Multi-Agent A* for Parallel and Distributed Systems

Raz Nissim and **Ronen Brafman**
Ben-Gurion University of the Negev
Be'er Sheva, Israel
raznis,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.

Introduction

As interest in multi-agent (MA) systems grows, so does the need to provide tools for multi-agent problem solving. Search, in particular, is among the most fundamental techniques for problem solving, hence the importance of adapting search algorithms to the MA setting.

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* (Kishimoto, Fukunaga, and Botea 2009). 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 (Tu et al. 2009; Burns et al. 2009). 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, we show super-linear speed-up, 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. 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.

In developing and describing MA-A*, we focus on *multi-agent planning*, using planning terminology and evaluating the algorithm on multi-agent planning problems. We do this for two reasons. First, it is the domain of primary interest to us, and to a large community of researchers on planning and multi-agent planning. Second, there exists a very simple and elegant formulation of multi-agent planning by Brafman and Domshlak (Brafman and Domshlak 2008), which makes explicit important notions, and in particular that of private and public variables and actions. However, search spaces with their operators, and planning spaces with their actions are almost synonymous, and so this development could be easily cast in search terminology.

Background

As noted earlier, we focus on multi-agent search in the context of multi-agent planning. MA planning is a wide and well studied field of research (Durfee 2001; Jonsson and Rovatsos 2011; ter Mors et al. 2010), but the most appropriate framework for our work is the MA-STRIPS model (Brafman and Domshlak 2008), presented by Brafman and Domsh-

lak (BD, for short). This framework minimally extends the classical STRIPS problem to MA planning for cooperative agents. The benefit of using this minimalistic model is that it is easy to see how the well known relationship between (single-agent) planning and general search is maintained in the multi-agent case, and that insights obtained using it could apply to more complex models of multi-agent planning.

A MA-STRIPS problem 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, 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 the presentation of MA-A* and for the brevity of its proof, we assume that all actions that achieve a goal condition are considered *public*. This assumption must hold in order to maintain completeness of MA-A* as presented, but the algorithm is easily modified to remove it.

Given a model of a distributed system such as MA-STRIPS, it is natural to ask how to search for a solution. The best known example of distributed search is that of distributed CSPs (Yokoo et al. 1998), and various search techniques and heuristics have been developed for it (Meisels 2007). Planning problems can be cast as CSP problems (given some bound on the number of actions), and the first attempt to solve MA-STRIPS problems was based on a reduction to distributed CSPs. More specifically, BD introduced the *Planning as CSP+Planning* methodology for planning by a system of cooperative agents with private information. This approach separates the public aspect of the problem, which involves finding public action sequences that satisfy a certain distributed CSP, from the private aspect, which ensures that each agent can actually execute these public actions in a sequence. Solutions found are locally optimal, in the sense that they minimize δ, the maximal number of public actions performed by an agent. This methodology was later extended to the first fully distributed MA algorithm for MA-STRIPS planning, *Planning-First* (Nissim, Brafman, and Domshlak 2010). *Planning First* was shown to be very efficient in solving problems where the agents are very loosely coupled, and where δ is very low. However, it does not scale up as δ rises, mostly due to the large search space of the Distributed CSP. As we will see later, the forward search algorithm we present scales much better *and* leads to a globally optimal solution.

We note that much work has been done attempting to solve MA planning in a partially observable and stochastic setting (using the DEC-POMDP model) (Szer, Charpillet, and Zilberstein 2005). This work, which uses different, more specialized methods and heuristics, falls out of the scope of this paper, which presents a *general* algorithm for solving the classical MA planning problem, using the MA-STRIPS formalism.

There is a growing interest in the use of parallelization techniques for scaling up planning algorithms. Recent results (Helmert and Röger 2008) show that in some cases, even almost perfect heuristics will not prevent exploring an exponential number of search nodes. Parallelization, as an orthogonal method of speeding up search, can continue the improvement of future planners. Kishimoto et al. (Kishimoto, Fukunaga, and Botea 2009) presented HDA*, a simple and effective parallelization of A*. HDA* distributes work between processors using a hash function, which assigns each state to a unique process. Communications are asynchronous and non-blocking, and duplicate detection is performed locally by each processor. HDA* was shown to achieve significant speedup using 4 processors, and to scale well up to 128 processors. Its efficiency (speedup divided by number of processors) ranges from 0.9 on 4-core machines, to 0.25 using 128 processors, constituting in sublinear speedup.

Multi-Agent A*

Overview

MA-A* 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.

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[1]. 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

[1]It may appear that agents are revealing their private data because they transmit their private state in their messages. However, as will be apparent in the algorithm, other agents do not use this information in any way, nor alter it. They simply copy it to future states. Only the agent itself can change the value of its private state. Consequently, this data can be encrypted arbitrarily – it is merely used as an ID by other agents.

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.

The MA-A* Algorithm

Algorithms 1-3 depict the MA-A* algorithm for agent φ_i.

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 ← 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_{φi}(s) > g_{φj}(s) then
2:     add s to open list and calculate h_{φi}(s)
3:     g_{φi}(s) ← g_{φj}(s)
4:     h_{φi}(s) ← max(h_{φi}(s), h_{φj}(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} ≥ g_{φi}(s)
5:     return
6: for all agents φj ∈ Φ do
7:     if the last action leading to s was public and φj has a public action for which all public preconditions hold in s then
8:         send s to φj
9: apply φi's successor operator to s
10: for all successors s′ do
11:     update g_{φi}(s′) and calculate h_{φi}(s′)
12:     if s′ is not in closed list or f_{φi}(s′) is now smaller than it was when s′ was moved to closed list then
13:         move s′ to open list
```

The pseudo-code is mostly self-explanatory, but some of its finer issues require further discussion.

Termination Detection Unlike in A*, expansion of a goal state in MA-A* does not necessarily mean an optimal solution has been found. In our case, a solution is known to be optimal only if all agents prove it so. Intuitively, a solution state s having solution cost f^* is known to be optimal if there exists no state s' in the open list or the input channel of some agent, such that $f(s') < f^*$. In other words, solution state s is known to be optimal if $f(s) \leq f_{lower-bound}$, where $f_{lower-bound}$ is a lower bound on the f-value of the entire system (which includes all states in all open lists, as well as states in messages that have not been processed, yet).

To detect this situation, we use Chandy and Lamport's *snapshot algorithm* (Chandy and Lamport 1985), which enables a process to create an approximation of the global state of the system, without "freezing" the distributed computation. Although there is no guarantee that the computed global state actually occurred, the approximation is good enough to determine whether a stable property currently holds in the system. A property of the system is *stable* if it is a global predicate which remains true once it becomes true. Specifically, properties of the form $f_{lower-bound} \geq c$ for some fixed value *c*, are stable when h is a *globally consistent* heuristic function. That is, when f values cannot decrease along a path. In our case, this path may involve a number of agents, each with its h values. If each of the local functions h_φ are consistent, and agents apply the $\max$ operator when receiving a message, as explained above, this property holds.

We note that for simplicity of the pseudo-code we omitted the detection of a situation where a goal state does not exist. This can be done by determining whether the stable property *"there are no open states in the system"* holds, using the same *snapshot algorithm*.

Parallel vs. Distributed Search via Heuristic Choice The quality of the heuristic function plays an important role in the success or failure of a forward search algorithm. In MA-A*, the nature of the heuristic is also the distinguishing factor between parallel and distributed search.

In centralized search, the global heuristic function is computed having *complete knowledge* of the problem. Specifically, it is aware of all operators in the system. Parallel search attempts to speed-up centralized search using multiple processors that have access to the complete problem description. We achieve this by allowing each agent complete knowledge of both private and public operators of all agents. Thus, all agents compute (and can share) the 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. We refer to this version of MA-A* as MAP-A*.

Existing techniques for parallel search distribute either the *workload* (e.g. HDA* (Kishimoto, Fukunaga, and Botea 2009)) or the *operators* (e.g. ODMP (Vrakas, Refanidis, and Vlahavas 2001)) between processors. These distribution methods are mostly independent of the problem structure (hash function for workload distribution, for example). In contrast, MAP-A* distributes the workload *and* search operators between agents according to the **MA structure** of the problem.

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. More specifically, each agent knows the complete description of its private and public operators. In addition, it is aware of the public aspects of all agents' public operators. Recall that any public action a of agent φ_i can have both public and private preconditions and effects. Agents other than φ_i have access to a projected version of a, containing only public preconditions and effects. 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. We refer to this distributed version of MA-A* as MAD-A*.

Relevancy and Timing of the Messages State s is considered relevant to agent φ_j if it has a public action for which all public preconditions hold in s and the last action leading to s was public (line 7 of Alg. 3). This means that all states that are products of private actions are considered irrelevant to other agents. As it turns out, since private actions do not affect other agents' capability to perform actions, an agent must send only states in which the last action performed was public, in order to maintain completeness (and optimality, as proved in the next section). Regarding states that are products of private actions as irrelevant, decreases communication, while effectively pruning large, symmetrical parts of the search space. The exploitation of symmetry is discussed in further detail later on.

As was hinted earlier, there exists some flexibility regarding when these relevant states are sent. A* can be viewed as essentially "sending" every state (i.e., inserting it to its open list) once it is generated. In MA-A*, relevant states can be sent when they are expanded (as in the pseudo-code) or once they are generated (changing Alg. 3 by moving the for-loop on line 6 inside the for-loop on line 10). The timing of the messages is especially important in the distributed setting, where agents may have different heuristic estimations. Sending the messages once they are generated increases communication, but allows for states that are not considered promising by some agent to be expanded by another agent in an earlier stage. Sending relevant states when they are expanded, on the other hand, decreases communication, but delays the sending of states not viewed as promising. Experimenting with the two options, we found that the lazy approach, of sending the messages only when they are expanded, dominates the other, most likely because communication is costly.

Proof of Optimality

First, we prove the following lemmas regarding the solution structure of a MA planning problem. We must note that as it is presented, MA-A* maintains completeness only if all actions which achieve some goal condition are considered public. This property is assumed throughout this section, but the algorithm is easily modified to remove it.

Lemma 1. *Let $P = (a_1, a_2 \ldots, a_k)$ be a legal plan for a MA planning problem Π. Let a_i, a_{i+1} be two consecutive actions taken in P by different agents, of which at least one is private. Then $P' = (a_1, \ldots, a_{i+1}, a_i, \ldots, a_k)$ is a legal plan for Π and $P(I) = P'(I)$.*

Proof. By definition of private and public actions, and because a_i, a_{i+1} are actions belonging to different agents, $varset(a_i) \cap varset(a_{i+1}) = \emptyset$, where $varset(a)$ is the set of variables which affect and are affected by a. Therefore, a_i does not achieve any of a_{i+1}'s preconditions, and a_{i+1} does not destroy any of a_i's preconditions. Therefore, if s is the state in which a_i is executed in P, a_{i+1} is executable in s, a_i is executable in $a_{i+1}(s)$, and $a_i(a_{i+1}(s)) = a_{i+1}(a_i(s))$. Therefore, $P' = (a_1, \ldots, a_{i+1}, a_i, \ldots, a_k)$ is a legal plan for Π. Since the suffix $(a_{i+2}, a_{i+3}, \ldots, a_k)$ remains unchanged in P', $P(I) = P'(I)$, completing the proof. □

Corollary 1. *For a MA planning problem Π for which an optimal plan $P = (a_1, a_2, \ldots, a_k)$ exists, there exists an optimal plan $P' = (a'_1, a'_2, \ldots, a'_k)$ for which the following restrictions apply:*

1. *If a_i is the first public action in P', then $a_1, \ldots, a_i$ belong to the same agent.*
2. *For each pair of consecutive public actions a_i, a_j in P', all actions a_l, $i < l \leq j$ belong to the same agent.*

Proof. Using repeated application of Lemma 1, we can move any ordered sequence of private actions performed by agent φ, so that it would be immediately before φ's subsequent public action and maintain legality of the plan. Since application of Lemma 1 does not change the cost of plan, the resulting plan is cost-optimal as well. □

Before proving the optimality of our algorithm, we prove the following lemma, which is a MA extension to a well known result for A*. In what follows, we have tacitly assumed a *liveness* property with the conditions that every sent message eventually arrives at its destination and that all agent operations take a finite amount of time. Also, for the clarity of the proof, we assume the atomicity of the *expand* and *process-message* procedures.

Lemma 2. *For any non-closed node s and for any optimal path P from I to s which follows the restrictions of Lemma 1, there exists an agent φ which either has an open node s' **or** has an incoming message containing s', such that s' is on P and $g_\varphi(s') = g^*(s')$.*

Proof. : Let $P = (I = n_0, n_1, \ldots, n_k = s)$. If I is in the open list of some agent φ (φ did not finish the algorithm's first iteration), let $s' = I$ and the lemma is trivially true since $g_\varphi(I) = g^*(I) = 0$. Suppose I is closed for all agents. Let Δ be the set of all nodes n_i in P that are closed by some agent φ, such that $g_\varphi(n_i) = g^*(n_i)$. Δ is not empty, since by assumption, $I \in \Delta$. Let n_j be the element of Δ with the highest index, closed by agent φ. Clearly, $n_j \neq s$ since s is non-closed. Let a be the action causing the transition $n_j \rightarrow n_{j+1}$ in P. Therefore, $g^*(n_{j+1}) = g_\varphi(n_j) + cost(a)$.

If φ is the agent performing a, then n_{j+1} is generated and moved to φ's open list in lines 9-13 of Algorithm 3, where $g_\varphi(n_{j+1})$ is assigned the value $g_\varphi(n_j) + cost(a) = g^*(n_{j+1})$ and the claim holds.

Otherwise, a is performed by agent $\varphi' \neq \varphi$. If a is a public action, then all its preconditions hold in n_j, and therefore n_j is sent to φ' by φ in line 8 in Algorithm 3. If a is a private action, by the definition of P, the next *public* action a' in P is performed by φ'. Since private actions do not change the values of public variables, the public preconditions of a' must hold in n_j, and therefore n_j is sent to φ' by φ in line 8 in Algorithm 3. Now, if the message containing n_j has been processed by φ', n_j has been added to the open list of φ' in Algorithm 2 and the claim holds since $g_{\varphi'}(n_j) = g_\varphi(n_j) = g^*(n_j)$. Otherwise, φ' has an incoming (unprocessed) message containing n_j and the claim holds since $g_\varphi(n_j) = g^*(n_j)$. □

Corollary 2. *Suppose h_φ is admissible for every $\varphi \in \Phi$, and suppose the algorithm has not terminated. Then, for any optimal solution path P which follows the restrictions of Lemma 1 from I to any goal node $s^\star$, there exists an agent φ_i which either has an open node s **or** has an incoming message containing s, such that s is on P **and** $f_{\varphi_i}(s) \leq h^*(I)$.*

Proof. : By Lemma 2, for every restricted optimal path P, there exists an agent φ_i which either has an open node s **or** has an incoming message containing s, such that s is on P and $g_{\varphi_i}(s) = g^*(s)$. By the definition of f, and since h_{φ_i} is admissible, we have in both cases:

$$f_{\varphi_i}(s) = g_{\varphi_i}(s) + h_{\varphi_i}(s) = g^*(s) + h_{\varphi_i}(s)$$
$$\leq g^*(s) + h^*(s) = f^*(s)$$

But since P is an optimal path, $f^*(n) = h^*(I)$, for all $n \in P$, which completes the proof. □

Another lemma must be proved regarding the solution verification process. We assume global consistency of all heuristic functions, since all admissible heuristics can be made consistent by locally using the pathmax equation (Méro 1984), and by using the *max* operator as in line 4 of Alg. 2 on heuristic values of different agents. This is required since we need $f_{lower-bound}$ to be monotonic non-decreasing.

Lemma 3. *Let φ be an agent which either has an open node s **or** has an incoming message containing s. Then, the solution verification procedure for state s^* with $f(s^*) > f_\varphi(s)$ will return false.*

Proof. Let φ be an agent which either has an open node s **or** has an incoming message containing s, such that $f_\varphi(s) < f(s^*)$ for some solution node s^*. The solution verification procedure for state s^* verifies the stable property $p =$ "$f(s^*) \leq f_{lower-bound}$". Since $f_{lower-bound}$ represents the lowest f-value of any open or unprocessed state in the system, we have $f_{lower-bound} \leq f_\varphi(s) < f(s^*)$, contradicting p. Relying on the correctness of the snapshot algorithm, this means that the solution verification procedure will return false, proving the claim. □

We can now prove the optimality of our algorithm.

Theorem 1. *Algorithm 1 terminates by finding a cost-optimal path to a goal node, if one exists.*

Proof. : We prove this theorem by assuming the contrary - the algorithm does not terminate by finding a cost-optimal path to a goal node. 3 cases are to be considered:

1. *The algorithm terminates at a non-goal node.* This contradicts the termination condition, since the solution verification procedure is initiated only when a goal state is expanded.
2. *The algorithm does not terminate.* Since we are dealing with a finite search space, let $\chi(\Pi)$ denote the number of possible *non-goal* states. Since there are only a finite number of paths from I to any node s in the search space, s can be *reopened* a finite number of times. Let $\rho(\Pi)$ be the maximum number of times any *non-goal* node s can be reopened by any agent. Let t be the time point when all non-goal nodes s with $f_\varphi(s) < h^*(I)$ have been closed forever by all agents φ. This t exists since *a*) we assume liveness of message passing and agent computations; *b*) after at most $\chi(\Pi) \times \rho(\Pi)$ expansions of non-goal nodes by φ, all non-goal nodes of the search space must be closed forever by φ; and *c*) no goal node s^* with $f(s^*) < h^*(I)$ exists[2].
   By Corollary 2 and since an optimal path from I to some goal state s^* exists, some agent φ expanded state s^* at time t', such that $f_\varphi(s^*) \leq h^*(I)$. Since s^* is an optimal solution, if $t' \geq t$, $f_{lower-bound} \geq f_\varphi(s^*)$ at time t'. Therefore, φ's verification procedure of s^* will return true, and the algorithm terminates.
   Otherwise, $t' < t$. Let φ' be the last agent to close a non-goal state s with $f_{\varphi'}(s) < f_\varphi(s^*)$. φ' has s^* in its open list or as an incoming message. This is true because s^* has been broad-casted to all agents by φ, and because every time s^* is closed by some agent (when it expands it), it is immediately broad-casted again, ending up in the agent's open list or in its message queue. Now, φ' has no more open nodes with f-value lower than s^*, so it will eventually expand s^*, initiating the solution verification procedure which will return true, since $f_{lower-bound} \geq f_\varphi(s^*)$. This contradicts the assumption of non-termination.
3. *The algorithm terminates at a goal node without achieving optimal cost.* Suppose the algorithm terminates at some goal node s with $f(s) > h^*(I)$. By Corollary 2, there existed just before termination an agent φ having an open node s', or having an incoming message containing s', such that s' is on an optimal path and $f_\varphi(s') \leq h^*(I)$. Therefore, by Lemma 3, the solution verification procedure for state s will return false, contradicting the assumption that the algorithm terminated.

This concludes the proof. □

[2]This is needed since the number of *goal* node expansions is not bounded.

MA Planning Framework

One of the main goals of this work is to provide a general framework for solving the MA planning problem. We believe that such a framework will provide researchers with fertile ground for developing new search techniques and heuristics for MA planning.

We chose Fast Downward (Helmert 2006) (FD) as the basis for our MA framework – **MA-FD**. FD is currently the leading framework for planning, both in the number of algorithms and heuristics that it provides, and in terms of performance – winners of the past three international planning competitions were implemented on top of it. FD is also well documented and supported, so implementing and testing new ideas is relatively easy.

MA-FD uses FD's translator and preprocessor, with minor changes to support the distribution of operators to agents. In addition to the PDDL files describing the domain and the problem instance, MA-FD receives a file detailing the number of agents, their names, and their IP addresses. The agents do not have shared memory, and all information is relayed between agents using messages. Inter-agent communication is performed using the TCP/IP protocol, which enables running multiple MA-FD agents as processes on multi-core systems, networked computers/robots, or even the cloud. MA-FD is therefore fit to run as is on *any* number of (networked) processors, in both its parallel and distributed setting.

Both MAP-A* and MAD-A* are currently implemented, and since both settings have full flexibility regarding the heuristics used by agents, all admissible heuristics available on FD are also available on MA-FD. New heuristics are easily implementable, as in FD, and creating new search algorithms can also be done with minimal effort, since MA-FD provides the ground-work (parsing, communication, etc.).

Empirical results

In order to evaluate MA-A*, in both its parallel and distributed setting, we performed experiments on IPC (ICAPS) benchmarks, in domains where tasks can be naturally casted as MA problems. The *Satellites* and *Rovers* domains where motivated by real MA applications used by NASA. *Satellites* requires planning and scheduling observation tasks between multiple satellites, each equipped with different imaging tools. *Rovers* involves multiple rovers navigating a planetary surface, finding samples and communicating them back to a Lander. *Logistics* and *Zenotravel* are two transportation domains, where multiple vehicles transport packages or people to their destination.

For each planning problem, we ran two configurations of centralized A*, one using the LM-cut heuristic (Helmert and Domshlak 2009) and the other using the Merge&Shrink heuristic[3] (Helmert, Haslum, and Hoffmann 2007), as well as 4 configurations of MA-A*: MAP-A* and MAD-A* using LM-cut[4] and Merge&Shrink, running on multiple processors[5]. All experiments were run on a AMD Phenom 9550 2.2GHZ quad-core processor. Time limit was set at 1.5 hours, and memory usage was limited to 4GB, regardless of the number of cores used.

We begin by discussing performance of MA-A* in its parallel setting. Table 1 depicts the runtimes, number of expanded nodes and efficiency values (speedup divided by the number of processors). In Rovers and Satellites, domains in which the agents are loosely-coupled, MAP-A* overwhelmingly dominates centralized A*. MAP-A* outperformed A* in all but one very small instance of Rovers, solving 6 problems unsolved by A*. Running on multiple processors, MAP-A* exhibited super-linear speedup, showing efficiency ranging between 1 to 19.5 in problems solved by both planners. This result is unique because existing parallel planners exhibit sublinear speedup, bounding their efficiency to be ≤ 1. In fact, super-linear speedup suggests that MA-A* exploits the *structure* of the problem and not only the additional computational power. This is discussed in further detail in the next section. In Logistics and Zenotravel, where the agents are tightly-coupled, MAP-A*'s efficiency decreases. Using LM-cut, efficiency values range from 0.46 to 2, but even though efficiency decreased in these tightly-coupled systems, MAP-A* running on multiple processors still always outperforms its centralized counterpart. MAP-A* using Merge&Shrink does not perform well in these tightly-coupled domains. In $Zenotravel10$, for example, efficiency drops as low as 0.15, causing a $\times 2.2$ speed-down. In tightly coupled systems, many states are sent as messages, because many of the actions are public. Communication is more time consuming than local computations, causing a drop in efficiency. The situation becomes more acute when using Merge&Shrink, which in general is less accurate (but faster to compute) than LM-cut, because as more states are expanded, more are sent between agents.

Table 2 shows the running time and the average of the agents' initial state h-values of centralized A* and MAD-A* running on multiple processors, as well as running time and δ-values of Planning First, running on a single processor. MAD-A* performed very well in comparison to (non-optimal) Planning First, outperforming it in all but one task, and solving 6 more problems, while achieving optimality[6]. When comparing MAD-A* to centralized A*, our intuition is that efficiency will decrease, due to the inaccuracy of the heuristic estimates. In fact, the local heuristics of the agents are much less accurate than the global heuristics, as is apparent from the lower average h values of the initial state, and by the higher number of states expanded by MAD-A*. In the tightly-coupled domains, we do notice very low (sublinear) efficiency values. However, in Satellites and Rovers, MAD-A* using Merge&Shrink exhibits nearly linear and super-linear speedup, respectively, solving 2 problems not solved by centralized A*. Our intuition is that the

[3] We used the LFPA configuration of Merge&Shrink with abstraction size limit = 50K.

[4] LM-cut is not consistent, so in order to maintain completeness, we enforced local consistency using the pathmax equation.

[5] In problems having up to 4 agents, each agent was allocated a processor. For problems with 5 agents, one processor was shared by two agents.

[6] Efficiency of MAD-A* w.r.t Planning First is not shown, but is super-linear except in $Rovers5$ using LM-cut.

Table 1: Comparison of centralized A* and MAP-A* running on multiple processors. Running time (in sec.), number of expanded nodes and efficiency values are shown.

Problem	#	LM-cut heuristic					Merge&Shrink heuristic				
		A*		MAP-A*			A*		MAP-A*		
(# of processors)	agents	time	expands	time	expands	eff.	time	expands	time	expands	eff.
logistics7-1(4)	4	55.3	155289	27	504102	0.51	0	1624	0.8	36847	0
logistics8-1(4)	4	24.1	43665	13	168545	0.46	0.1	45	0.8	3552	0.03
logistics10-0(4)	5	203	193846	66.6	627314	0.76	81.7	3219478	75.9	3056185	0.27
Rovers3(2)	2	0	50	0	90	1.00	0	12	0	98	1.00
Rovers4(2)	2	0	9	0.04	68	0	0	9	0.08	123	0
Rovers5(2)	2	8.8	37397	1.8	18975	2.44	11.7	419110	5.8	266433	1.01
Rovers6(2)	2	–	–	236	2255393	∞	–	–	238	17402585	∞
Rovers7(3)	3	6.7	18315	1	12929	2.23	55.9	2890357	9.1	762620	2.05
Rovers8(4)	4	–	–	154	1271971	∞	–	–	–	–	N/A
Rovers12(4)	4	12.1	15222	0.9	10704	3.36	119	3577293	6.6	456744	4.51
Rovers14(4)	4	–	–	598	5311741	∞	–	–	–	–	N/A
satellites5(3)	3	1.3	1174	0.1	793	4.33	7	117756	2.2	82579	1.06
satellites6(3)	3	3.5	2976	0.2	1650	5.83	23.5	495348	0.4	19062	19.58
satellites7(4)	4	94.5	36652	12.4	53465	1.91	–	–	347	17042327	∞
satellites8(4)	4	–	–	94.8	345667	∞	–	–	–	–	N/A
satellites9(4)	5	–	–	105	2132756	∞	–	–	–	–	N/A
satellites10(4)	5	–	–	61.8	95192	∞	–	–	–	–	N/A
zenotravel9(3)	3	72.1	15408	20	29321	1.20	56.7	1590010	81.7	1526996	0.23
zenotravel10(3)	3	16.1	1587	4.3	3453	1.25	26.7	388274	60.5	770787	0.15
zenotravel12(3)	3	458	41311	57	41819	2.68	–	–	–	–	N/A
zenotravel13(3)	3	–	–	382	185827	∞	–	–	–	–	N/A

same properties that helped MAP-A* achieve super-linear speedup, kept MAD-A*, with its less accurate heuristics, on par with centralized search in loosely-coupled domains.

Discussion

MA-A* raises a number of research challenges and opportunities, which we now discuss. Naturally, the first question is why it works well in weakly coupled environments. It is known that when using a consistent heuristic, A* is optimal in the number of nodes it expands to recognize an optimal solution. In principle, it appears that MA-A* expands the same search tree, so it is not clear, a-priori, why we reach super-linear speedup. We believe there are two reasons for this: one is delayed expansion of some nodes, and the other is symmetry exploitation.

Delayed expansion refers to the fact that in MA-A*, when an agent expands a node, it does not apply all actions to it, but only its actions. This means that nodes are expanded in parts. This is somewhat analogous to pruning methods used in planning, known as helpful actions (Hoffmann and Nebel 2001). The idea behind helpful actions is to recognize which actions are more likely to be useful at the current state and delay (or completely ignore) the application of other actions. MA-A* is complete, and does not ignore actions. However, different agents will expand the same node at different stages, and when a solution is reached, some expansions that would have taken place in A* may not be applied in MA-A*. As expanding a state entails evaluating its successors, the positive effect of delayed expansions is especially felt when using heuristics such as LM-cut, which are accurate but expensive to compute.

Symmetry exploitation utilizes the notion of public and private actions. As we noted in Corollary 1, the existence of private actions implies the existence of multiple effect-equivalent permutations of certain action sequences. A* does not recognize or exploit this fact, and MA-A* does. Specifically, imagine that agent φ_i just generated state s using one of its public actions, and s satisfies the preconditions of some action a of agent φ_j. Agent φ_i will eventually send s to agent φ_j, and the latter will eventually apply a to it. Now, imagine that agent φ_i has a private action a' applicable at state s, resulting in the state $s' = a'(s)$. Because a' is private to φ_i, from the fact that a is applicable at s we deduce that a is applicable at s' as well. Hence, A* would apply a at s'. However, in MA-A*, agent φ_j would not apply a at s' because it will not receive s' from agent φ_i. Thus, MA-A* does not explore all possible action sequences. If a' (which generates s') is needed in the solution plan, MA-A* will insert it, but not before a (which is a public action of φ_j). Rather, it will insert it before the next public action of agent φ_i. We note that this type of symmetry does not pertain only to MA systems, but to any factored system having internal operators. Therefore, our intuition is that it could be exploited, even in single-agent problems, by re-factoring them into MA ones. How to do this remains an open question.

Perhaps the greatest practical challenge suggested by the distributed version of MA-A* is that of computing a global heuristic by a distributed system. In some domains, the existence of private information that is not shared leads to serious deterioration in the quality of the heuristic function, greatly increasing the number of nodes expanded. We believe that there are techniques that can be used to alleviate this problem. As a simple example, consider a public action

Table 2: Comparison of centralized A* and MAD-A* running on multiple processors. Running time (in sec.), average initial state h-values and efficiency values are shown.

		LM-cut heuristic					Merge&Shrink heuristic					Planning	
Problem	#	A*		MAD-A*			A*		MAD-A*			First	
(# of processors)	agents	time	init-h	time	init-h	eff.	time	init-h	time	init-h	eff.	time	δ
logistics7-1(4)	4	55.3	39	178	35	0.08	0	44	57	36	0	–	N/A
logistics8-1(4)	4	24.1	41	172	37	0.04	0.1	44	49.5	37	0	–	N/A
logistics10-0(4)	5	203	41	–	35	0	81.7	43	–	36	0	–	N/A
Rovers3(2)	2	0	11	0	6	1.00	0	9	0	6.5	1.00	0.3	2
Rovers4(2)	2	0	8	0	6	1.00	0	8	0	6	1.00	0.2	1
Rovers5(2)	2	8.8	16	10.4	10.5	0.42	11.7	20	4	10.5	1.46	9.3	3
Rovers6(2)	2	–	23	716	16.5	∞	–	27	325	17.5	∞	–	N/A
Rovers7(3)	3	6.7	15	6.2	11	0.36	55.9	14	7.2	9	2.59	38.5	3
Rovers8(4)	4	–	21	–	13	N/A	–	15	–	10	N/A	–	N/A
Rovers12(4)	4	12.1	16	36.7	9	0.08	119	16	22.2	8.25	1.34	–	N/A
Rovers14(4)	4	–	18	–	10	N/A	–	17	–	11	N/A	–	N/A
satellites5(3)	3	1.3	14	7	8.3	0.06	7	13	3.8	8	0.61	52.3	2
satellites6(3)	3	3.5	17	2.5	8.3	0.47	23.5	18	9.2	9.3	0.85	457	3
satellites7(4)	4	94.5	19	–	9	0	–	14	343	9.5	∞	–	N/A
satellites8(4)	4	–	21	–	10	N/A	–	15	–	10	N/A	–	N/A
satellites9(4)	5	–	22	–	13.8	N/A	–	18	–	11	N/A	–	N/A
satellites10(4)	5	–	26	–	12	N/A	–	17	–	10	N/A	–	N/A
zenotravel9(3)	3	72.1	18	1108	14	0.02	56.7	19	370	14	0.05	–	N/A
zenotravel10(3)	3	16.1	20	–	17	0	26.7	22	–	17	0	–	N/A
zenotravel12(3)	3	458	18	–	14	0	–	–	–	–	N/A	–	N/A
zenotravel13(3)	3	–	22	–	18	N/A	–	–	–	–	N/A	–	N/A

a_{pub} that can be applied only after a private action a_{priv}. For example, in the rover domain, a *send* message can only be applied after various private actions required to collect data are executed. If the cost of a_{pub} known to other agents would reflect the cost of a_{priv} as well, the heuristic estimates would be more accurate. Another possibility for improving heuristic estimates is using an additive heuristic. In that case, rather than taking the maximum of the agent's own heuristic estimate and the estimate of the sending agent, the two could be added. To maintain admissibility, this would require using something like cost partitioning (Katz and Domshlak 2008). One obvious way of doing this would be to give each agent the full cost of its actions and zero cost for other actions. The problem with this approach is that initially, when the state is generated and the only estimate available is that of the generating agent, this estimate is very inaccurate, since it assigns 0 to all other actions. In fact, the agent will be inclined to prefer actions performed by other agents, as they appear very cheap, and we see especially poor results in domains where different agents can achieve the same goal, as in the Rovers domain, resulting in estimates of 0 for many non-goal states.

Finally, the efficiency of existing parallel search methods decreases as the number of processors increases (Kishimoto, Fukunaga, and Botea 2009; Vrakas, Refanidis, and Vlahavas 2001). We have shown that the efficiency of MA-A*, on the other hand, depends more on the structure of the problem than the number of processors. This could mean that in loosely-coupled systems, even extremely large problems could become solvable by adding some computational power, which would not make much difference in the centralized case. For this intuition to be verified, however, a further study regarding MA-A*'s scaling behavior (with respect to the number of processors) must be conducted.

Conclusion

We presented MA-A*, a simple formulation of A* for MA systems. Its parallel and distributed variants are separated only by the way heuristic estimates are computed. The parallel variant, in which agents have complete knowledge of the system, exhibits super-linear speedup on problems where the agents are loosely coupled. In its distributed setting, MA-A* dominated the best (non-optimal) distributed planner, while achieving optimality. In some cases, MAD-A* outperformed its centralized counterpart, despite having less accurate heuristic estimates. We also presented the MA-FD framework for MA planning. MA-FD is based on the successful FD framework, and provides the necessary building blocks for implementing MA planning algorithms and heuristics. Our hope is that having a *simple* and *effective* MA planning framework will increase interest in MA planning research, as FD has done for centralized planning.

References

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

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

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

Durfee, E. H. 2001. Distributed problem solving and planning. In *EASSS*, 118–149.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *AAAI*, 944–949.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.* 14(1):253–302.

ICAPS. The international planning competition. `http://www.plg.inf.uc3m.es/ipc2011-deterministic/`.

Jonsson, A., and Rovatsos, M. 2011. Scaling up multiagent planning: A best-response approach. In *ICAPS*.

Katz, M., and Domshlak, C. 2008. Optimal additive composition of abstraction-based admissible heuristics. In *ICAPS*, 174–181.

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

Meisels, A. 2007. *Distributed Search by Constrained Agents: Algorithms, Performance, Communication (Advanced Information and Knowledge Processing)*. Springer.

Méro, L. 1984. A heuristic search algorithm with modifiable estimate. *Artif. Intell.* 23(1):13–27.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *AAMAS*, 1323–1330.

Szer, D.; Charpillet, F.; and Zilberstein, S. 2005. Maa*: A heuristic search algorithm for solving decentralized pomdps. In *UAI*, 576–590.

ter Mors, A.; Yadati, C.; Witteveen, C.; and Zhang, Y. 2010. Coordination by design and the price of autonomy. *Autonomous Agents and Multi-Agent Systems* 20(3):308–341.

Tu, P. H.; Pontelli, E.; Son, T. C.; and To, S. T. 2009. Applications of parallel processing technologies in heuristic search planning: methodologies and experiments. *Concurrency and Computation: Practice and Experience* 21(15):1928–1960.

Vrakas, D.; Refanidis, I.; and Vlahavas, I. P. 2001. Parallel planning via the distribution of operators. *J. Exp. Theor. Artif. Intell.* 13(3):211–226.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Knowl. Data Eng.* 10(5):673–685.