A Tractable Heuristic that Maximizes Global Utility through Local Plan Combination

Eithan Ephrati
Computer Science Department
University of Pittsburgh
tantush@cs.pitt.edu

Martha E. Pollack
Computer Science Department
and Intelligent Systems Program
University of Pittsburgh
pollack@cs.pitt.edu

Jeffrey S. Rosenschein
Institute of Computer Science
The Hebrew University
jeff@cs.huji.ac.il

Abstract

We consider techniques suitable for combining individual agent plans into a global system plan, maintaining a commitment to considerations of global utility that may differ radically from individual agent utilities. We present a three-stage heuristic reduction process, consisting of a transformation from local to global utility measures, a global assessment of the local evaluations of agents, and approximation algorithms to maximize resource usage over time. We also consider how these techniques can be used with self-motivated agents, and show how the overall process can be distributed among a group of agents.

Introduction

Distributed Artificial Intelligence (DAI) has traditionally considered the global utility of a system as strongly connected to the local utilities of the agents that comprise that system. So, for example, the sum (or product) of agent utilities might be maximized by the designers of an interaction environment (Rosenschein & Zlotkin 1994), even when the agents themselves are self-motivated; global utility is often assumed to be related to pareto optimality. In Distributed Problem Solving systems, these connections between local and global optimality are also widespread (Durfee, Lesser, & Corkill 1987; Smith 1980; Sandholm 1993).

This paper, on the other hand, considers systems where the relationship between individual and global utility functions is not straightforward. We analyze situations where individual agents produce locally optimal plans (exploiting the parallelism and relative simplicity of smaller search spaces), and then the system coordinator wishes to derive a globally optimal plan (with arbitrary global criteria) by appropriately combining these local plans.

The synthesis of a global plan out of locally developed ones has been examined in the DAI and Planning literatures (Ephrati & Rosenschein 1991; 1993; 1994; Wellman 1993; Kraus 1993). Usually, the problem is addressed by implicit or explicit negotiation mechanisms. These negotiation mechanisms, however, presuppose that the global utility function more or less directly reflects the local concerns of agents. In our current research, the global considerations (resources, constraints) may not be mirrored by the local functions that created the constituent plans.

Our overall approach is to employ several heuristic reduction steps that simplify the problem; this is necessary since forming an optimal solution is infeasible (i.e., takes too long) in dynamic environments. The approach is closely related to that taken by Etzioni (Etzioni 1991) for single-agent, multiple-goal situations. The heuristic reduction steps that we employ borrow from ideas developed in economics, operating systems, and complexity theory.

An Example

In this section, we describe an example of an application in which there is a natural separation between the global utility function, which should be maximized for overall system performance, and the individual utility functions that each agent should try to maximize in forming a plan to achieve its tasks. The example derives from the "Pacifica NEO Scenario," which was developed by Reece and Tate for the RL/ARPA Planning Initiative as part of the PRECiS environment (Reece *et al.* 1993). This scenario involves the fictional island nation of Pacifica, which has four cities, four roads, and a bridge, as well as certain other natural features, such as a volcano. When the volcano erupts, it can make the nearby roads impassable. Additionally, there are periodic monsoons, which can also close roads and/or the bridge. Groups of American citizens are located at or near each city; the overall mission objective is to move all these people by truck to the capital city of Delta, and from there evacuate them by airplane to Honolulu. We have implemented this scenario as the simulated dynamic environment of DIPART—the Distributed, Interactive Planner's Assistant for Real-time Transportation planning—a prototype simulation system that includes a network of agents, each of which is intended to assist a human planner (Pollack *et al.* 1994).

In a particular DIPART scenario, multiple human planners are each tasked with some subset of the overall mission objectives of evacuating the American citizens from Pacifica. In particular, we typically have one planner with primary responsibility for the evacuation of each city, although this does not preclude interactions among the planners. Each human planner performs his or her operation with the assistance of a planning node of the DIPART system. The planning nodes are responsible for forming plans to satisfy the user's goals, for coordinating communication with other planners, and for alerting the user to reports from agents in the (simulated) world.

The individual planning agents may each have a different perspective on the current state of the environment. Indeed, security concerns may preclude them from having complete knowledge of the mission plans. Thus, there may well be a separation between the utility functions that the individual agents are employing in generating plans, and the overall utility function that should be maximized in conjoining the individual plans into a global plan.

Assumptions

Throughout this paper, we make the following assumptions:

- All the agents in the system work towards the achievement of a global goal G, with a (global) deadline $\hat{T}(G)$. In the DIPART system, the global goal is to evacuate all the U.S. citizens from Pacifica; the global deadline is assumed to be fixed by an agent outside of the system.
- The individual agents periodically initiate or are assigned tasks, the completion of which contributes to the overall goal. These tasks are thus subgoals. We identify the set of agent i's tasks as time t as $\{g_i^t\}$. Each individual task will have a local deadline: $\hat{t}(g_{i_j}^t)$. In the DIPART system, an individual task may consist of evacuating 60 people from the city of Abyss within 22 time units. If this is the first task assigned to agent 3 at time 44, then $\hat{t}(g_{3_1}^{44}) = 66$. For notational simplicity, we assume, when we can do so without loss of generality, that each agent has only one assigned task at a time, and we then identify i's task at t simply as g_i^t.
- Each agent will form a complete plan to achieve each of its tasks. In this paper, we ignore the process of plan generation, and instead concentrate on finding a tractable way to coordinate the execution of generated plans. $P(g_i^t)$ denotes i's plan to achieve g_i^t. (When needed, we generalize the notation in the obvious way to handle the case in which the agent has multiple simultaneous tasks.) We assume that each plan has an associated cost $(c(P(g_i^t)))$, which can be computed by the agent generating it. The cost is typically a function of the resources that the plan will consume. Each task has an associated reward $r(g_i^t)$, which the agent is told at the time it is assigned the task, or which the agent determines according to its local view. Thus, it can estimate the value or profit of successfully completing the plan, $v(P(g_i^t))$. Note that we assume constant rewards (i.e., step utility functions for plans), which is the most general case, since heuristic methods can better deal with partial satisfaction of plans (see the Section on Approximation Algorithms below). Similarly, we assume that executing plans cannot be preempted.

We are concerned with situations in which agents have partial, possibly inconsistent views of the world (e.g., they are located at remote distances from one another, and the environment changes dynamically). Since communication takes time and is costly, agents separately generate plans to achieve their local tasks; we assume that sufficient communication to achieve coordination is infeasible or inadvisable. The problem, of course, is that the resulting local plans may be incompatible, due to incomplete information about both other agents' activity and the environment. The goal of the system is thus to coordinate the execution of these plans in a way that will eventually maximize the overall utility of the system, although not necessarily the utility functions of individual agents. Both the global utility function and the individual utility functions of the agents may be arbitrarily dependent on resources consumed, time used, and the reward associated with the plan.

Note that this general statement covers many Distributed Problem Solving (DPS) scenarios; in most such scenarios, however, some of the constraints are relaxed (e.g., there are no local deadlines, the set of local plans is known ahead of time, etc.). Our general heuristic method remains suitable for all such relaxed scenarios.

The Heuristic Reduction Steps

Essentially, the problem we address is to find the best feasible execution combination (schedule) of the local plans within the global deadline. A schedule is a mapping from time points to sets of plans proposed by the agents for execution at that time. Since plans may be incompatible with one another, not all schedules are feasible: typically only a proper subset of the plans proposed for execution at any given time can be performed. Moreover, not all feasible schedules are equally good: some contribute to the global utility function more than others. To determine an optimal schedule, a system would have to generate all alternative schedules, identify the feasible ones, and select from those the schedule with the best outcome. In an idealized setting, in which the scheduling system had complete knowledge of all the plans that would be generated ahead of time, these computations could be performed off-line.

However, we do not assume that the system has global knowledge of the plans that will be generated (nor even, ahead of time, of the tasks that will be assigned to the agents). The global activity is determined by the set of individual plans, which are generated in parallel, according to the agents' local views. In other words, in our model the agents receive tasks, and generate plans for those tasks; some subset of the generated plans are selected for execution; execution begins; and then the cycle repeats. Instead of trying to coordinate activity among agents ahead of time, coordination must therefore take place on-line, during the plan generation and execution process. This calls for the first heuristic reduction step.

Step 1: From Local Utility to Global Utility

The overall goal of the system is to find a schedule in which the global utility is maximized. However, because the system at each point in time only has access to the plans that have already been generated, and the local utilities that the generating agents ascribe to them—it is not prescient—the best it can do is to identify the current combination of plans that maximizes utility. This process must be iterated over time: the system must repeatedly choose a plan combination with maximal aggregated utility.

It might appear that the problem is thus, at each time point, to find a compatible set of proposed plans that maximizes the aggregated value. But this is still insufficient, because the execution time of each plan has a significant influence on the overall performance of the system. For example, a shorter plan of less value, executed now, may allow later execution of a higher valued plan.

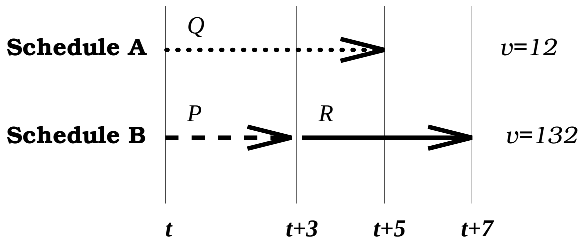

Figure 1: Two Alternative Execution Combinations

Figure 1 illustrates a decision that needs to be made at time t between the execution of two plans P and Q, where the declared value of P is 11 and its execution time and deadline are both 3 units; the declared value of Q is 12 and its execution time and deadline are 5 units. Assume further that plan Q is selected for execution, because of its higher value, and that then, at time $t+1$, a new plan R is proposed, with declared value 121 and execution time 4 units. R, however, also has a deadline of time $t+8$, and is incompatible with both P and Q. Then, as illustrated by Schedule A in Figure 1, R cannot be scheduled, because its deadline cannot be met (it cannot be started until after Q is complete, at time $t+5$, and then there is not sufficient time left). On the other hand, if P had been chosen (Schedule B), R would have been executable, resulting in a total reward to the system of 132 compared to 12.

The problem of on-line scheduling has been addressed in the real-time operating systems literature. A solution proposed there involves the use of *value density*, which is a function both of value and of expected execution time (Jensen, Locke, & Tokuda 1985; Locke 1986). The essential idea is to maximize the *capacity* of the system at all times (where capacity stands for value per time unit). Note that if at every instant the system operates in its maximal capacity, the optimal solution will be achieved. Thus the objective of the system at a given time step is to maximize its future capacity.

Formally, let t_i and v_i denote the execution time and value function of the plan p_i, respectively. Then the value density d_i is defined to be v_i/t_i.[1] In economic terms, this is exactly the *marginal utility* (with respect to time) of p_i. In economics, marginal utility serves as the basic measure of the efficiency of aggregated productivity. Marginal utility has been used in a similar manner within the field of AI by Etzioni (Etzioni 1991) to heuristically guide a single agent in choosing among alternative methods to achieve its goals. Within the DAI literature, marginal cost was used by Sandholm (Sandholm 1993) to measure the quality of bids within the context of the Contract Net.

If only one job (or plan) can be executed at a time, then value-density scheduling is optimal:

Theorem 1 (Due to W. E. Smith 1956) *If a set of processes, each with a known completion time, is processed in decreasing order of value-time density the resulting total value at every process completion time is optimal.*

In our case, several plans may be executed in parallel. We therefore can only approximate the optimal schedule by trying to execute feasible *combinations* of plans, in decreasing value-density. Thus, at each decision point our system should allow the parallel execution of the plan combination that scores the highest aggregated value-density.

Note also that the declared value of each plan is based on the local view of some agent. One motivation for this work is to handle cases in which there is a disparity between the individual and global utility functions. Thus, the system should learn how to "translate" individual value estimates into its own global utility units.

[1] Recall that we are assuming that all value functions are step-functions; $v - i$ is the constant, non-zero value of task i up until its deadline.

Step 2: Global Assessment of Local Evaluation

Our next approximation step is therefore to shift from declared value-density to *expected value-density* of each plan. To compute expected value, the system normalizes the declared value by an amount determined by previous interactions with the declaring agent. An estimate by an agent that tends to overestimate will be corrected by a weight < 1; similarly, underestimating agents will have their estimates increased.

More formally, if d_i is the value density of some plan (computed using agent i's declared value and execution time estimate), then we define the expected density to be $z_i^t d_i$, where z_i^t is the normalization factor at time t. To derive z_i^t, we maintain a record of i's evaluation and actual performance. Let v_i' and t_i' denote respectively the actual contribution and the actual time consumed by p_i. Then the prediction ratio, z_i', of i's bid is $\frac{v_i'}{t_i'}/\frac{v_i}{t_i}$ (thus if i underestimates its plan, $z_i^t > 1$, and if it overestimated its plan, $z_i^t < 1$).

There are several alternatives to dynamically update i's prediction. The simplest and statistically most straightforward one involves taking this ratio to be the mean of performance for some history (starting at some prior t_p):

$$z_i^{t+1} = \frac{\sum_{t_p}^{t} z_i^t}{t - t_p}.$$

Since the agent's estimate of the value density is made of two estimated values, this sampling could be refined as to more accurately deal with these separate values: $z_i^{t+1} = (\sum_t v_i' / \sum_t t_i')/(\sum_t v_i / \sum_t t_i)$ for some history T.

Agents may improve or degrade over time, and hence the history horizon should not be too long. Alternatively, it would be possible to have a weighted mean such that later samples have higher weight. Many alternative heuristics could also be used to update the agent's prediction ratio, for example, setting $z_i^{t+1} = z_i^t z_i'$ would yield a more conservative update. Yet another alternative would be to assume a specific error distribution over the agent's assessments, and to use reinforcement learning strategies to compute the normalization factor (Sandip, Sekaran, & Hale 1994). Such strategies may be most effective when the agents are assumed to be self-motivated.

Obviously, the main drawback of the sampling methods is they may only approximate the actual adjustment required. Therefore, the sampling strategy should be as specific as possible. For example the weighting function may be tailored to specific environments of intended use. The more specific the function is, the more accurate the effect of the weighting.

One problem with this scheme is that an agent who always underestimates the value of its plans may be subject to "starvation". Initially, the system will have no history about this agent, and so will not adjust its estimates. If it continues to underestimate, its plans may never be selected for execution, and hence the system will never learn that the agent is a consistent underestimator. One solution to this problem is to increase the normalization factor of "losing" agents in direct relation to their consecutive losses.

Step 3: Approximation Algorithms

As described so far, what the system must do is solicit declared values for each plan from the participating agents, and then normalize the declarations. The next step is to select for execution a consistent set of plans that maximizes normalized value-density. To be consistent, a set of plans must not collectively have resource requirements that exceed the total resources available.[2] Let r_i denote the resources used by p_i, and $\mathcal{R}^t$ denote the total system resources available at time t. The problem can then be stated as follows:
Find $max \sum_i^n d_i x_i$ subject to $\sum_1^n r_i x_i \leq R^t$, such that $x_i \in \{0,1\}$, $1 \leq i \leq n$.

Unfortunately, this is exactly the **0/1-Knapsack** problem, which is known to be NP-hard.[3] Therefore, a further approximation method is needed. Fortunately, there are many heuristic algorithms for handling the 0/1-knapsack problem (Horowitz & Sahni 1978). Figure 2 outlines one such algorithm (a so-called ϵ-approximation algorithm (Horowitz & Sahni 1978, p. 580)), described using the terminology developed so far in this paper. As input, the algorithm accepts $D = \{d_i\}$—the corrected value-densities of the n plans, and $R = \{r_i\}$—their resource usages. The parameter k determines the precision of the outcome.

Approx$(D, R, \mathcal{R}^t, n, k)$

1. $D_{max} = 0$; Sort R and D such that $\frac{d_i}{r_i} \geq \frac{d_{i+1}}{r_{i+1}}, 1 \leq i < n$.
2. For each feasible execution combinations F of size $\leq k$ do:

(a) $D_F = \sum_{i \in F} d_i$
(b) $D_{max} = \max(D_{max}, D_F +$ GreedyValueDensityCompletion$(F))$

Figure 2: Approximation Algorithm for the 0/1-Knapsack Problem

[2] For example, in the DIPART scenario, if there are a fixed number n of trucks available for transporting citizens, the tasks selecting for execution cannot collectively require more than n trucks.

[3] Surprisingly, if instead of a rigid deadline we had assumed that each plan has an associated utility function that returns a value for partial completion of the plan (in the case of preemption), then a greedy solution will be optimal, and there is no need to solve a Knapsack Problem (Boddy & Dean 1994).

Note that the algorithm once again uses a value-density approach. However, it makes its selections using d_i divided by r_i, i.e., value-density divided by estimated resource use. The plans are initially sorted in decreasing order of value-time-resources density (Line 1). The main processing is done in the loop of Line 2, in which all feasible combinations of size $\leq k$ are considered. (A plan that cannot meet its local deadline or the global one may not be a part of a feasible combination.) Then each such combination is completed (to the maximal resource usage) by repeatedly adding plans according to their value-resource-time density. This is done using the greedy subroutine "GreedyValueDensityCompletion" which has as a parameter the set of plans that have already been considered and therefore may not be considered again. For example, for $k = 1$ the algorithm would compare n combinations, where the i-th combination consists of p_i and all the plans that can be executed in parallel with it, selected (iteratively) in decreasing value-time-resources density.

The advantage of the algorithm is that its deviation from the optimal solution may be as small as desired. Let D^* be the global (aggregated) value-density resulting from the actual optimal combination, and let D be the value-density returned by the algorithm, then $\frac{|D^*-D|}{D} < \frac{1}{k+1}$. This bounded deviation is attained with a complexity of $O(n^{k+1})$, and on the average the deviation from the optimal solution is much smaller (Horowitz & Sahni 1978).

It is up to the system designer to choose the desired k value. However, since the planning problem itself is NP-Complete, in most circumstances the planning phase will be much more time-consuming than the scheduling step, so a higher k may be justified.

Self-interested Rational Agents

We next consider how to extend our approach to situations involving self-interested rational agents. Such agents need to be motivated to contribute to the group welfare. One way to achieve this is to provide the agents with some kind of incentive or reward (Kraus 1993; Zlotkin & Rosenschein 1993; Ephrati, Perry, & Rosenschein 1994; Wellman 1992). A straightforward policy would pay each agent the cost associated with its performed work plus a minimal additional compensation.

Unfortunately, given such a policy, the agents may be tempted to overstate the value of their plans in order to ensure that they will be selected for execution and thus the agents will be paid. Therefore, the reward should *both* compensate the agent with the true execution cost of its plan, and motivate it to declare what it believes to be the *true* global contribution of a plan *according to the system's objective.*

Fortunately, there exists a straightforward policy of this kind: let d_i be the declared value density of p_i and let d_i^* be the actual contribution that resulted from the execution of the plan. Then the reward policy, $R(p_i, d_i)$ should be:

$$R(p_i, d_i) = \begin{cases} c(p_i) + f(d_i^*) & \text{if } d_i = d_i^* \\ c(p_i) & \text{else} \end{cases}$$

where $f(d_i^*)$ is some arbitrary positive function of d_i^*. This policy guarantees that each agent will try to estimate correctly its contribution. (Recall that $c(p_i)$ defines the cost of executing plan p_i.) Moreover, as the following theorem shows, this reward policy can be tuned to motivate agents to come up with higher-quality plans:

Theorem 2 (a) *Given the above reward policy, it is the dominant strategy of each agent to submit a global estimate as accurate as possible.* (b) *If $f(d_i^*)$ strictly increases with d_i^*, a rational agent will be motivated to generate plans that achieve more in a shorter time (if desired, f could further be refined in a straightforward manner to consider resources as well).*

Proof.

Let $B(p_i)$ denote i's resulting benefit from the execution of p_i. Clearly, $B(p_i)$ equals the reward that i receives, $R(d_i)$, minus its investment (the execution cost $c(p_i)$).

(a) To show that each agent's dominant strategy is to declare what it believes to be the true value-density of each alternative plan that it has generated, no matter what the others do, we have to show that i's benefit from declaring the correct value d_i^*, of some p_i is greater than its benefit from any other declaration d_i' (for that specific p_i). Since $c(p_i)$ is identical in both cases, it is sufficient to show that $R(d_i^*) - R(d_i') > 0$ for any $d_i' \neq d_i^*$. And indeed, by definition: $R(d_i^*) - R(d_i') = c(p_i) + f(d_i^*) - (c(p_i)) = f(d_i^*) > 0$.

(b) Let $\hat{d}_i$ and d_i' be two accurate assessments (of $\hat{p}_i$ and p_i', respectively), such that $\hat{d}_i > d_i'$. We have to show that $B(\hat{p}_i) > B(p_i')$. And indeed, by definition:
$B(\hat{p}_i) - B(p_i') = R(\hat{d}_i) - c(\hat{p}_i) - [R(d_i') - c(p_i)] =$
$c(\hat{p}_i) + f(\hat{d}_i) - c(\hat{p}_i) - [c(p_i') + f(d_i') - c(p_i')] =$
$f(\hat{d}_i) - f(d_i') > 0$ (f is strictly increasing with its argument). □

An important implication of this theorem is that the agents will be motivated to normalize their own assessment of the value of any plan given their beliefs about the global utility. Thus, if we have self-interested rational agents and a payoff system such as that described here, the system no longer needs to do any correction of the agents' declarations. Instead, all it needs do is provide agents with feedback about the actual value of their plans.

Note that any reward policy that allows near misses (e.g., rewards agents also in the case where $| d_i - d_i^* | < \delta$ for some small δ) would make the process sensitive to possible manipulation (e.g., an agent may try to overestimate the value of a plan within the bounds of δ as to increase the chances of its execution).

Distribution of the Process

So far, we have not taken a fully distributed approach; we assumed a central scheduler. We next consider how to loosen this requirement. We consider a simple example from the DIPART domain.

Assume that at a certain decision point there are four agents $a_1 \ldots a_4$, each of which has generated the following plans (described by their resource usage, r_i (number of trucks in our case), and their normalized value-time density (d_i):[4] $r_{1_1} = 1$, $d_{1_1} = 11$, $r_{1_2} = 33$, $d_{1_2} = 43$, $r_{2_1} = 21$, $d_{2_1} = 31$, $r_{2_2} = 45$, $d_{2_2} = 55$, $r_{3_1} = 23$, $d_{3_1} = 33$, $r_{3_2} = 43$, $d_{3_2} = 53$, $r_{4_1} = 11$, $d_{4_1} = 21$, $r_{4_2} = 55$, $d_{4_2} = 55$. Assuming that there are 110 trucks available, the agents should derive the feasible combination with the maximal value-density. The optimal feasible combination is made of $\{d_{1_1}, d_{4_1}, d_{2_1}, d_{1_2}, d_{3_2}\}$ with a resulting aggregated value of $159 (= 11 + 21 + 31 + 43 + 53)$ which is attained by using 109 trucks. Of course in the general case finding this optimal solution is intractable, which is the reason we employ the approximation algorithm.

In this example, we use a simplified version of the approximation algorithm for the 0/1-Knapsack problem outlined in Figure 2. This version guarantees a solution within a factor of two from the optimal one (Garey & Jhonson 1979). It is derived from the original algorithm by the following modifications:

1. In Step 1 of the algorithm D_{max} should be initiated to the value of the plan with the highest *value-time* density, $D_{max} = max_i d_i$ (i.e., not value-time-resource density).
2. Instead of considering all the alternatives in Step 2 of the algorithm, the algorithm simply returns $\max(\text{GreedyValueDensityCompletion}(D_{max}), D_{max})$. (i.e., either the plan with the maximal value-time density, or the greedy value-time-resource density combination excluding this plan—whichever scores a higher value).

The essential idea underlining the distribution of the process is to derive the global combination through pairwise interactions. More formally we define the key steps of the process as follows:

- At the beginning of the process each agent determines the vector $\vec{d_i} = \langle d_{i_{max}}, Rest_i \rangle$ where $d_{i_{max}}$ is the value of the plan with the highest value-time density, and $Rest_i = d_{i_k}, \ldots, d_{i_m}$ denotes the rest of the values sorted in decreasing order of their value-time-resource density. For example $\vec{d_1} = \langle 43, 11 \rangle$.
- The combined vector $\vec{d_i} \oplus \vec{d_j}$ is simply defined to be: $\langle \max(d_{i_{max}}, d_{j_{max}}),$ $\text{D-Merge}(\min[d_{i_{max}}, d_{j_{max}}], Rest_i, Rest_j) \rangle$ where D-Merge maintains the value-time-resource decreasing order of the plans. For example, the combined vector of a_1 and a_2 is $\langle 55, 11, 31, 43 \rangle$.
- The final outcome is determined by the final global combination, $\langle d_{f_{max}}, Rest_f \rangle$ as determined at the end of the iterative pairwise aggregation process. It is set to be:

$$\max(d_{f_{max}}, \sum_{k=1, d_k \in Rest_f}^{m} d_k \text{ such that} \sum_{k=1}^{m} r_k \leq \mathcal{R}^t)$$

e.g., either the plan with the highest value, or the aggregated value of the iterative greedy selection of plans in $Rest_f$ that can be executed in parallel.

We here consider two alternative ways to conduct the pairwise combination—the *ring topology* and the *tree topology*:

In a *tree topology*, there are $\frac{n}{2}$ pairs of agents. Each pair $\{a_i, a_j\}$ finds the combined vector of its members $\langle \vec{d_i} \oplus \vec{d_j} \rangle$. Then we combine these pairs so as to form $n/4$ groups of four agents each. These groups are then paired, and together they determine the combined vector of the joint group, and so forth (resulting in a binary tree). The combined vector of the entire group is the combination of the last two groups. This process takes $\log_2 n$ steps and involves $2 \times (n-1)$ messages.

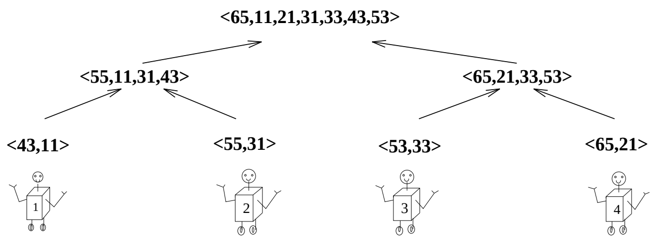

Figure 3: Binary Pairwise Schedule

Figure 3 describes how the tree topology would work in our example. Assuming that the agents are initially divided into the groups $\{a_1, a_2\}$ and $\{a_3, a_4\}$. The resulting global combination, $\vec{d_f} = \langle 65, 11, 21, 31, 33, 43, 53 \rangle$, is derived at the root of the tree. The value of the feasible greedy combination of of Rest_f is $139 (= 11 + 21 + 31 + 33 + 43)$ which uses $89 (= 1 + 11 + 21 + 23 + 33)$ trucks. Since this value is greater than 65 this the final combination chosen. That is, plans p_{1_1} ,p_{4_1}, p_{2_1}, p_{3_1}, and p_{1_2} will be executed in parallel.

[4] Although d and r are two different and only remotely dependent measures, for the clarity of the representation the artificial values used in this example are such that $d = r + 10$.

The *ring topology* assumes a *cyclic* order over the agents. The process works as follows. a_1 hands in its vector $(\vec{d_1})$ both to a_2 and a_3. a_2 calculates the combination $\langle \vec{d_2} \oplus \vec{d_1} \rangle$ and hands it to both a_3 and a_4 and so forth. In the general case a_i calculates the combination of its own vector with the aggregated ith combination, and hands the result to both a_{i+1} and a_{i+2}. Finally, at the nth step a_n calculates the global combination and hands it to a_1 to broadcast the result. The resulting process in our scenario is described in Figure 4.

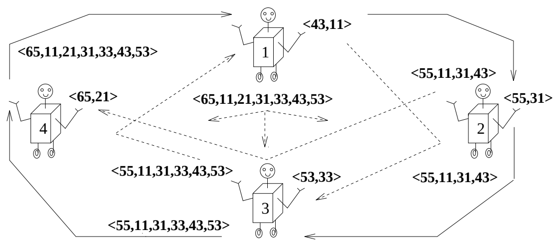

Figure 4: Ring Schedule

Recall that the reward policy that we have developed in the previous section guarantees an accurate assessment by the agent regarding the value of its own plans. However, each agent may still profit from tampering with the *other* agent's assessments (e.g., an agent may be tempted to falsely undermine another agent's plans in order to make his plans look relatively better). For this reason, the aggregated combination is sent *twice* at each step (from agent a_i to both a_{i+1} and a_{i+2}— as illustrated by the dashed lines in Figure 3). This duplicate transmission makes it possible for a_{i+2} to verify that a_{i+1} didn't alter the aggregated combination which he received from a_i. Thus, the temptation of passing on a false statement about the others' vectors is avoided (although the process is still sensitive to manipulation by *coalitions*, but this is not our concern here). Note that in the tree topology this problem does not exist, since *two* agents calculate the combination.

For example, consider the third step in the the process above: by altering the p_{1_2} value to be ≤ 40, a_3 can cause p_{3_2} to be sorted before p_{1_2} and thus make the system execute both its plans instead of just one, ending up with an additional potential gain. Surprisingly, in this case the resulting global combination will actually improve. However, in other cases similar manipulative behavior will result in less efficient global combination, and the deviation boundary of the approximation algorithm would no longer be guaranteed.

Conclusions

We have considered techniques suitable for combining individual agent plans into a global system plan, maintaining a commitment to considerations of global utility that may differ from individual agent utilities. We presented a three-stage heuristic reduction process, consisting of a transformation from local to global utility measures, a global assessment of the local evaluations of agents, and approximation algorithms to maximize resource usage over time. We considered how these techniques could be used with self-motivated agents (who must be harnessed to system-wide purposes), and showed that a reward function could be used to make accurate approximation on the part of agents the dominant rational policy. Finally, we showed how the overall process could be distributed among a group of agents, doing away with the need for a central scheduler.

Acknowledgments

This work has been supported in part by the Air Force Office of Scientific Research (Contract F49620-92-J-0422), by the Rome Laboratory (RL) of the Air Force Material Command and the Defense Advanced Research Projects Agency (Contract F30602-93-C-0038), by an NSF Young Investigator's Award (IRI-9258392) to Prof. Martha Pollack, by the Israeli Ministry of Science and Technology (Grant 032-8284) and by the Israel Science Foundation (Grant 032-7517).

References

Boddy, M., and Dean, T. L. 1994. Deliberation scheduling for problem solving in time-constraint environments. *Artificial Intelligence* 67:245–285.

Durfee, E. H.; Lesser, V. R.; and Corkill, D. D. 1987. Cooperation through communication in a distributed problem solving network. In Huhns, M. N., ed., *Distributed Artificial Intelligence*. Los Altos, California: Morgan Kaufmann Publishers, Inc. chapter 2, 29–58.

Ephrati, E., and Rosenschein, J. S. 1991. The Clarke Tax as a consensus mechanism among automated agents. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 173–178.

Ephrati, E., and Rosenschein, J. S. 1993. Multi-agent planning as a dynamic search for social consensus. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 423–429.

Ephrati, E., and Rosenschein, J. S. 1994. Divide and conquer in multi-agent planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 375–380.

Ephrati, E.; Perry, M.; and Rosenschein, J. S. 1994. Plan execution motivation in multi-agent systems. In *Proceedings of The Second International Conference on AI Planning Systems*, 37–42.

Etzioni, O. 1991. Embedding decision-analytic control in a learning architecture. *Artificial Intelligence* 49:129–159.

Garey, M. R., and Johnson, D. S. 1979. *Comouters and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. E. Freeman and Company.

Horowitz, E., and Sahni, S. 1978. *Fundamentals of Computer Algorithms.* Rockville, Maryland: Computer Sciece Press.

Jensen, E. D.; Locke, C. D.; and Tokuda, H. 1985. A time-driven scheduling model for real-time operating systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, 112–122.

Kraus, S. 1993. Agents contracting tasks in non-collaborative environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 243–248.

Locke, C. 1986. *Best-Effort Decision Making for Real-time Scheduling.* Ph.D. Dissertation, Carnegie-Mellon University.

Pollack, M. E.; Znati, T.; Ephrati, E.; Joslin, D.; Lauzac, S.; Nunes, A.; Onder, N.; Ronen, Y.; and Ur, S. 1994. The DIPART project: A status report. In *Proceedings of the Annual ARPI Meeting.*

Reece, G. A.; Tate, A.; Brown, D. I.; and Hoffman, M. 1993. The PRECiS environment. Technical report, ARPA-RL/CPE.

Rosenschein, J. S., and Zlotkin, G. 1994. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers.* Cambridge, Massachusetts: MIT Press.

Sandholm, T. 1993. An implementation of the contract net protocol based on marginal calculations. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 256–262.

Sandip, S.; Sekaran, M.; and Hale, J. 1994. Learning to coordinate without sharing information. In *Proceedings of the Twelfth national Conference on Artificial Intelligence*, 426–431.

Smith, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* C-29(12):1104–1113.

Wellman, M. P. 1992. A general equilibrium approach to distributed transportation planning. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 282–289.

Wellman, M. P. 1993. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research* 1:1–23.

Zlotkin, G., and Rosenschein, J. S. 1993. A domain theory for task oriented negotiation. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 417–422.