

The Marbles Manifesto: A Definition and Comparison of Cooperative Negotiation Schemes for Distributed Resource Allocation

Martin Frank⁺, Alejandro Bugacov⁺, Jinbo Chen⁺, Gordon Dakin^{*}, Pedro Szekely⁺, Bob Neches⁺

(+) Information Sciences Institute of the University of Southern California
4676 Admiralty Way
Marina del Rey, California 90292

(*) Crystaliz, Inc.
9 Damon Mill Square, 4D
Concord, MA 01742

{frank,bugacov,jinbo}@isi.edu, gad@crystaliz.com, {szekely,rneches}@isi.edu

Abstract

Marbles schemes are a family of cooperative and adaptive algorithms for distributed resource allocation problems. Long-term goals for these schemes are fault-tolerance and real-time performance in which a good timely solution is preferable to an optimal but too late solution. This paper reports work in progress where we compare the performance and analyze characteristics of different Marbles schemes and centralized solvers working on large scale and complex resource allocation problems.

Introduction – the Marbles Vision

Recent advances in miniaturization and robotics have led to interest in research on the “autonomous agents” that make up a team of, say, robotic soccer players or Unmanned Combat Air Vehicles (UCAVs). These agents can act individually but are better off coordinating with their peers. A subset of this problem is distributed real-time resource allocation – deciding under time pressure which soccer player will take the final shot on the goal, or which UCAVs will neutralize a newly discovered enemy threat.

There is a spectrum of approaches for distributed real-time resource allocation, ranging from no communication at all (physics-based approach: agents observe each others’ behavior but do not explicitly communicate, much like a wolf pack closing in on prey) to communication of the full rationale of behavior (argumentation approach: agents back up requests to others by an argument of why they should grant it).

In this continuum, our Marbles schemes and all other “market-inspired” approaches fall in-between. Compared to a purely physics-based approach, they obviously use more messages yet also explore a more complex set of

alternatives. Compared to the argumentation approach, they exchange messages of smaller complexity, yet the prices set by supply and demand can possibly communicate rationale in an alternative, more compact fashion, and potentially steer the group of agents to sensible behavior via “the invisible hand of the market”.

External Marbles Scheme Properties

“Marbles” schemes¹ are a family of resource allocation algorithms that are characterized by the following properties:

Distributed. Each task only knows about its local requirements, and communicates with potential resources for those requirements exclusively through messages. Hence, each task and each resource can – but does not have to – be located on a different machine.

Cooperative. Marbles schemes are not designed to tolerate malicious participants, which distinguishes our research from work on e.g. electronic commerce and automated auctions; we believe that security against external attack of cooperative negotiation schemes is best located at a lower level (such as the message transport and encryption level). The cooperative nature of the negotiation also means that tasks participating in resource auctions can altruistically commit suicide by permanently withdrawing, and therefore lowering resource prices that possibly help others succeed. The distributed algorithms for concluding that tasks are unlikely to succeed further distinguishes our work from work on competitive auctions.

Adaptive. A Marbles scheme can adapt a current partial solution to a new situation rather than having to re-compute the new solution from scratch. This makes them applicable in cases where “the world can’t stop while a solver computes a solution for everyone”, that is, in cases where

¹ The name is not an acronym. Bob Neches likened the agent behavior to “kids trading marbles” in an early design discussion, and the name stuck.

the time interval between situation changes is smaller than the total running time of a non-adaptive centralized solver.

Real-Time. The individual negotiation participants should be explicitly aware of time and adapt their behavior based on how much time is left.

Fault-Tolerant. A Marbles scheme should be robust against a set level of message loss, in the sense of being able to make statements like “given an average message delay of 2 seconds and a message loss rate of 5%, this negotiation has a 99% likelihood of concluding in less than 3 minutes”. Obviously, no message-based scheme can ever be robust in the sense of making a 100% real-time response guarantee if there is a non-zero chance of a message getting lost.

Internal Marbles Scheme Properties

We further characterize Marbles scheme by their “internal” properties; that term is accurate in the sense of being more linked to our approach than the above “external” ones. However, these choices do “shine through” to the user level, so the distinction between external and internal is not as sharp as it may sound.

Domain-based task valuation. Marbles schemes put a value on the execution of tasks that is quantitative and that has meaning to domain practitioners.¹ The value of resources is exclusively derived from the value of the tasks they enable; they have no intrinsic domain value of their own.

Lack of inflation. We do not allow inflation (the artificial introduction of currency not backed up by domain value during negotiation) because the overall solution can otherwise not be verified in domain terms. For example, imagine that a negotiation scheme introduces inflation by increasing the value of tasks the longer they go unfilled during negotiation: consequently, the basis of the proposed overall solution cannot be analyzed by a domain expert without understanding the negotiation algorithm. Thus, the problem with mixing intrinsic task value and negotiation-scheme-dependent artificial “value” is that it would make the term “domain” currency meaningless.

Ever-fluctuating prices. In the prototypical open-outcry auction, participants bid until no one wants to bid higher, and the highest bidder then owns the resource from that point in time on. In contrast, Marbles schemes resources continually auction themselves -- the auctions never “close”. That is, you can only be the current, not final, winner, of a resource -- if the situation changes because, e.g., a new high-valued task appears you will lose it.

¹ For example, the value of executing a training mission in our Marine Corps application is measured in “combat readiness percentage” (CRP) gain. For example, a pilot may gain 0.3% CRP from participating in a mission. The Marbles schemes themselves are unaware of the meaning of that number - they simply see a task of domain value 300 (we multiply Marines CRP by 1000 to convert it to an integer).

Formal Problem Statement

Below we introduce the minimalistic problem statement that our existing Marbles schemes operate on. In the future, we will continually expand the problem definition to, e.g., be able to shift tasks in time, to introduce a notion of equity in resource use, and so on, but the current problem already captures the essential challenge of distributed resource allocation. Note that none of our existing Marbles schemes presented below exhibits all of the desirable properties outlined in the Marbles Vision section; in particular, none of them can make real-time response or fault tolerance guarantees yet.

Problem

There is a collection of available *resources* that are characterized by a unique name (and nothing else). There is a collection of possible *tasks* that are worth a fixed domain value if they are executed. They need to acquire one resource for each of their *requirements* to be executed. Each resource can only be used for at most one task. Each task knows in advance which resources are suitable for its requirements. (Thus, we neglect a prior “resource discovery” phase.)

This problem is very complex if tasks have multiple requirements (“complementaries” exist, in economic jargon) - it would be trivial if each task had just a single requirement.

Solution

A solution consists of an assignment of resources to requirements such that every task has either none or all of its requirements filled. The quality of a solution is measured by the sum of the domain values of its satisfied tasks; a higher sum indicates a better solution.

Running Example

We will use the following example to explain how the various Marble scheme variants operate. There are four resources called *A*, *B*, *C*, and *D*. There are two tasks called *Q* and *R* of domain value 300 and 100, respectively. Each of the two tasks has two requirements that can be filled by the resources indicated with a triangle below. This particular example was chosen because it is small yet leads to backtracking behavior if schemes assign resources to requirements from left to right (as they usually do). The optimal solution of domain value 400 is obvious (*Q* gets *A* and *D*, *R* gets *B* and *C*).

		A	B	C	D
Q (300)	1	▲	▲		▲
	2	▲			
R (100)	1	▲		▲	
	2		▲		

A Rough Taxonomy of Solvers

We will present a number of “solvers” – any piece of code that produces a solution given a problem in the above terms. Our research interest is exclusively in fully distributed resource allocation schemes, but we have also build a number of centralized solvers for comparison purposes. In addition, some of our Marbles variants have so far only addressed part of the challenge in a distributed way because we have not had the time to make them fully distributed.

All Marbles solvers fundamentally perform two tasks: assigning resources to the highest-bidding tasks (“allocation”), and eliminating tasks from competition (“elimination”) because they drive up the prices for others without seeming to have a chance of obtaining all of their needed resources. Each of the variants indicates if it solves each phase in a distributed or centralized fashion.

Marbles2 [allocation: distributed, elimination: centralized]

The main inspiration behind this Marbles variant is that the cost of a resource should be defined by the value that the second-highest bidder places on it (the “displacement” or “opportunity” cost of the resource). Consequently, resources cost zero if no one else wants them.

Message Protocol

Task to resource: bid(amount), withdrawal(); resource to task: loss(), win(amount that can be lower than bid), priceChange(can be up or down but recipient is still winning).

The Running Example under Marbles2

In this variant of our Marbles schemes, tasks attempt to fill each requirement one at a time, bidding all of their available value to satisfy the next unfilled requirement. Exclamation marks indicate currently winning bidders.

1. *Q* simultaneously bids 300 on *A*, *B*, and *D* to satisfy its first requirement. *R* bids 100 on *A* and *C*.

		A	B	C	D
Q (300)	1	▲300	▲300		▲300
	2	▲			
R (100)	1	▲100		▲100	
	2		▲		

2. *Q* obtains *A* for 100 (the cost as a displacement cost is determined by the second highest bidder). It reacts by bidding 200 for *B* and *D* (because it has internally determined that it is better off by using *A* for its second requirement¹, and has already spent 100 of its 300 value for obtaining a resource).

¹ Interestingly, we handle internal assignments by an internal use of the very same Marbles scheme where each

		A	B	C	D
Q	1	▲	▲200		▲200
	2	▲100!			
R (100)	1	▲100		▲100	
	2		▲		

3. *R* wins *C* for 0 (as there are no competing bidders). It reacts by bidding 100 on *B* to obtain its second resource, and by completely withdrawing its bid for *A* (it already has a resource for its first requirement for free; otherwise it would have bid on *A* whatever it had to pay for *C* minus the minimum bid increment/decrement).

4. *Q* gets notified that the price of its *A* dropped to 0 (because all competition disappeared). It thus now increases its bids for *B* and *D* to 300.

		A	B	C	D
Q (300)	1	▲	▲300		▲300
	2	▲0!			
R (100)	1	▲		▲0!	
	2		▲100		

5. *Q* wins *B* for 100 (because that’s *R*’s bid). It is now satisfied, but bids 99 for *D* (a cheaper resource is always preferable) just in case.

6. *Q* wins *D* for 0 because no one else wants it. It withdraws its bid for *B* because nothing beats a free resource.

7. *R* gets notified that it is now the winner on *B* (also for 0). The scheme is in a terminal state unless the environment changes (new high-value tasks could steal resources, for example).

		A	B	C	D
Q (300)	1	▲	▲		▲0!
	2	▲0!			
R (100)	1	▲		▲0!	
	2		▲0!		

Thus, in the end it has been determined that there is no competition for resources at all – all tasks can be satisfied with the available resources.

Experience and Limitations of Marbles2

As is evident from the curves in the Evaluation section, this Marbles scheme (the first one written) tends to produce the lowest-quality solutions and also require largest number of messages. We believe that the latter is true because tasks bid on *all* qualified resources for every requirement, and in addition the scheme bids down prices one by one in epsilon

requirement gets the same constant value to bid on resources won by the task, but we won’t go into the details of that here.

increments (rather than in logarithmic sizes as some of the schemes below do). We have not had the time to investigate why the former is true.

Msmarbles [allocation: distributed, elimination: distributed]

In the Msmarbles (Multi-Sized Marbles) scheme each task has the same number of marbles. The size of each marble is the total value of the task divided by the number of marbles that the task has. Consequently, tasks with higher value have larger marbles.

Message Protocol

Tasks bid on resources by placing marbles on them. A task can bid one marble at a time, and must wait for a price-update message from the resource before placing another marble. Resources grant themselves to the task that has placed the largest value (not largest number) of marbles on them. When a task runs out of marbles, it can withdraw its marbles from a resource. When it does so, the resource returns all marbles to all tasks that have bid on it, with one exception. The resource keeps one marble from the current winner. In essence, the price for the current winner goes down to one marble.

When a task withdraws its marbles from a resource, it will not attempt to bid on that resource again unless it has available at least one more marble than it got back. We call this number of marbles the task's "block amount" on a given resource. Block amounts always go up, and eventually will reach the point where a task cannot win an allocation of resources for all its requirements because the block amounts on the required resources exceeds the total number of marbles that a task has. When this happens, the task voluntarily withdraws from competition by withdrawing all marbles from all resources. The scheme converges because tasks keep withdrawing until all remaining tasks succeed.

The intuition behind Msmarbles is that if the valuation of resources emerges incrementally, in small steps, it will be more accurate. This will enable tasks to make more informed decisions about where to place or withdraw marbles and when to give up, and thus lead to a better solution.

The timing of withdrawals is critical. It is advantageous to delay withdrawals as long as possible because by that time other tasks may have withdrawn first and hence they become subject to the eventually deadly block-amounts. In order to diminish the advantages of delays, we made each task have the same number of marbles, each task bid a single marble at a time, and each task wait for a reply before bidding the next marble. Richer tasks will have an advantage, as they should, because they can delay placing marbles. Poorer tasks may need several bids to catch up to the bid of a richer task, hence allowing the richer task to hold on to its marbles for a longer time.

One of the main qualities of Msmarbles is that multiple medium-sized tasks can together bid up the valuation of multiple resources forcing a richer task to become subject

to several block amounts, and eventually forcing it to give up. This enables the Msmarbles scheme to make trade-offs between multiple medium-sided tasks and few richer tasks.

The Running Example under Msmarbles

The first graph below shows the behavior of Msmarbles in the simple running example. In this example we gave each task 8 marbles (twice the number of resources). Task Q's marbles are worth 37.5 points, whereas Task R's marbles are worth 12.5 points. Lines labeled A, B, C and D represent the valuation of resources A, B, C and D over time. Lines Q-A, Q-B and Q-C represent the amount task Q has bid for resources A, B and C. R-A, R-C and R-B represent task R's bids. Initially, both tasks bid on A. Then they bid on the next resource they need: Q bids on B and R bids on C. When responses come back, Q learns that it is winning both resources. Task R learns that it is losing on A and winning on C. Task R must now bid for B, its only choice for requirement 2, and it keeps placing marbles on it until it outbids task Q. When Q is outbid it determines that the price increment to win D is 0+ (i.e., any amount larger than 0), and hence places a marble on D. At this point, both tasks are fulfilled and they stop bidding.

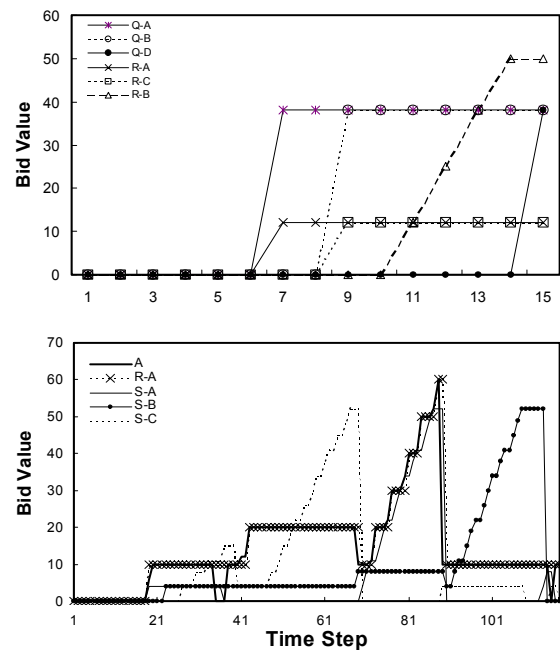


Figure 1: Bid values sequence for the Msmarbles scheme

The second graph shows a more complex example where not all tasks can be fulfilled, and tasks need to withdraw bids and eventually withdraw from competition. The graph shows the evolution of the price for resource A, the amount task R bids on A (R-A), and the amounts task S bids on resources A, B and C (S-A, S-B and S-C respectively). In this example there are 4 tasks and 8 resources (not all shown in the graph), and S is the poorest task with 60 points. The graph shows how S first went on a bidding war for resource C and eventually withdrew because it needed marbles to bid on other resources. Similarly, S had losing

bidding wars for resources A and B. A, B and C were the only choices that S had to fulfill one of its requirements, and after the three withdrawals, the block amounts went so high that S would have had to use all its marbles to win one of those resources, leaving no marbles to win resources for its other requirements. At that point, S gave up, enabling the other three tasks to succeed. The price for A went down sharply enabling the task that needed it to use its marbles for other resources. (The second problem comes from an example that Walsh uses to demonstrate that simple auctions cannot be used to compute optimal resource allocations when complementarities are present. For this particular example -- but by no means for all -- Msmarbles computes the optimal solution).

Experience and Limitations of Msmarbles

The Msmarbles algorithm has not been as thoroughly evaluated as the others, so that implementation bugs disqualify it from the systematic comparison with the other algorithms in the Evaluation section. The solutions of the examples it does run are of high quality (defined as “close to the best solutions of other schemes”). However, the scheme is also one of the slowest, using significantly more messages than the others.

Marblesize [allocation: distributed, elimination: distributed]

The motivation of the Marblesize scheme is to allow trading off the quality of the solution against the number of messages needed through different pre-specified Marbles “sizes”.

In the Marblesize scheme, no resource is free and the price for a resource is determined by the current highest bid. To acquire a resource, a task needs a certain number of marbles. Marbles have given size that can be subdivided in equal parts an arbitrary number of times. The size of the marbles represents the minimum amount a task can bid on a resource. For each task, the initial marble size is equal to the task value divided by the number of requirements in that task. At the beginning, each task selects a possible combination of resources for its requirements and bid one marble on each of them. After that the bidding mechanisms continues based on the following rules: (1) If a task has more than one possible combination of resources, it chooses the cheapest one based on the current bids on those resources and allocate all its value among them but placing at least one marble on each resource. (2) A task wins if it is winning on all of its current bids. (3) A task loses if it is losing on all of its current bids. (4) A task that is winning on some of its bids can move one marble at a time from a winning resource bid to a losing resource bid. (5) A task can cut its marble size until the marble size is less than the minimum marble size allowed. (6) A losing task tries another resource combination and repeats the process. If it cannot find a new combination of resources it commit suicide.

Message Protocol

Task to resource: bid (amount), withdrawal (); Resource to task: loss (), win ().

The Running Example under Marblesize

First round: Q: Marble size (150) R: Marble size (50).

		A	B	C	D
Q (300)	1	▲	▲150!		▲
	2	▲150!			
R (100)	1	▲50		▲	
	2		▲50		

The two requirements of task Q are winning so no changes happen in that task. In task R both requirements are losing. Since the first bidding proposal is no good it tries a second bidding proposal [C,B] while keeping a marble size of 50.

Second round: Q: Marble size (150) R: Marble size (50).

		A	B	C	D
Q (300)	1	▲	▲150!		▲
	2	▲150!			
R (100)	1	▲		▲50!	
	2		▲50		

Now R is winning on C that nobody wants and tries to move its marbles from C to B.

Third round: Q: Marble size (150) R: Marble size (25).

R cuts its marble size to the minimum size of 25. Although R is still winning on C, it cannot move its marble anymore because each resource needs at least one minimum size marble. So R’s second proposal is declared dead. Since it cannot try a third proposal, R is declared dead and the process terminates. (Thus, the scheme fails to find the optimal solution for this simple problem -- nevertheless it is the single best scheme we have for large problems, as will become evident in the Evaluation section.)

		A	B	C	D
Q (300)	1	▲	▲150!		▲
	2	▲150!			
R (100)	1	▲		▲25!	
	2		▲75		

Experience and Limitations of Marblesize

The Marblesize scheme has the unique ability to trade off solution quality against speed of convergence. The table below shows the impact that the minimum marble size has on the total number of messages and the quality of the solution. As is evident, it is possible to control the minimum marble size to trade-off solution quality for computational time. In this example, an increase of less than 1% of solution quality is paid by a 10 fold increase in the number of messages.

Number of Subdivisions	Total Number of Messages	Maximum Value of Solution
4	7986	19017
3	6013	18950
2	3635	18894
1	1250	18880
0	793	18649

In terms of scalability with respect to problem size, the number of messages and solving shows a phase transition behavior where, for fixed number of resources, the number of messages increases sharply with the number of tasks until it reaches a certain value where starts decreasing again, resembling the critical behavior observed in other combinatorial problems. We believe that this is due to the fact that for large number of tasks the lack of resources leads to quick suicide of most tasks with large requirements, thus the competition quickly decreases along the process.

Grabmarbles [allocation: distributed, elimination: distributed]

Grabmarbles is a variation of the Marblesize scheme which relies on heuristic selection of resource combinations. As in Marblesize, a task bids on the cheapest set of resources that will satisfy its requirements. Unlike the Marblesize scheme, rebidding is not permitted after a losing resource bid, and bids are not based on marble sizes. Instead, a task agent submits a bid that is a heuristic evaluation of the task, based on its domain value, number of task requirements, and number of alternative resources. A task only bids for resources whose prices (the evaluations of the currently winning tasks) are less than the bidding task's own evaluation. When a task agent loses a bid, it gives up on the current resource set and tries another if possible. The heuristics used by Grabmarbles were originally applied to Marbles2, and improvements in solution quality motivated the application of those heuristics to Marblesize.

A heuristic task evaluation function is defined for a given task and resource. (Note that this heuristic function actually violates the "no inflation" rule for Marbles schemes, making it impossible to use the prices paid for resources as an indication for the contribution of domain value. This has not been an issue because we have only measured pure solution quality so far.) The following example of a task evaluation function rewards tasks that have only one or two alternative resources to choose from, otherwise penalizing the task according to its number of requirements.

```
function taskeval (dval, reqs, alts)
  if alts = 1 return dval / reqs;
  else if alts = 2 return dval / (2 * reqs);
  else return dval / (4 * reqs);
```

Message Protocol

Task to resource: bid (amount), withdrawal (); Resource to task: loss (), win ().

The Running Example under Grabmarbles.

First round: Q selects A and B.

		A	B	C	D
Q (300)	1	▲	▲37.5!		▲
	2	▲150!			

The running example is analyzed here using the task evaluation function described above. All resources are initially free, so task agent Q selects A and B. Q's domain value of 300 and its 2 requirements yield an evaluation of 37.5 for resource A, while its evaluation with respect to A (150) reflects the fact that A is Q's only alternative resource for requirement 2.

Second round: R selects B and C.

		A	B	C	D
R (100)	1	▲		▲25!	
	2		▲50!		

The possible resource sets available to task agent R are (A,B) and (C,B). The cheaper alternative is (C,B), whose total price of 37.5 is due to Q's currently winning bid. Like task Q, R's second requirement has only one qualified alternative, so R's task evaluation with respect to resource B comes to 50. Task Q is outbid for resource B, so it withdraws its bids and tries another resource combination.

Third round: Q selects A and D.

		A	B	C	D
Q (300)	1	▲	▲		▲37.5!
	2	▲150!			
R (100)	1	▲		▲25!	
	2		▲50!		

Task agent Q finally selects price-free resources A and D. Both tasks are now satisfied, reaching the optimal solution domain value of 400, with 15 messages passed.

Experience and Limitations of Grabmarbles

The Grabmarbles scheme produces solutions that are comparable to those of Marblesize, with a relatively small number of messages. The use of heuristics in evaluating each task's "deservedness" with respect to different resources has a globally beneficial effect on resource allocation. In the Marblesize scheme, the relative merit of competing tasks is resolved through the process of rebidding and transfer of funds between resources. In Grabmarbles, the selection of resources through heuristics tends to direct the task agents toward resources they can realistically attain, while avoiding resources that are critical to other tasks. The focus on globally beneficial resource selection helps to eliminate the need for rebidding.

The choice of task evaluation formula used in Grabmarbles has not yet been automated. The quality of

solutions is greatly affected by how well suited the evaluation formula is for a particular problem set. The results shown in the curves in the Evaluation section were obtained using the following evaluation function.

```
function taskeval (dval, reqs, alts)
return dval / reqs - 2 * alts;
```

This evaluation formula fails to yield the optimal solution domain value for the running example problem. The previous formula emphasizes the lack of resources available to a task, while the above formula only uses this as a tie-breaker. A hybrid evaluation formula, combining features of the two shown, has produced good solutions to all of these problem sets. But there remains a need for the automatic selection of an appropriate formula for a given problem, based on the distribution of task requirements per task, and alternative resources per requirement.

Brute-Force [allocation: centralized, elimination: centralized]

We have built a trivial centralized brute-force solver that enumerates all possible solutions and then picks the best one. It is impractical for more than about 15 tasks and 30 resources but serves its purpose in producing small-size challenge problems for the Marbles schemes for which the optimal solution is known.

Random [allocation: centralized, elimination: centralized]

Similarly, we have built a solver which synthesizes a random solution, keeps it if it beats the previous one, and keeps doing this until it exceeds a given time limit. We have used it to establish lower bounds on the solution quality for large-size problems.

Simulated Annealing [allocation: centralized, elimination: centralized]

We have implemented a Simulated Annealing (SA) solver [Kirkpatrick 1983] to further compare the results of the different Marble solvers against well-known centralized schemes. The SA algorithm seeks to escape local maximum by accepting downhill moves with a probabilistic model based on statistical mechanics. In our implementation of SA we start by randomly assigning resources to tasks until all resources are allocated. Then, for a number of *maxFlips* times, we perturb or flip the state of the system to a neighboring state by randomly picking a task, a requirement from that task and a new resource for that requirement from its list of eligible resources. We evaluate δ , the change in the total value, and always accept the move if $\delta \geq 0$. If $\delta < 0$, we accept the move with probability $\exp(\delta/T)$, where T is the *temperature* parameter. We repeat this procedure for different values of T , starting with a high value of T and decreasing it following a geometric scheduling such that $T_{i+1} = 0.5 * T_i$.

Experience and Limitations of the SA implementation

In terms of performance the SA solver ranks very close to but actually below the Marblesize solver. In certain problem instances SA beats Marblesize in finding a higher value in comparable execution size but on average Marblesize beats SA. SA provides the maximum number of flips (*maxFlips*) as its mechanism for externally controlling or trading-off quality of solution for execution time, similar to Marblesize using marble granularity for the same purpose. Even for a surprisingly low values of *maxFlips*, SA finds solutions within a few percent of the highest value with a significant speed up in solution time. With such a low value of *maxFlips*, SA is our “most efficient” solver (as measured by dividing solution quality by running time).

SAT Encoding [allocation: centralized, elimination: centralized]

We have also implemented a centralized SAT solver by encoding the resource allocation problem into Boolean satisfiability formulas in conjunctive normal form (CNF). In this approach, the allocation of resources to tasks is obtained by finding truth assignments to the resulting formulas. To use satisfiability testing for optimal allocation of resources we turn to the problem of finding valid assignments of resources *for at least k* (with $k \leq N$, the total number of tasks) tasks and then do a binary search to find the maximum k . This problem can then be encoded into a CNF formula of the following form:

$$f = f_k \wedge f_{cross} \wedge f_1 \wedge f_2 \wedge \dots \wedge f_N$$

Where f_k is responsible for switching on at least k of the variables representing the N tasks, f_{cross} precludes resources from being assigned to more than one requirement and f_i ($i=1,2,\dots,N$) selects eligible resources within each individual task.

SAT encoding of the running example

To encode the running example presented above for at least two tasks ($k = 2$) filled we define the following 13 boolean variables. First we introduce the tasks variables: t_1 and t_2 , that represent each task in the formula. Then we define the resources variables $A_{11}, A_{12}, A_{21}, B_{11}, B_{21}, C_{21}$ and D_{11} . Where $A_{ij} = \text{TRUE}$ indicates the assignment of resource A to task i requirement j . To select at least 2 different tasks variables we introduce four additional variables p_1, p_2, r_1, r_2 with the condition that $p_1 \rightarrow \neg r_1, p_2 \rightarrow \neg r_2, (p_1, r_1) \rightarrow t_1$ and $(p_2, r_2) \rightarrow t_2$. With this variables definition, the formulas introduced above take the following form:

$$f_{k=2} = (p_1 \vee p_2) \wedge (r_1 \vee r_2) \wedge (\bar{p}_1 \vee \bar{r}_1) \wedge (\bar{p}_2 \vee \bar{r}_2) \wedge (t_1 \vee p_1 \vee r_1) \wedge$$

$$(t_1 \vee \bar{p}_1) \wedge (t_1 \vee \bar{r}_1) \wedge (t_2 \vee p_2 \vee r_2) \wedge (t_2 \vee \bar{p}_2) \wedge (t_2 \vee \bar{r}_2)$$

$$f_{cross} = (\bar{A}_{11} \vee \bar{A}_{12}) \wedge (\bar{A}_{11} \vee \bar{A}_{21}) \wedge (\bar{A}_{12} \vee \bar{A}_{21}) \wedge (\bar{B}_{11} \vee \bar{B}_{21})$$

$$f_1 = (\bar{t}_1 \vee A_{11} \vee B_{11} \vee D_{11}) \wedge (\bar{t}_1 \vee \bar{A}_{11} \vee \bar{B}_{11}) \wedge (\bar{t}_1 \vee \bar{A}_{11} \vee \bar{D}_{11}) \wedge (\bar{t}_1 \vee \bar{B}_{11} \vee \bar{D}_{11}) \wedge (\bar{t}_1 \vee A_{12})$$

$$f_2 = (\bar{t}_2 \vee A_{21} \vee C_{21}) \wedge (\bar{t}_2 \vee \bar{A}_{21} \vee \bar{C}_{21}) \wedge (\bar{t}_2 \vee B_{22})$$

We solve the resulting formula f using a Java implementation [Jackson] of the WSAT [Selman 1993] solver. One can verify that f evaluates to TRUE by setting A_{12} , B_{22} , C_{21} , D_{11} , t_1 , t_2 , p_1 and r_2 to TRUE and all other variables to FALSE, which yields the correct solution for the running problem.

Experience and Limitations of the SAT Encoding

Our current SAT-based solver performs very well compared to Marblesize and Simulated Annealing for small and medium size problems (i.e., $N \approx 50$). For larger problems (e.g., $N=100$) the solution time degrades about an order of magnitude compared to Marblesize and SA but it is still able to produce high value results. By controlling the number of solutions that we ask WSAT to generate, we can externally trade-off solution quality with execution time and the solver can sometimes find solutions within less than 5% of the best value found with SA but with 10 to 20 times speedup. Another advantage of this approach is that it can be used to rapidly estimate the maximum number of filled tasks without having to search for the optimal solution.

In its current implementation the SAT-based solver is fully centralized but the same SAT encoding approach can be combined with Marbles or other distributed market mechanisms [Walsh 1998] to produce a distributed solver.

Evaluation

We evaluated the performance of the different solvers described above on synthetic problems that have the same characteristics of the problems stated above but with arbitrary number of resources and tasks. The problems were generated by randomly assigning to each task a certain number of requirements and a task value. The set of possible resources for each task was also randomly selected from the original resource pool. These random values were independently selected from three different Gaussian distributions. Thus, the dominant parameters in describing a given problem are: a) number of tasks, b) number of resources, c) r , average number of requirements per tasks, d) v , average task value and e) p , the average number of possible resources per requirement.

In Figure 2, we compare the performance of our solvers for 30 different problems with 100 resources and 100 tasks. (The problems were generated with $r=4$, $v=300$ and $p=10$.) The parameters we use to evaluate performance are the total value (i.e., the sum of the task values for all filled tasks) of a solution and the (execution) time it took the solver to find that solution. In the upper and lower panel of Figure 2, we compare the results for total value and time, respectively. We see that Marblesize, Grabmarbles and Simulated Annealing can find comparable results of the

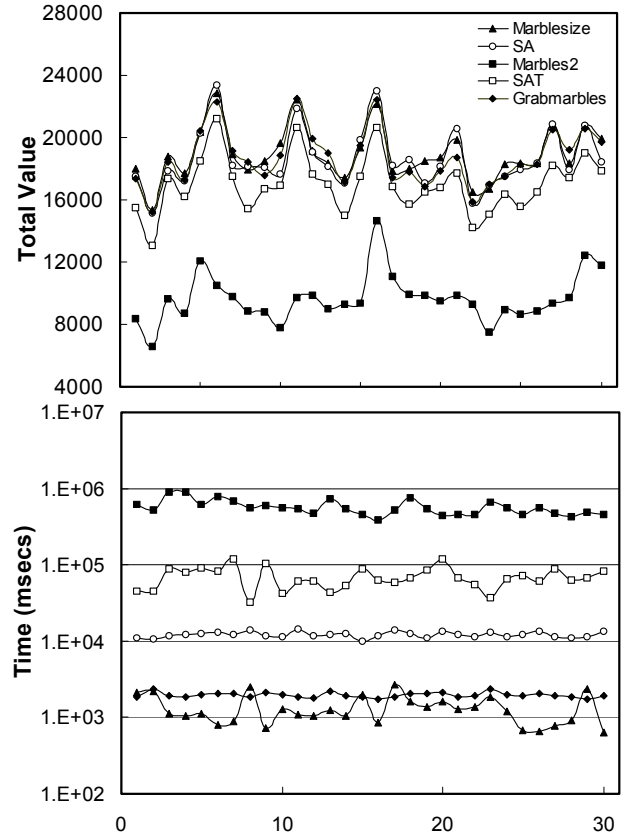


Figure 2: Performance evaluation of solvers on a resource allocation problem with 100 resources and 100 tasks

total value but with Marblesize being 3 to 4 times faster than Grabmarbles and about an order of magnitude faster than Simulated Annealing. The results obtained with SAT and Marbles2 are of lesser quality in terms of performance but we see that they follow the same structure found in the other curves suggesting that all curves are somehow converging towards an optimal solution.

In Figure 3 we study the behavior of the total number of messages, execution time and total value of solutions found with Marblesize for different size of the problem. In Figure 3a, shows results for 100 tasks and different number of resources while in Figure 3b the results correspond to 100 resources and different number of tasks. In both curves we observe an easy-hard-easy phase-transition effect where the number of messages (and time) increases very drastically as the problem gets larger until it reaches a peak and after that drops down again. This property of Marblesize is due to the fact that unlikely to succeed tasks drop out of the competition very early in the process and do not waste any bidding messages. Since the distributions of task values and number of requirements per tasks are independent, tasks with large number of resources and low task value end up with marbles of relatively small size that makes them lose in all bids before entering competition. In Figure 3b, the execution time continues to rise slowly after the transition peak while the number of messages drops down and this is

due to the fact that in the initial phase tasks need to evaluate alternative combination of resources before bidding and the total computational time of this operation increases with the number of tasks.

Related Work

What we are after are distributed negotiation schemes in which (1) domain experts can understand the decisions made by negotiation participants because they use a domain currency for making their trade-offs that the experts share, and that (2) can be “steered” in its collective real-time response, fault tolerance, and solution quality behavior by changing their relative desirability at run-time.

coordinate between themselves (as they cannot individually auction themselves but must bundle up with others to be bid on in combination). We cannot say with certainty that there may not be a space for them in real-time adaptive distributed resource allocation but we are not currently exploring this route.

In (c) 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 (solving an NP-complete problem) and report the assignments back to the requesters. This is obviously not an option for truly distributed resource allocation either, and we are not investigating this avenue further either.

This leaves (a) single-resource auctions, in which each

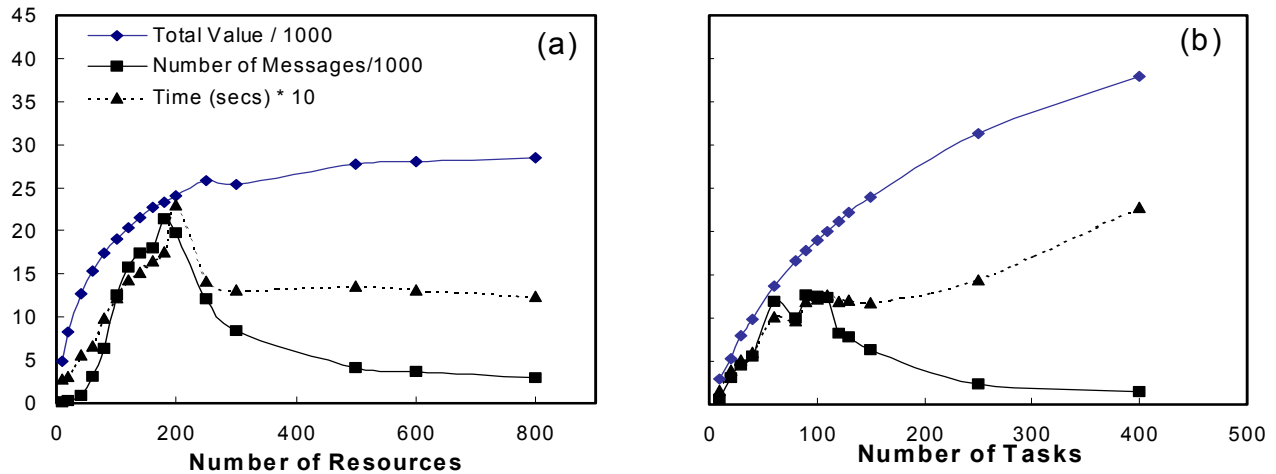


Figure 3: Easy-hard-easy phase-transition behavior of the total number of messages and computational time for the Marblesize scheme. (a) 100 tasks, (b) 100 resources

We list the most relevant non-market-inspired previous work on distributed resource allocation in the References section, but do not have the space here to discuss them at any length (they generally address neither (1) nor (2) above). Instead, we will use the remaining space to put the auction protocols we use for resource acquisition in the context of prior work.

A negotiation protocol in our terms defines the types of messages that can be sent and how they can be strung together (the syntax of message exchange). In our terminology, this – together with the bidding strategies of requesters and the auctioning strategy of requesters – defines a “scheme” for market-based distributed resource allocation.

Walsh et al. (1998) outline the fundamental choices in this design space: (a) single-resource auctions, (b) combinatorial auctions, and (c) Vickery auctions.

We view (b) combinatorial auctions as generally inapplicable to truly distributed assignment of resources. This is because they need a large number of messages to

resource can auction itself off to the highest-value task based solely on its local bid information. Our Marbles schemes are a subclass of single-resource auction. However, the distributed algorithms introduced for the *altruistic task suicide* phase distinguish our Marbles schemes from work on competitive auctions. This task suicide phase is fundamental for the quick convergence of the Marbles schemes: by lowering resource prices it usually helps other tasks succeed.

Conclusion

It is obviously far too early for us to make any claims on how far from “optimal” in any sense our currently implemented Marbles schemes are (be that in term of the quality of the solution, in terms of the number of messages needed, or any combination thereof). However we can conclude the following:

1. Marbles-type distributed collaborative negotiation schemes are an exciting and worthwhile research program for years to come; this is because there are many “optimal” solvers depending on how much the application domain values fault-tolerance, average response time, real-time response guarantee, and quality of the solution.
2. We are seeing “phase transitions” in our problems as is evidenced in Figure 3; to be precise, we are seeing Gauss-like curves for the amount of messages needed based on a varying number of resources for a fixed number of tasks. A Marbles scheme finds out quickly that few tasks can be satisfied with the very few resources, as well as that nearly all tasks can be satisfied with the abundant resources, but uses substantially more computation if there are “just enough resources for most of the tasks with the right assignments”. However, we currently have no way of predicting how much negotiation a given problem requires (but are working on that in our ATTEND project).
3. It seems that the Marbles schemes with good performance all seem to have the property of eliminating (apparently) losing tasks very early on.
4. As this is work in progress we have not compared our schemes against other distributed algorithms at great length. However, based on our performance comparisons of our best Marbles schemes to the well-known centralized Simulating Annealing strategy we believe that this family of market-inspired collaborative negotiation schemes is well-suited to the real-time distributed solution of resource allocation problems.

Acknowledgments

We gratefully acknowledge funding by DARPA ITO ANTS program under program manager Dr. Janos Sztipanovits (Contracts No. F30602-00-2-0533 and F30602-99-1-0524). Part of this work was also funded by AFOSR under grant number F49620-01-1-0341. Alejandro Bugacov thanks Bart Selman and Carla Gomes for assistance on the SAT encoding.

References

Andersson, M., and Sandholm, T. 1999. Time-Quality tradeoffs in reallocation negotiation with combinatorial contract types. In *Proceedings of AAI-99*, Orlando, Florida.

Atkins, E.; T. Abdelzaher; Shin, K.; and E. Durfee, E. 1999. Planning and resource allocation for hard real-time. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, Washington.

Boutilier, C.; Goldzmidt, M.; and Sabata, B. 2000. Sequential auctions for the allocation of resources with complementarities. In *Proceedings of IJCAI-99*, Stockholm, Sweden.

Choy, M., and Singh, A. 1992. Efficient fault tolerant algorithms in distributed systems. In *24th ACM Symposium on Theory of Computing*, pp. 593–602.

Collins, J.; Sundareswara, R.; Tsvetovat, M.; Gini, M.; and Mobasher, B. 1999. Search strategies for bid selection in multiagent contracting. In *JCAI-99 Workshop on Agent-mediated Electronic Commerce (AmEC-99)*.

Ferguson, D.; Nikolaou, C.; Sairamesh, J.; and Yemini, Y. 1996. Economic models for allocating resources in computer systems. In S. Clearwater (Ed.), *Market-Based Control: A Paradigm for Distributed Resource Allocation*. Hong Kong: World Scientific.

Garey, M., and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: W.H. Freeman and Company.

Gent, I., and Walsh, T. 1993. Towards an Understanding of Hill-Climbing Procedures for SAT. In *Proceedings of AAAI-93*, pp. 28-33.

Jackson. We used Dr. D. Jackson’s Java implementation of WSAT available at <http://sdg.lcs.mit.edu/walksat>

Kirkpatrick, S.; Gelatt, C.; and Vecchi, M. 1983. Optimization by Simulated Annealing. *Science*, 220, 671-680.

Milgrom, P. 2000. Putting auction theory to work: The simultaneous ascending auction. *Journal of Political Economy*.

Sandholm, T., and Lesser, V. 1997. Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. *Readings in Agents*, pp. 66-73, Morgan Kaufmann.

Selman, B.; Kautz, H.; and Cohen, B. 1996. Local search strategies for satisfiability testing. *AAAI-92, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, 26:521-532.

Smith, R. 1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers* 29(12):1104-1113.

Waldspurger, C., and Wehl, W. 1994. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pp. 1–11.

Walsh, W.; Wellman, M.; Wurman, P.; and MacKie-Mason, J. 1998. Auction protocols for decentralized scheduling. In *Eighteenth International Conference on Distributed Computing Systems*, Amsterdam, The Netherlands.

Walsh, W.; Wellman, M. 1998. Market SAT: An extremely decentralized (but really slow) algorithm for propositional satisfiability. In *Seventh National Conference in Artificial Intelligence*, 303-309, 2000.

Wellman, M. 1996. The economic approach to artificial intelligence. *ACM Computing Surveys* 28 (4es):14–15.