Multi-Agent A* for Parallel and Distributed Systems (Extended Abstract)

Raz Nissim
Ben-Gurion University of the Negev
Be'er-Sheva, Israel
raznis@cs.bgu.ac.il

Ronen I. Brafman
Ben-Gurion University of the Negev
Be'er-Sheva, Israel
brafman@cs.bgu.ac.il

ABSTRACT

Search is among the most fundamental techniques for problem solving, and A* is probably the best known heuristic search algorithm. In this paper we adapt A* to the multi-agent setting, focusing on multi-agent planning problems. We provide a simple formulation of multi-agent A*, with a parallel and distributed variant. Our algorithms exploit the structure of multi-agent problems to not only distribute the work efficiently among different agents, but also to remove symmetries and reduce the overall workload. Given a multi-agent planning problem in which agents are not tightly coupled, our parallel version of A* leads to super-linear speedup, solving benchmark problems that have not been solved before. In its distributed version, the algorithm ensures that private information is not shared among agents, yet computation is still efficient – sometimes even more than centralized search – despite the fact that each agent has access to partial information only.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms

Keywords

Distributed Search, Parallel search, Multi-Agent Planning.

1. INTRODUCTION

A* is probably the most celebrated heuristic search algorithm. Its good theoretical properties make it the favorite algorithm when searching for a provably optimal solution. The main contribution of this paper is MA-A*, a multi-agent formulation of A*. MA-A* attempts to make the most of the parallel nature of the system, i.e., the existence of multiple computing agents, while respecting its distributed nature, when relevant, i.e., the fact that some information is local to an agent, and cannot be shared. It is not a shallow parallelization or distribution of A*, as some successful parallel implementations of A* [4]. Rather, it is structure-aware, using the distinction between local and globally relevant actions and propositions to focus the work of each agent, dividing both states and operators among the agents, and exploiting symmetries that arise from the multi-agent structure. Moreover, MA-A* reduces exactly to A* when there is a single agent, unlike existing multi-core search methods [2]. MA-A* comes in two flavors, a parallel one and a distributed one, that differ only in the nature of the heuristic functions used.

To evaluate MA-A* we apply it to a number of multi-agent planning problems, comparing its performance to the best current optimal centralized planner and to the best (non-optimal) distributed planner. In the parallel case, our preliminary experiments show super-linear speed-up, as opposed to sublinear speedup by the best parallel planner, on problems in which agents are not tightly coupled. This stems from the fact that our algorithm is able to exploit the internal structure of the problem, and not only the added computational power. Using this variant, we were able to solve a number of planning problems that were so far beyond the reach of the best centralized optimal planners, and show up to $\times 20$ speedup on problems solved by both systems. In the distributed case, the agents are constrained to use only information that is directly accessible to them, i.e., information about their own operators and non-private aspects of the operators of other agents. Thus, this variant is truly distributed, and private information is not shared. In that setting, one would hope that the distributed algorithm would do not much worse than the centralized one (which has access to all information, but less computing power). Here, we see that the lack of global information is costly. Yet, even now, as long as the system is somewhat decoupled, the distributed algorithm can outperform the centralized one.

2. MULTI-AGENT A*

A MA-STRIPS problem [1] for a set of agents $\Phi = \{\varphi_i\}_{i=1}^k$ is given by a 4-tuple $\Pi = \langle P, \{A_i\}_{i=1}^k, I, G\rangle$, where P is a finite set of propositions, $I \subseteq P$ and $G \subseteq P$ encode the initial state and goal, respectively, and for $1 \leq i \leq k$, A_i is the set of actions agent φ_i is capable of performing. Each action $a = \langle pre(a), eff(a)\rangle$ is given by its preconditions and effects.

The MA-STRIPS model distinguishes between private and public variables and operators. A *private* variable of agent φ is required and affected only by the actions of φ. An action is *private* if all variables it affects and requires are private. All other actions are classified as *public*. That is,

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

φ's private actions affect and are affected only by φ, while its public actions may require or affect the actions of other agents. For ease of presentation we assume that all actions that achieve a goal condition are considered *public*.

MA-A*, presented in algorithms 1-3, is a distributed variation of A*, which maintains a separate search space for each agent. Each agent maintains an *open list* of states that are candidates for expansion and a *closed list* of already expanded states. It expands the state with the minimal $f = g + h$ value in its open list. When an agent expands state s, it uses its own operators only. This means that two agents expanding the same state will generate *different* successor states.

Algorithm 1 MA-A* for Agent φ_i

```
1: while did not receive true from a solution verification
   procedure do
2:    for all messages $m$ in message queue do
3:       process-message($m$)
4:    $s \leftarrow extract - min(openlist)$
5:    expand($s$)
```

Algorithm 2 process-message($m = \langle s, g_{\varphi_j}(s), h_{\varphi_j}(s)\rangle$)

```
1: if $s$ is not in open or closed list or $g_{\varphi_i}(s) > g_{\varphi_j}(s)$ then
2:    add $s$ to open list and calculate $h_{\varphi_i}(s)$
3:    $g_{\varphi_i}(s) \leftarrow g_{\varphi_j}(s)$
4:    $h_{\varphi_i}(s) \leftarrow max(h_{\varphi_i}(s), h_{\varphi_j}(s))$
```

Algorithm 3 expand(s)

```
1: move $s$ to closed list
2: if $s$ is a goal state then
3:    broadcast $s$ to all agents
4:    initiate verification of stable property $f_{lower-bound} \geq$
      $g_{\varphi_i}(s)$
5:    return
6: for all agents $\varphi_j \in \Phi$ do
7:    if the last action leading to $s$ was public and $\varphi_j$ has
      a public action for which all public preconditions hold
      in $s$ then
8:       send $s$ to $\varphi_j$
9: apply $\varphi_i$'s successor operator to $s$
10: for all successors $s'$ do
11:    update $g_{\varphi_i}(s')$ and calculate $h_{\varphi_i}(s')$
12:    if $s'$ is not in closed list or $f_{\varphi_i}(s')$ is now smaller than
       it was when $s'$ was moved to closed list then
13:       move $s'$ to open list
```

Since no agent has complete knowledge of the entire search space, messages must be sent, informing agents of open search nodes relevant to them. Agent φ_i characterizes state s as relevant to agent φ_j if φ_j has a public operator whose public preconditions (the preconditions φ_i is aware of) hold in s. In principle, a relevant state *must* be sent to φ_j (and this is what A* would effectively do). However, in some cases, this can be avoided, and there is also some flexibility as to when precisely the message will be sent. We discuss these finer details later, and for now, assume a relevant state is sent once it is generated.

The messages sent between agents contain the full state s, i.e. including both public and private variable values, as well as the cost of the best plan from the initial state to s found so far, and the sending agent's heuristic estimate of s. When agent φ receives a state via a message, it checks whether this state exists in its open or closed lists. If it does not appear in these lists, it is inserted into the open list. If a copy of this state with higher g value exists in the open list, its g value is updated, and if it is in the closed list, it is reopened. Otherwise, it is discarded. Whenever a received state is (re)inserted into the open list, the agent computes its local h_φ value for this state, and assigns the maximum of its h_φ value and the h value in the received message.

Once an agent expands a solution state s, it sends s to all agents and initiates the process of verifying its optimality. When the solution is verified as optimal, the agent initiates the trace-back of the solution plan. This is also a distributed process, which involves all agents that perform some action in the optimal plan. When the trace-back phase is done, a terminating message is broad-casted.

Termination detection is done using Chandy and Lamport's *snapshot algorithm* [3], which enables a process to create an approximation of the global state of the system, without "freezing" the distributed computation.

In the parallel setting, MA-A* allows each agent complete knowledge of both private and public operators of all agents. Thus, all agents compute (and can share) a single, global heuristic function, meaning that $h_{\varphi_i}(s) = h_{\varphi_j}(s)$ for all agents $\varphi_i, \varphi_j \in \Phi$ and for all states s. In the distributed setting, we assume that agents have access to public information and their own private information only. Because each agent has different information, it must compute its own local heuristic function. Thus, each agent can compute its heuristic estimate using a domain description that contains its own actions, as well as all public actions projected to public variables. The algorithm is completely agnostic as to how the agent uses this description to compute its private heuristic function. This allows us great flexibility, since different agents may use different heuristics. In fact, this is the *essence* of distributed search – each agent is a separate entity, capable of making choices regarding how it performs search.

Detailed experimental results, as well as proof of correctness and optimality are available in a technical report [5].

3. REFERENCES

[1] R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, pages 28–35, 2008.

[2] E. Burns, S. Lemons, R. Zhou, and W. Ruml. Best-first heuristic search for multi-core machines. In *IJCAI*, pages 449–455, 2009.

[3] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.

[4] A. Kishimoto, A. S. Fukunaga, and A. Botea. Scalable, parallel best-first search for optimal sequential planning. In *ICAPS*, 2009.

[5] R. Nissim and R. Brafman. Multi-agent a* for parallel and distributed systems. Technical Report 12-03, Department of Computer Science, Ben-Gurion University of the Negev, 2012.