Fast Winner Determination for Agent Coordination with SBB Auctions[*]

(Extended Abstract)

Kenny Daniel
Computer Science Department
University of Southern California
941 W. 37th Place
Los Angeles, CA 90089-0781, USA
kfdaniel@usc.edu

Sven Koenig
Computer Science Department
University of Southern California
941 W. 37th Place
Los Angeles, CA 90089-0781, USA
skoenig@usc.edu

ABSTRACT

The runtime of winner determination for each round of a sequential bundle-bid auction (= SBB auction) has recently been shown to be linear in the number of submitted bids, which makes SBB auctions appealing for solving cooperative task-assignment problems. In this paper, we introduce the Shrewd (= SHrewd Resource Efficient Winner Determination) algorithm, whose runtime is linear in the number of submitted bids but typically much smaller than the runtime of the existing winner-determination algorithm for SBB auctions, making them feasible for larger bundle sizes.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

auctions, combinatorial auction, coordination, decentralized control, distributed problem solving, sequential bundle-bid single-sale auction, task allocation, winner determination

1. INTRODUCTION

Auctions promise to solve cooperative task-assignment problems with small communication and computation costs [1]. Sequential bundle-bid auctions (= SBB auctions) with bundle size k are multi-round auctions that assign k tasks in each round to one or more agents so that the team cost increases the least [2]. The runtime of winner determination for each round of an SBB auction has recently been shown to be linear in the number of submitted bids, making SBB auctions appealing [2]. In this paper, we introduce Shrewd (= SHrewd Resource Efficient Winner Determination), whose runtime is linear in the number of submitted bids but typically much smaller than the one of the existing winner-determination algorithm for SBB auctions since it exploits the structure of the optimization problem better. Our results make SBB auctions feasible for larger bundle sizes, thus advancing the state of the art in solving SBB auctions.

[*]This research was partly supported by NSF awards under contracts ITR/AP0113881, IIS-0098807 and IIS-0350584 as well as seed funding from NASA's Jet Propulsion Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

Cite as: Fast Winner Determination for Agent Coordination with SBB Auctions (Short Paper), Kenny Daniel and Sven Koenig, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

2. PROBLEM STATEMENT

We use multi-agent routing problems as examples of task-assignment problems [3]. The agent cost to visit a set of given targets corresponds to the smallest travel distance that the agent needs to visit the targets from its current location. The team cost is the sum of the agent costs. The objective of a multi-agent routing problem is to determine which agent should visit which targets in which order so that each agent visits at most a given number of targets (= its capacity), each target is visited by an agent, and the team cost is small. In this paper, we use SBB auctions to achieve this objective under the assumption that the terrain, the locations of all agents and the locations of all targets are known.

An SBB auction with bundle size $k \geq 1$ assigns k targets to one or more agents in each round. Each agent submits a constant number of bids per round that does not depend on the number of agents or tasks. Each bid $b = (b._{agent}, b._{tasks}, b._{cost})$ is a triple that consists of an agent, a set of at most k tasks and a bid cost. Let B be the set of bids submitted by all agents in a given round. Two bids $b, b' \in B$ are *compatible* iff $b._{agent} \neq b'._{agent}$ and $b._{tasks} \cap b'._{tasks} = \emptyset$. A *portfolio* is a set of bids. The team cost of a portfolio $B' \subseteq B$ is $c^{team}(B') := \sum_{b \in B'} b._{cost}$. A portfolio $B' \subseteq B$ is *potentially winning* iff its bids are pairwise compatible and $| \cup_{b \in B'} b._{tasks}| = k$. A portfolio $B' \subseteq B$ is *winning* iff it is potentially winning and its team cost is no larger than the team cost of all potentially winning portfolios. The objective of winner determination is to determine a winning portfolio. The runtime of the only existing winner-determination algorithm is linear in the number of submitted bids [4]. However, it is too slow for real-time task allocation. In the following, we thus develop a novel winner-determination algorithm.

3. SHREWD

Our novel Shrewd (= SHrewd Resource Efficient Winner Determination) proceeds as follows: For every non-increasing sequence $[s(1) \ldots s(l)]$ of positive integers that sum to k, it constructs a primary search tree. Each non-root node in the primary search tree is labeled with a bid that must be pairwise compatible with the bids of all of its non-root ancestors. The portfolio of a node is the bid of the node together with the bids of its non-root ancestors. The portfolio of a node at depth l is thus a potentially winning portfolio. Shrewd constructs the primary search tree starting with a dummy root node as follows: Shrewd repeatedly picks a leaf node of the primary search tree at depth less than l, constructs a secondary search tree for that leaf node and then adds all of the nodes in that secondary search tree as the successors nodes of the leaf node in the primary search tree.

Assume that the depth of the leaf node of the primary search tree is i and its portfolio is B'. Each edge in the secondary search tree is labeled with a constraint. Each node in the secondary search tree is labeled with a bid, namely the bid with the smallest bid cost that is pairwise compatible with the bids in B', is on a set of $s(i+1)$ tasks and satisfies all constraints that label the edges from the root node of the secondary search tree to the node in question. Shrewd deletes the node from the secondary search tree if no such bid exists. Shrewd constructs the secondary search tree as follows: Shrewd repeatedly picks a leaf node of the secondary search tree and then adds one successor node for each one of the following constraints, where b is the bid that labels the leaf node and b' is a variable that represents the bid that labels its successor node:

- Type A: "$t \notin b'$*.tasks*," one constraint for each $t \in b$*.tasks*; and
- Type B: "b'*.agent* $\neq b$*.agent*."

Shrewd deletes a successor node from the secondary search tree if the number of constraints of Type A that label the edges from the root node to the successor node is larger than $\sum_{j=i+2}^{l} s(j)$ or the number of constraints of Type B that label the edges from the root node to the successor node is larger than $l - i - 1$.

Once Shrewd has constructed all primary search trees, it returns the portfolio of the leaf node of any primary search tree with the smallest team cost as winning portfolio. We can prove that Shrewd finds the winning portfolio and runs in time linear in the number of bids.

4. OPTIMIZED SHREWD

The secondary search trees of Shrewd can be optimized in case $s(i+1) = 1$. Everything remains unchanged, except that, in this case, Optimized Shrewd repeatedly picks a leaf node of the secondary search tree and then adds one successor node for each one of the following constraints, where b is the bid that labels the leaf node and b' is a variable that represents the bid that labels its successor node:

- Type C: "b*.agent* $= b'$*.agent* and b*.tasks* $\neq b'$*.tasks*."

Shrewd deletes a successor node from the secondary search tree if the number of constraints of Type C is larger than $l - i - 1$. We can prove that optimized Shrewd finds the winning portfolio and runs in time linear in the number of bids.

5. EXPERIMENTAL RESULTS

Table 1 shows upper bounds on the number of nodes in the primary search trees for the existing winner-determination algorithm, Shrewd and Optimized Shrewd. (We call them upper bounds because additional pruning of nodes might be possible.) Optimized Shrewd uses fewer search tree nodes than Shrewd, and Shrewd uses fewer search tree nodes than the existing winner-determination algorithm.

Bundle Size	Existing	Shrewd	Opt Shrewd
$k=1$	1	1	1
$k=2$	17	11	5
$k=3$	1,482	227	32
$k=4$	244,809	15,166	410
$k=5$	$9.8 \cdot 10^7$	$3.7 \cdot 10^6$	10,122
$k=6$	$7.4 \cdot 10^{10}$	$3.4 \cdot 10^{10}$	587,893
$k=7$	$9.8 \cdot 10^{13}$	$1.2 \cdot 10^{13}$	$7.9 \cdot 10^7$

Table 1: Upper Bounds on the Number of Search Tree Nodes

Bundle Size	Team Costs	Run Times		
		Existing	Shrewd	Opt Shrewd
$k=1$	500.77	0.0028	0.0028	0.0024
$k=2$	515.81	0.0265	0.0251	0.0245
$k=3$	491.89	1.4335	0.2030	0.1990
$k=4$	483.09	–	2.2822	1.7762
$k=5$	481.12	–	208.8787	10.5177
$k=6$	480.73	–	–	487.6028

Table 2: Team Costs and Runtimes

To demonstrate that fewer search tree nodes translate into smaller runtimes, we generated 50 random instances of a multi-agent routing problem with 40 targets and 10 agents of capacity 4 randomly placed on a 51×51 eight-neighbor grid that modeled a building, similar to [4]. Table 2 shows the average team costs and runtimes (in seconds) of a Java 1.6 implementation of an SBB auction on an AMD Athlon 64 X2 Dual Core 3800+ PC with 4GB of RAM running Ubuntu 8.04. The runtimes are for the complete SBB auction to assign all targets to agents, including bidding and winner determination for all rounds. Each agent uses a combination of the two-opt and cheapest-insertion heuristics to approximately solve the TSPs necessary for computing its agent costs. Optimized Shrewd ran faster Shrewd, and Shrewd ran faster than the existing winner-determination algorithm. Optimized Shrewd could solve SBB auctions with bundle size $k = 6$ while the existing winner-determination algorithm could only solve SBB auctions with bundle size $k = 3$. This is a big improvement because the runtime of all of these winner-determination algorithms is exponential in the bundle size. Furthermore, the increase in bundle size made possible by Shrewd decreased the team cost substantially for our multi-agent routing problem. Further increases in bundle size then no longer reduced the team cost significantly. Thus, SBB auctions with bundle size $k = 5$ present a good tradeoff between small runtimes and small team costs.

6. REFERENCES

[1] M. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94:1257–1270, 2006.

[2] S. Koenig, C. Tovey, X. Zheng, and I. Sungur. Sequential bundle-bid single-sale auction algorithms for decentralized control. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1359–1365, 2007.

[3] M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, S. Koenig, A. Kleywegt, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Proceedings of the International Conference on Robotics: Science and Systems*, 2005.

[4] C. Tovey, M. Lagoudakis, S. Jain, and S. Koenig. The generation of bidding rules for auction-based robot coordination. In L. Parker, F. Schneider, and A. Schultz, editors, *Multi-Robot Systems: From Swarms to Intelligent Automata*, pages 3–14. Springer, 2005.