 times faster than incremental bidding strategies.

## 4.5 Suboptimality

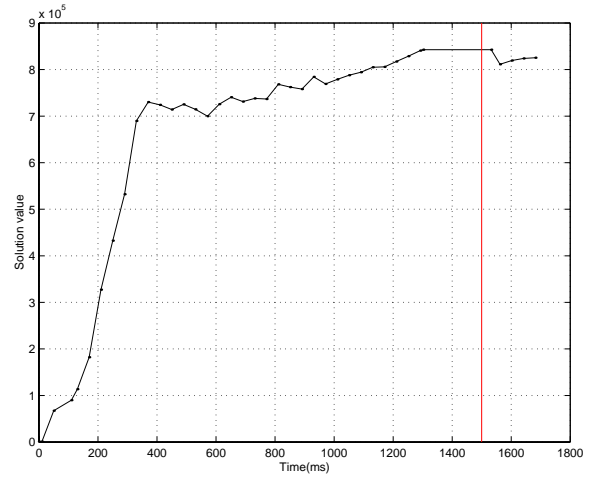
The DMS algorithm cannot guarantee optimality because the algorithm forces tasks to withdraw without doing a complete analysis to prove that a task does not belong to the optimal solution. This is also its strength: the solution converges quickly.

We identified two reasons that cause tasks to withdraw prematurely.

The first reason is that a task may withdraw prematurely because it incorrectly believes that every possible set of resources it could use costs more than it can afford. This happens because either (a) a task does not have an accurate picture of resource prices, or (b) other tasks have bid up the prices of resources but will later withdraw their bids. This problem is tied to the nature of distributed multi-agents environment.

The second reason is that resource prices only approximately embody the demands from tasks due to the aggressive bidding strategy. Low-value tasks could fail to reveal their demands through resource prices if the resource prices are already higher than they afford. And because of failure of demand revelation, high-value tasks can unintentionally block low-value tasks. Table 2 presents such an example – if Q bids *first* and selects A it will hold on to it forever.

One possible approach that we are rapidly developing is



**Figure 4: Preliminary adaptivity results: the DMS algorithm adapts rapidly to the changes of the disappearance of 20 resources after 1500 milliseconds (the adapting time is 15% of the time of reaching initial convergence) The problem had 400 tasks and 800 resources in the beginning.**

a generalization of the aggressive strategy called the incrementally aggressive bidding strategy. Using incrementally aggressive bidding strategy, a task always start with bidding just enough for the cheapest resource set and reserves the remaining money. If a task fails to acquire resources by bid adjustment, it will re-start a new try and bid more aggressively by allocating an extra amount of money from the reserved money. The extra amount is a certain percentage of the reserved money of the task. The percentage increases after each failure.

This incrementally aggressive bidding strategy diminishes the likelihood that tasks withdraw incorrectly because it gives lower-value tasks a chance to bid on resources and thus steer higher-value tasks to other resources.

The problem with the aggressive bidding strategy is that a high-value task may bid high on a resource A that a low-value task needs even though the high-value task could have used resource B. When the low-value task sees the high price of A it may withdraw because it cannot win. In order for both tasks to succeed, the low-value task needs to steer the high-value task away from A. The incremental aggressive bidding solves this problem because the high-value task starts with a lower bid on A, which the low-value task can outbid. If the high-value task can use an alternative cheaper resource it will do so, allowing the low-value task to also acquire the resources it needs.

Our initial evaluation of the incremental strategy shows that it can indeed lead to higher value solutions. However, this comes at a cost of additional messages and additional computation time. In our tests, the incrementally aggressive bidding strategy produced 4%-9% better solution quality but requires 5 times more messages and 4 times more computation time.

Given that the incremental strategy is highly adjustable (one can adjust the size of the initial bids and of the increments), we believe that the scheme can be fine-tuned to strike a balance between solution quality and computation

costs (messages and time). In the next phase of our work we will run experiments to understand this trade-off, allowing applications to specify trade-off between solution quality and computation costs.

Another possible approach to address the problem caused by the resource price approximation is to allow negotiation between tasks. By direct negotiation between tasks, a task could ask another task to bid on different resources if the requests changes are mutually beneficial.

Task (value)	Resources	
	A	B
Q (100)	1	★ ○ ★ ●
R (99)	1	★ ●

**Table 2: A minimal problem for which the DMS algorithm may find a sub-optimal solution.**

## 5. RELATED WORK

Market-based/economically-inspired approaches are inherently distributed and efficient ways to allocate resources among tasks. Auctions lie in the core of these resource allocation systems. To allow bundled preferences in resource allocation problems could enhance resource utilization efficiency in many real world domains, however, it increases computational complexity to find optimal solutions.

Walsh et al. [12] outline the fundamental choices for auction-based resource allocation schemes: (a) Vickery auctions, (b) combinatorial auctions, and (c) single-resource auctions.

In Vickery auctions, every resource requester has an incentive to report his true requirements to a centralized auction mechanism which can then make an optimal assignment of resources (internally solving an NP-complete problem) and report the assignments back to the requesters. This type of auction is not well suited for distributed resource allocation because of its single point of failure.

In single-resource auctions, each resource can auction itself off to the highest-value task based solely on its local bid information. For resource allocation problems without complementarities, i.e., tasks with only a single requirement, a traditional single-resource auction is guaranteed to compute an allocation within a fraction of the optimal solution – as the bid increment is reduced, the solution approaches optimality. But the traditional single-resource auctions have no mechanisms to express bundled resource preferences.

To address inefficiencies of the traditional single-resource auctions, several researchers proposed combinatorial auctions [6, 1, 3, 9]. In combinatorial auctions, a task could send a bid for a set of resources {A,B,C}. We suspect that combinatorial auctions are generally also inapplicable to distributed resource allocation, because they need a large number of messages to coordinate between themselves, as they cannot individually auction themselves but must bundle up with others to be bid on in combination.

Aiming at the desired attributes from an approximation technique, the DMS algorithm use a unique approach. Instead of using one bid to express the bundled preferences, in the DMS algorithm, tasks use a bundle of separated bids and coordinate them by bid adjustment.

## 6. FUTURE WORK

We would like to evaluate the tradeoff of different bidding strategies on performance.

We would like to integrate the “Synthetic Pheromone Control” mechanism into the DMS algorithm. Van Parunak et al., ([5],[10]) proposed the “Synthetic Pheromone Control” mechanism that helps individual task agents decide when to bid, when to skip bidding, and when to give up by imitating the dynamics of insects. They validated the mechanism by extending an early version of the DMS algorithm (called “Marblesize” in [2]). The extension to the DMS algorithm reduces the number of messages to reach comparable solutions.

We would like to perform theoretical analysis of the factors attributing to the hardness of the combinatorial resource allocation problems. So by varying the factors in the problem generator, different characteristic problems can be generated to evaluate the algorithm.

We would like to enhance the problem generator to produce problems with a known optimal solution (by constructing the solution first, then generating resources that make the solution possible, and by then loosening up the constraints so that that solution becomes hard to find), and to assess how the DMS algorithm performs for these problems.

We have run the DMS algorithm in a distributed agents simulator which correctly simulates the asynchronicity of message delivery between agents, but does not simulate message delays, message losses, and ungraceful agent death. We would like to study the effect of these factors on the performance of our algorithm.

We would like to improve the Dynamic Marble Size algorithm so that giving it infinite time will result in it finding an optimal resource assignment. Currently our algorithm can’t improve after certain amount time because our algorithm does not perform exhaustive backtracking (not even with an infinitely small Marble cut limit and no restart limit). The challenge is to ensure that any future extension of the algorithm still exhibits good anytime performance.

We are also expanding the problem statement by introducing time (making it a scheduling problem) and by introducing resource learning (a resource get more qualified by participating in some tasks).

## 7. CONCLUSION

The preliminary results indicate that the DMS algorithm not only is competitive in quality and superior in speed with a well-known centralized technique, but also exhibits good anytime performance. The solution profile shows the particularly sharp transition, which could offer users an indication of when it is profitable to wait longer. The empirical results also indicate the DMS algorithm exhibits good scalability (approximately linear with problem size).

Based on the empirical results, we believe that the DMS algorithm is promising as a good distributed approximation approach for NP-Hard combinatorial resource allocation, and we are actively pursuing the extensions described in Sections 4.5 and 6.

## 8. ACKNOWLEDGMENTS

We gratefully acknowledge funding by DARPA ITO ANTS program (Contracts No. F30602-00-2-0533 and F30602-99-1-0524). Part of this work was funded by AFOSR under

grant number F49620-01-1-0341. We thank H. Van Dyke Parunak for personal communication about [5].

## 9. REFERENCES

- [1] J. S. Banks, J. O. Ledyard, and D. P. Porter. Allocating uncertain and unresponsive resources: An experimental approach. *RANF Journal of Economics*, 20:1–25, 1989.
- [2] M. Frank, A. Bugacov, J. Chen, G. Dakin, P. Szekely, and B. Neches. The marbles manifesto: A definition and comparison of cooperative negotiation schemes for distributed resource allocation. In *Proceedings of the 2001 AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, pages 36–45. American Association for Artificial Intelligence, November 2001.
- [3] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of IJCAI-99, Stockholm, 1999*, 1999.
- [4] R. McAfee and J. McMillan. Analyzing the airwaves auction. *Journal of Economic Perspectives*, 10(1):159–175, 1996.
- [5] V. Parunak, S. Brueckner, J. Sauter, and R. Savit. Effort profiles in multi-agent resource allocation. In *Proceedings of Autonomous Agents and Multi-Agent Systems, Bologna, Italy*, pages 248–255. ACM, July 2002.
- [6] S. Rassenti, V. Smith, and R. Bulfin. A combinatorial auction mechanism for airport slot allocation. *Bell Journal of Economics*, 13:402–417, 1982.
- [7] M. Rothkopf, A. Pekec, and R. Harstad. Computationally manageable combinatorial auctions. volume 44, pages 1131–1147, 1998.
- [8] T. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *In Proceedings of the National Conference on Artificial Intelligence (AAAI), Washington, D.C., July 1993.*, pages 256–262, 1993.
- [9] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.
- [10] J. Sauter, H. V. D. Parunak, S. Brueckner, and R. Matthews. Tuning synthetic pheromones with evolutionary computing. In *Workshop on Evolutionary Computing and Multi-Agent Systems (GECCO)*, pages 321–324, San Francisco, CA 2001.
- [11] J. Walser. Solving linear pseudo-boolean constraint problems with local search. In *In Proceedings of the 14th National Conference on Artificial Intelligence, AAAI-97, Providence, RI, (1997)*.
- [12] W. Walsh, M. Wellman, P. Wurman, and J. MacKie-Mason. Auction protocols for decentralized scheduling. In *Eighteenth International Conference on Distributed Computing Systems (ICDCS-98)*, Amsterdam, The Netherlands, 1998.
- [13] W. Zhang. Modeling and solving a resource allocation problem with soft constraint techniques. *Technical Report: WUCS-2002-13*, May 2002.