A General, Fully Distributed Multi-Agent Planning Algorithm

Raz Nissim
Department of Computer Science
Ben-Gurion University, Israel

raznis@cs.bgu.ac.il

Ronen I. Brafman
Department of Computer Science
Ben-Gurion University, Israel

brafman@cs.bgu.ac.il

Carmel Domshlak
Faculty of Industrial Engineering
and Management
Technion, Israel

dcarmel@ie.technion.ac.il

ABSTRACT

We present a fully distributed multi-agent planning algorithm. Our methodology uses distributed constraint satisfaction to coordinate between agents, and local planning to ensure the consistency of these coordination points. To solve the distributed CSP efficiently, we must modify existing methods to take advantage of the structure of the underlying planning problem. In multi-agent planning domains with limited agent interaction, our algorithm empirically shows scalability beyond state of the art centralized solvers. Our work also provides a novel, real-world setting for testing and evaluating distributed constraint satisfaction algorithms in structured domains and illustrates how existing techniques can be altered to address such structure.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Plan execution, formation, and generation, Heuristic methods*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence - *Intelligent agents*, *Multiagent systems*.

General Terms

Algorithms, Performance, Experimentation.

Keywords

Multi-Agent Planning, Distributed Problem Solving, Distributed Constraint Satisfaction, Single-Agent Planning.

1. INTRODUCTION

Planning and decision making processes take place in many day-to-day situations: a shipping and logistics company has packages to be delivered all over the world, a transportation company schedules trains and buses, and NASA's automated Mars Rovers must carry out plans that make optimal use of their resources.

In many cases, the systems for which we plan are naturally viewed as Multiagent (MA) systems. For example, the shipping company is composed of multiple agents that perform the shipping tasks: the pilots/airplanes that ship packages between airports, and the local drivers/trucks that ship packages within a certain locality. In [2], Brafman & Domshlak [BD] posed the

following question: "Can a centralized planner, planning for such a system exploit its MA structure in order to improve its worst case time-complexity?" Using a very simple and general MA planning formalism MA-STRIPS that minimally extends the well known STRIPS language [4], they showed that under certain conditions, the answer is positive. Roughly speaking, when the MA system is loosely coupled and there exist plans in which the load on agents is reasonably balanced, adding agents into the system results in only polynomial increase of the time complexity.

BD solve the planning problem by compiling it into a particular CSP. Beyond the utilization of MA structure in formulating this CSP, BD's method holds special interest to the Multiagent systems' planning community since it also explains how a fully distributed planning algorithm can be obtained by replacing the (centralized) CSP algorithms with distributed CSP (*DisCSP*) algorithms, and running local planning routines within each agent. The motivation for fully distributed algorithms has been discussed extensively in the *DisCSP* literature. There are many instances of systems that are engaged in some cooperative behavior, but where each agent has its own capabilities and its own local planning tools that are best viewed as black-boxes by the other entities. Moreover, the ability to plan in a distributed manner can enhance the robustness of plan execution by a MA system – if something goes wrong at plan execution time, re-planning would not require sending all information to and relying on some central solver. Finally, there is growing interest in the use of parallelization techniques for scaling up planning algorithms [11]. A fully distributed algorithm provides a natural path to parallelizing the solution of planning problems that have a natural MA structure.

The result of BD is a powerful one, theoretically, but it raises an important question: Can it lead to planning algorithms that are efficient in practice? There are two key issues: First, BD's approach generates a CSP that has a small number of variables, each with a huge domain. Even the most sophisticated current CSP solution algorithms reduce to almost breadth-first search on such problems. Hence, the first hurdle in making this approach practical is restructuring the CSP, carefully pruning its domains, and using dynamic (lazy) domain generation to control its actual size. The second problem is that BD's results rely on a CSP algorithm (variable elimination) that comes with good worst-case performance guarantees, but is impractical in solving real-world problems. The obvious solution here would be to use state-of-the-art backtrack-based solution algorithms. Unfortunately, as our empirical evaluation will show, existing methods and heuristics cannot cope with this problem, and we must formulate modified algorithms that take into account problem structure in selecting the agent, variable, and value orders.

The resulting algorithm is a practical, fully distributed MA planning algorithm for planning problems described using the MA-STRIPS language. On classical, centralized, problem domains, this distributed algorithm is no match to state-of-the-art centralized planners. However, in loosely-coupled MA domains, where agents have little interaction with each other, it scales-up better than these planners. Our algorithm is general because it relies on the use of general principles -- inter-agent coordination is formulated as a CSP combined with local planning – and relies on the minimalistic MA-STRIPS language. These techniques extend to much more expressive formalisms that can deal with action costs, resource constraints, and more.

Beyond the MA planning community, our work should be of special interest to researchers of *DisCSP*s, whose formulation and solution play a central role in it. In recent years we have seen more powerful *DisCSP* algorithms emerge, which have most often been tested on randomly generated problems. Real-world instances often exhibit structure that is not present in randomly generated instances, and as our work shows, exploiting such structure is essential in scaling up solution methods in such domains. Our work provides an interesting case study showing how existing algorithms can be adapted to handle this structure more effectively.

The paper is structured as follows. The next section defines the MA planning model used, *DisCSP*s and BD's *Planning as CSP+Planning* methodology. Section 3 presents problems we face when using this methodology in practice, and how we propose to solve them. Our novel distributed search algorithm, *Planning First*, is described next, as well as methods of using planning mechanisms to improve the search. Section 4 shows experimental results on several planning domains, when comparing two versions of our algorithm, with the Fast Forward [3] centralized planner. A comparison of classical *DisCSP* heuristics with ones empowered by planning obtained knowledge is shown here as well. Section 5 summarizes the paper and its contributions and discusses issues that arose as well as suggestions for future work.

2. BACKGROUND

2.1 Multi-Agent Planning Model
We consider a classical MA planning setting where agents act with complete information and actions are deterministic. The problems considered are ones that can be expressed in MA-STRIPS [2], a minimalistic MA-extension of the STRIPS language [4]. MA-STRIPS can easily be extended with aspects such as time, resources, preferences, etc.

Definition 1: *A* **MA-STRIPS Problem** *for a system of agents* $\Phi = \{\varphi_i\}_{i=1}^k$ *is given by a quadruple* $\Pi = \langle P, \{A_i\}_{i=1}^k, I, G \rangle$, *where:*
- *P is a finite set of propositions,* $I \subseteq P$ *encodes the initial state, and* $G \subseteq P$ *encodes the goal conditions,*
- *For* $1 \leq i \leq k$, A_i *is the set of actions that the agent* φ_i *is capable of performing. Each action* $a \in A_i$ *has the standard STRIPS syntax and semantics, that is,* $a = \langle Pre(a), Add(a), Del(a) \rangle$ *is given by its preconditions, add effects and delete effects.*

We assume WLOG that actions have unique names.

To get a clearer picture of the MA-STRIPS model, consider the well known Logistics domain, in which a set of packages should be moved on a roadmap from their initial to their target locations using a given fleet of vehicles such as trucks, airplanes, etc. The packages can be loaded onto and unloaded off the vehicles, and each vehicle can move along a certain subset of road segments. Propositions are associated with each package location, on the map or in a vehicle, and with every truck location on the map. Possible actions are *drive*, *load*, and *unload*, each with its suitable parameters (e.g., *drive(truck, origin, destination)* and *load(package, truck, at-location)*). Associating each vehicle with an agent, we might assign this agent all *drive*, *load* and *unload* actions in which it is involved.

Such an MA-STRIPS problem Π induces dependencies on the agents Φ. In what follows, we use $vars(a)$ to denote $pre(a) \cup add(a) \cup del(a)$ and $effects(a)$ to denote $add(a) \cup del(a)$. Let $P_i = \bigcup_{a \in A_i} vars(a)$ be the set of all atoms affected by and/or affecting the actions of agent φ_i. By *internal* and *public* propositions of φ_i we refer to $P_i^{int} = P_i \setminus \bigcup_{\varphi_j \in \Phi \setminus \{\varphi_i\}} P_j$ and $P_i^{pub} = P_i \setminus P_i^{int}$ respectively. That is, if $p \in P_i^{int}$, no other agents can require or affect p. Using this definition of internal propositions, we can derive the partition $A_i = A_i^{int} \cup A_i^{pub}$ of agent φ_i's actions into *internal* and *public actions* respectively. That is, A_i^{int} is the set of all actions whose description contains only internal atoms of φ_i, while all other actions of φ_i are public.

To illustrate this important partition, let's go back to the Logistics domain. Here, since all vehicle locations are internal propositions, all the *move* actions are certainly internal to the respective vehicle agents. On the other hand, *load/unload* actions are public just if they affect the position of a package in some of its public locations, i.e., locations that can be reached by at least two agents. Given φ_i's action *a*, the projection of *a* onto φ_i's private propositions is denoted as $a|_{int} = \langle pre(a) \cap P_i^{int}, add(a) \cap P_i^{int}, del(a) \cap P_i^{int} \rangle$. If $a \in A_i^{int}$ then $a = a|_{int}$. Otherwise, $a|_{int}$ might have fewer propositions. For example, the public action *load(p, tr, loc)*, where *loc* is a public location, has the following preconditions: *at(p, loc)* and *at(tr, loc)*. It's internal projection, however, would require only *at(tr, loc)*, since *at(p, loc)* is a public proposition. The external projection of *a* onto φ_i's public propositions, $a|_{ext}$, is defined analogously.

A solution to a MA-STRIPS problem is equivalent to its STRIPS counterpart – an ordered sequence of actions taking the system from its initial state to a state containing all goal propositions.

The *agent interaction graph* IG_Π plays an important role in determining how loosely-coupled a problem is. The nodes of IG_Π correspond to the system's agents Φ. A directed edge from node φ_i to node φ_j exists in IG_Π if there exist actions $a_i \in A_i$ and $a_j \in A_j$ such that $effects(a_i) \cap pre(a_j) \neq \emptyset$. In other words, an edge from φ_i to φ_j indicates that φ_i either supplies or destroys a condition required by an action of φ_j. For example, we would have a directed edge between an airplane and a truck if the airplane unloads package *p* in a location reachable by the truck, since preconditions for loading *p* are supplied by the airplane. Edges in both directions between two agents are possible.

2.2 Distributed Constraint Satisfaction
A *DisCSP* is composed of a set of *k* agents $A_1, A_2 \ldots A_k$. Each agent A_i contains a set of constrained variables with a domain of values for each variable. A *constraint* is a subset of the Cartesian product of the domains of the constrained variables. A *binary*

constraint R_{ij} between any two variables X_i and X_j is a subset of the Cartesian product of their domains. In a *DisCSP*, the agents are connected by constraints between variables that belong to different agents [5]. In addition, each agent has a set of constrained variables, i.e. a *local constraint network*. An *assignment* of a variable is the pair $< var, val >$ where val is a value from var's domain. A *solution* to a *DisCSP* is a complete set of assignments that satisfies all the constraints. The act of checking whether locally assigned values satisfy non-local constraints is done by sending messages between agents. It is important to note that previous works on DisCSPs typically make the simplifying assumptions that each agent has exactly one variable and that all constraints are binary. Relaxing these assumptions to general cases is straightforward, but comes at the cost of more agents and/or larger domains. Since this work deals with **non-binary constraints** as well as **multi-variable agents**, these assumptions **will not** be made here.

2.3 Planning as CSP+Planning

The *Planning as CSP+Planning* methodology solves the MA-planning problem by separating the public and private aspects of the problems. Eventually, all aspects must be dealt with, and in a consistent manner.

The public part of the solution contains public "plans," i.e., sequences of public actions, for each agent. The consistency of these "plans" is ensured by seeking sequences of public actions that satisfy a certain CSP. That is, we formulate a CSP in which each agent has a single variable, and the possible values of this variable are sequences of public actions of bounded size. The constraints in this CSP express certain consistency requirements between these actions. Intuitively, these constraints correspond to standard planning constraints ("make sure that the preconditions of actions are true before it is executed," and "make sure the goal is true in the end"), but restricted to public propositions only. That is, at this stage, we ignore the actions' internal preconditions.

The private, or internal, part of the solution ensures that the agent can actually execute these public actions in a sequence. Formally, it can be viewed as a unary constraint on each of the variables in the CSP defined above. That is, it restricts the public action sequences of each agent to be locally consistent, meaning that these sequences can be extended with internal actions to ensure that the internal preconditions of each action are satisfied as well. Note that this is indeed a unary constraint.

This is where the term CSP+Planning comes from – the public aspects, or the coordination between agents is dealt with using a CSP, and the local, or internal aspect is dealt with using a planner.

The whole process is wrapped in an iterative deepening type search which gradually increases the upper bound, δ, on the number of public actions of each agent, along with the global plan for the whole system. These public actions serve as the coordination points between the agents. In tightly-coupled systems, one would expect the number of coordination points to be large, and this entire process is likely to be inefficient. In truly distributed and loosely-coupled systems, one would expect δ to be small, potentially increasing the efficiency of the process.

Returning to the Logistics domain, say a package needs to be transported from a location in central Boston to Manhattan. Two trucks are available, one in each city, as well as an airplane that can fly between the cities. Each truck would have one coordination point, corresponding to unloading the package at

Boston Airport or loading the package at NY-Airport. The airplane would pick up the package at Boston and Drop it off in NY, corresponding to two coordination points. Flight control and city navigation leading up to these coordination points would be up to the local planning of the airplane and trucks, respectively.

So far, relying on the formulation of BD, we described a centralized algorithm for solving the MA planning problem. As BD note, by using a *DisCSP* solver to handle the coordination part, we readily obtain a distributed planning algorithm. As this algorithm is the basis upon which we work, we now describe it in more detail.

Given δ, we define DisCSP$_{\Pi;\delta}$ as follows: for each agent we create a **variable** representing the agent's choice of coordination points. Each such choice is a vector of length δ, where each entry in the vector is assigned a pair (a, t), where a is the public action to be performed at time $t \in \{1, 2, ..., k\delta\}$. Notice that an entry could be empty, representing an action sequence shorter than δ. The **domain** of an agent's variable consists of the Cartesian product of the agent's public action sequences and all possible time assignments of length δ. Our next step is to create the **coordination constraints**, which verify the agent's public commitments with those of other agents. For every public precondition p of every coordination point (a, t) of agent φ_i, someone supplies p at time $t' < t$, and no one destroys p between t' and t. Next, we create the internal planning constraints for each agent. These constraints verify that the agent is capable of generating internal actions which provide the internal preconditions of the public actions it has committed to, in the right order. In other words, given an ordered sequence of coordination points, we try to solve a single agent planning problem while ensuring the solution contains this sequence. If and only if such a solution exists, the action sequence is feasible.

The high-level skeleton for this algorithm, as presented in [2], is depicted below. δ-increment messages are sent by a randomly selected initializing agent when it is informed that no solution exists for δ.

procedure <u>MA-Planning</u>(Π)
    $\delta := 1$
    loop
        Construct $DisCSP_{\Pi;\delta}$
        if (solve-csp($DisCSP_{\Pi;\delta}$)) **then**
            Reconstruct a plan p from solution
            return p
        else
            $\delta := \delta + 1$
    endloop

BD's work proves the *soundness* and *completeness* of this algorithm as well as tractability under certain conditions. Moreover, obtained plans are locally optimal, since the iterative δ-loop ensures that the maximal number of coordination points between agents is minimized, optimizing plans in this sense.

3. FROM THEORY TO PRACTICE

Theoretically, at this point we are done. We just need to take a black box *DisCSP* solver, and implement the above algorithm. However, when implementing this algorithm, we quickly came across several challenges. First, the constraints of the generated *DisCSP* are not binary. This is due to the disjunction of one aspect of the coordination constraint. Recall that one constraint stipulates

that "some agent must supply each precondition of each action in an agent's public plan". This constraint effectively glues together agents that otherwise might not affect one another, but who are able to supply the same condition to another agent, creating complex non-binary constraints. Since all *DisCSP* solvers assume constraints are binary, none could be used as a "black box" unless we change the encoding. The second problem is directing the distributed search. It is known that the order in which a *DisCSP* solver algorithm assigns the problem's variables has a potentially profound effect on its efficiency. Existing variable ordering heuristics, as well as the principles behind them (*fail-first* [6] for example) all suffer from a common flaw, stemming from the structure of the problem they help to solve. CSPs contain no information about variables and values other than constraints. Therefore, the heuristics can only use the constraints graph and domain sizes when making a choice. This is, of course, fine when solving a time-tabling or randomly generated problem, but when we encode a MA-Planning problem to a *DisCSP*, these heuristics seem to disregard information that could speed up the search.

The issue of non-binary constraints is solved using BD's **Extended Coordination Constraint**. Now, each coordination point, (a, t), would contain a requirement, in the form "I require proposition v from agent φ_j at time t'" for every public precondition of a. An agent's choice of a value would now explicitly state which agent should provide each of its public preconditions, and when. For example, for the action "Agent *truck* will load *package1* in airport at time 4" requires that the precondition *at(package1,airport)* hold before time 4. Now, who will achieve this proposition and when is part of the value choice made by the agent. For example, one possible value Agent *truck* could make would be: "I will load *package1* in airport at time 4 and agent *airplane* will supply *package1* at the airport at time 2." This unglues the providers, which now only need to worry about the condition they are explicitly required to supply. Generating the constraints for the problem created is rather straightforward. Two values are inconsistent if one requires a condition from a variable that does not supply it, or if one destroys the other's requirement.

This encoding has two main drawbacks. First, the variables' domain sizes are very large. Each variable now encompasses the entire set of the agent's public actions to be executed, their execution times, and which agent is required to achieve each of their preconditions. The variable's domain is therefore a Cartesian product of all the agent's possible public action sequences of length δ, all possible execution time sequences, and all possible requirement decisions for said action sequences. Therefore, when δ grows, these domains explode, rendering even a relatively small MA-Planning problem into a virtually unsolvable *DisCSP*. Second, since every variable encompasses all aspects of its agent's plan, we are left with little control over variable and value selection. This results in a "blind" search, which has no chance of handling large, structured planning problems.

We therefore must find an encoding that:

1) Keeps the agent interaction graph simple*,* so that agents will not interact with each other in complex constraints;
2) Keeps the domains relatively small;
3) Gives us control over variable and value selection;
4) Allows for the use of planning obtained knowledge, combined with known CSP solving principles and heuristics, to guide the search process.

We address these issues in the following subsections.

3.1 Separating the Agent's Variable

Instead of one variable that encompasses all aspects of a possible plan, we modify BD's encoding and create three separate variables: 1) **Actions Variable** (*ActVar*) – contains an ordered sequence of public actions, 2) **Times Variable** (*TimeVar*) – contains an ordered sequence of times to perform the selected actions and 3) **Requirements Variable** (*ReqVar*) – contains binding instructions for other agents in order to supply the actions' preconditions.

Splitting-up the variable, of course, does nothing to decrease the search space. However, we gain *flexibility and control in variable and value selection*. Now, through the correct use of variable ordering, effective pruning can be done due to the more delicate and precise branching of the search process. More importantly, the separation allows us to *use the fact that this is a planning problem to our advantage*. Whether centralized or distributed, a CSP doesn't have any information about the value of a variable other than its constraints. Therefore, when we encode all the aspects of an agent's plan as an assignment of a variable, we are actually losing information that can help direct the search efficiently! Two key pieces of information are 1) **whether or not the plan achieves the agent's goals** and 2) **whether the plan is locally feasible**. This information is, of course, encoded into the *DisCSP*'s constraints, but it cannot help as a heuristic for variable and value ordering. If we could, therefore, obtain these two key pieces of information, using them would improve our search in two ways: First, action sequences that are not locally feasible will be removed from *ActVar*'s domains. Since the action selection is independent of the actions' execution times and requirements, assigning *ActVar* first effectively prunes large branches of the search tree. Second, action sequences that are part of goal achieving plans are given preference when assigning the *ActVar*. This turns an otherwise blind search to one that's oriented toward the planning problem's goals.

3.2 Planning First Distributed Search

Until now, the agent's local planning procedure was used only to determine whether each of the agent's coordination points could be reached from the previous one using internal actions, i.e., whether the agent's commitments are feasible. Now, in order to create a search process that's **more suited for planning problems**, we present the *Planning First* methodology, which uses local planning more extensively. We then show how planning tools and methodologies help improve existing *DisCSP* heuristics as well as reducing the search space.

3.2.1 *Assigning the Variables*

Before assigning values to its variables, the agent *plans locally* using all of its actions (private and public) ignoring preconditions supplied by other agents. The plan must achieve all the agent's private sub-goals and must contain no more than δ public actions. Once this *relaxed plan* is found, *ActVar* is assigned the value containing the plan's public actions, arranged in their order in the plan. Now, we can generate the domain of *TimeVar* only including values that are consistent with the assigned action sequence, and assign a value for *TimeVar*. Assigning a value for *ReqVar* would now create action landmarks (actions and respective execution times, required from an agent) for the agent's neighbors in the agent interaction graph. When the next agent is selected, it performs relaxed planning with the action landmarks it has accumulated. If a plan achieving all the agent's sub-goals is

not found, the agent backtracks, sending a backtrack-CPA to a conflicting agent. Otherwise, it assigns its *ActVar* and continues to assign its *TimeVar* and *ReqVar* as before. Since the agent knows the public plans of previously assigned agents, *ReqVar*'s domain can be further filtered. When an agent receives a backtrack-CPA, meaning that there is no consistent assignment for the variables following the agent, it tries to reassign *ReqVar* and *TimeVar* in that order. If their domains are empty, *ActVar*'s assignment is added to the local planner's forbidden plans. The agent now tries to find a new plan different from the ones it has tried before. If no such plan exists, it backtracks as well.

The order in which an agent's variables are assigned follows the *action-first* principle. Choosing the actions first assures us that the plan is valid and goal-achieving, before we try to find its actions' execution times. Furthermore, action variables are known to be more constrained, either by action landmarks or local feasibility issues, than time variables. It is likely, then, that an assignment of an agent's *ActVar* would prune the search space better than an assignment of its *TimeVar*. Here, we start to feel the flexibility gained by a multi-variable agent, in controlling the search flow.

3.2.2 Empowering DisCSP Heuristics

Heuristics, especially *variable ordering*, are proven to greatly affect the efficiency of the search. In our case, we are dealing with extremely large domains and therefore an extremely large search space, where uninformed search would stand no chance. Since we already have a fixed order in which an agent assigns its variables, we now must choose a method for agent ordering.

Classical variable ordering heuristics like *Min-Domain* rely on the number of consistent values in a variable's domain and therefore require fully generated domains. Since our domains are generated dynamically (*ActVar* only chooses a value, without actually generating its domain and *TimeVar*'s domain is generated only after the assignment of *ActVar*), counting consistent values would entail exhaustive generation of domains, with much computational effort. Instead, in our algorithm agent selection is performed dynamically and will follow two principles: *goal-achieving* and *most-constrained*. The *goal-achieving* principle rates agents by the number of sub-goals they can achieve. Both private and public sub-goals are considered. Having goal-achieving agents making assignments first, directs the search toward the planning problem's goals. The *most-constrained* principle rates agents by the number of action landmarks they have accumulated. This principle adds a fail-first mechanism to the agent selection, where agents with many action landmarks will be more likely not to find a feasible plan, forcing them to backtrack. Combining the two principles, we get the *goal-achieving heuristic*, where ties are broken using the *most-constrained* principle.

In following our aim to make the search goal-oriented, *value ordering* has a large role to play. Because the agent locally plans first, when assigning an agent's *ActVar*, we are assured that the search sub-tree created by this choice of actions is oriented toward achieving the agent's goals. Therefore, *ActVar*'s value ordering heuristic will rely completely on the planning with action landmarks the agent performs. When the agent can supply goals that are public, the local planner would strive to find a plan satisfying as many public goals as it can. Once we've assigned *ActVar* and *TimeVar*, as well as generated *ReqVar*'s domain, we must choose a value for it. When assigning the agent's *ReqVar*, we follow the *least constraining value first* principle. Choosing the assignment that adds the fewest action landmarks to

unassigned agents is the option we found worked best, although empirically *ReqVar*'s value ordering heuristic didn't make a big difference performance-wise.

3.2.3 Reducing ActVar's Domain

An *essential action* a of a planning problem Π is an action that is taken in *every* plan for Π. In other words, without at least one execution of a, no solution can be found for Π. Finding certain essential actions for a planning problem can be done quite efficiently, as shown in [7]. For every action a, we use delete-relaxation planning (planning using actions obtained by ignoring delete effects) without a. If no solution is found, a is essential.

In a MA setting, this procedure must be altered since agents might want some of their actions to remain private. Every agent must find essential actions separately, with knowledge only of other agents' public actions. For every public action a, φ_i will perform relaxed planning *without* a as before, using all its actions (private and public), but using only the external projection of the public actions of other agents. Since an action of one agent could be essential for achieving another agent's private goals, agents could discover other agent's essential actions, and inform one another of such essential actions before the distributed search begins.

Since agents perform essential action discovery independently, this procedure can be performed *in parallel*, before the actual search begins. Once agents have their essential public actions, they perform a simple check to see whether the number of essential actions is not larger than δ. If it is, the *solve-csp* method immediately returns "no-solution", and δ is incremented. More importantly, when assigning *ActVar*, if an essential action is not included in the plan found, this plan is disregarded and the search for a different one continues. This way, many values of *ActVar* that satisfy the agent's internal goals, but do not contain actions that are essential to the entire system, are ignored. This reduces *ActVar*'s domain substantially. Because *ActVar* is assigned before the agent's other variables, this reduction effectively prunes large branches of the search tree, almost effortlessly, and has an essential role in the success of our algorithm.

3.2.4 Adapting the DisCSP Solver Algorithms

The *Planning First* search methodology requires adaptation and alteration of existing *DisCSP* solver algorithms. First, the assignment of local variables becomes a relatively complex operation. Second, variables' domains cannot be instantiated a priori, i.e., before the search process starts. For instance, *ReqVar*'s domain must be dynamically generated every time the agent assigns *ActVar* and *TimeVar*. Lastly, dynamically generated domains require dynamically generated constraints as well. In effect, the *DisCSP* solver now requires **smart agents**, capable of local planning and dynamic generation of domains and constraints. Following this, *DisCSP* solvers can no longer be used as a "black box". Moreover, some internal procedures of distributed search algorithms, such as *Forward Checking*, must also be modified to cope with complex variable assignment and dynamically changing domains.

As an example of the issues that arise when we want to adapt an existing *DisCSP* solver to our needs, we'll look at the Asynchronous Forward Checking (*AFC*) [8] algorithm. *AFC* keeps one partial assignment at all times. When an agent assigns its variable, it sends copies of the partial assignment to all unassigned agents. These agents now check if their variable has a value consistent with the partial assignment. If not, they inform

unassigned agents the partial assignment is NOGOOD. Performing this forward-checking process is simple when all domains and constraints are generated a priori, by checking every value in the variable's domain for consistency with the partial assignment. In our case, some decisions must be made regarding how forward-checking is performed. It is clear that if an agent cannot find a valid local plan given a partial assignment, then that assignment is NOGOOD. But what if a valid plan is found and *ActVar* is assigned? Do we go on to generate *TimeVar*'s and *ReqVar*'s domain and their constraints? This issue is important since this generation process is computationally expensive, especially since forward checking is performed many times during the search. It is clear that relying only on whether a valid local plan exists does not achieve full detection of NOGOOD partial assignments. This is because many times a local plan can be found but consistent execution times or requirements for it cannot. Here, we have a trade-off between the computational effort we invest in forward-checking, and the procedure's NOGOOD detection rate. Such questions must be answered for various backtracking techniques, such as *Back-Jumping, Back-Marking* and *Conflict-Based Back-Jumping* [9]. *Back-Jumping*, for example, requires every variable to keep track of the deepest variable conflicting with it, and backtracks to it when no consistent value can be found. Similarly to *forward-checking*, this is done easily when domains are generated a priori. In our case, however, we do not have the luxury of checking all values in our domain for consistency, since the variables' domains are never fully generated. Again, we must address the trade-off between the computational effort we invested in generating these domains, and the efficiency of the *back-jumping* mechanism. Reliance only on *ActVar*'s domain will in most cases effectively reduce *back-jumping* to simple chronological backtracking. On the other hand, generating *TimeVar*'s and *ReqVar*'s domains will, most of the times, require more effort than the jump will save.

In our implementation, we examined a number of variants of *forward-checking* and *back-jumping* (the two procedures required by AFC), and we found that the middle ground is the most effective. Specifically, we checked whether *ActVar* and *TimeVar* have non empty domains, but we ignored *ReqVar*'s domain.

3.2.5 *The Agent's Internal Planner*
Planning First also requires adaptation of the smart agent's local planner. The planner should strive to satisfy as many of the initial public goals as it can. This follows our aim of making the search as goal-oriented as possible, with the added bonus of minimizing the requirements from other agents, following the *least constraining value first* principle. This can be done by trying to solve the local planning problem where public sub-goals are encoded as private ones, and removing them one by one if the search fails. Another option is using rewards or negative costs so the planner prefers public-goal achieving plans. Private goals must always be satisfied since no other agent can supply them. Since some, if not most, of the generated plans will not lead to a solution for the entire system, the planner must store failed plans and not assign their action sequences again. Furthermore, the planner must check if a plan contains the agent's *essential actions* and that the number of public actions in the plan does not exceed δ. These useless plans should be ignored, and their corresponding nodes in the search tree should not be expanded further.

Action landmarks can be encoded into planning tasks given to the planner, and therefore require no special alteration of the planner.

This is done by adding $\delta + 1$ propositions, where landmark action i requires the ith proposition and supplies the $i + 1$th proposition. Propositions 1 and $\delta + 1$ are added to the initial and goal states respectively. This method, called *Planning with Action Landmarks* [2], ensures the ordered execution of the landmarks, allowing any number of internal actions between them.

3.2.6 *Putting It All Together*
Our smart agent will follow this high-level schema when receiving the current partial assignment (*CPA*):

procedure <u>Receive-CPA</u>(cpa)
   if(cpa is a backtrack-cpa) **then**
      if(*TimeVar* and *ReqVar* have non-empty domains) **then**
         assign local variables and send to next agent
         send unassigned agents forward check message
         return
   else
      reset local domains and forbidden plans
   loop
      if(Find-local-plan-with-landmarks(cpa)) **then**
         ActVar := extract public actions from plan
         generate *TimeVar*'s domain and assign
         generate *ReqVar*'s domain and assign
         if(*TimeVar*, *ReqVar* are assigned) **then**
            send assigned cpa to next agent
            send unassigned agents forward check message
            return
         else
            add plan to forbidden plans
      else
         perform modified back-jumping
         return
   endloop

4. EXPERIMENTAL EVALUATION
Our intuition now is that in planning domains where agents have little interaction, our algorithm can show scalability beyond centralized solvers. In order to verify this intuition, we ran experiments on several domains, which have varying levels of agent interaction. For our empirical study we compared the centralized *Fast-Forward* (*FF*) [3] solver with two versions of our algorithm. We decided to run the experiments on the standard IPC [12] domains. Designed for evaluation of monolithic, single-agent planning, many of these domains unfortunately do not well fit the framework of multi-agency. In particular, many of these domains (e.g., Gripper, Miconic, Grid, Blocksworld, etc.) are single-agent by their nature. Logistics, Rovers and Satellites were identified as the domains that allow for natural reformulation of their tasks in terms of MA-STRIPS. The logistics domain was used throughout the paper to illustrate our ideas. Rovers and Satellite, as formulated in the IPC, are single-agent domains, but they were originally motivated by real MA applications used by NASA. The Satellites domain requires planning and scheduling observation tasks between multiple satellites, each equipped with different imaging tools. The Rovers domain involves multiple rovers navigating a planet surface, finding samples and communicating them back to a Lander. These benchmark domains give us a nice view of *Planning-First*'s strengths and weaknesses. All experiments were run on an AMD Phenom 9550 2.2GHZ processor. Memory usage was limited to 2.5 GB.

When implementing *Planning-First*, we chose *Asynchronous Forward-Checking* as our "black-box" *DisCSP* solver algorithm. *FF* was used as the internal planner for our agents. Both algorithms were of course adapted to our needs as described in sections 3.2.4-5. As is common in *DisCSP* algorithm evaluation, distributed search is performed using a simulator, in which agents are simulated by threads which communicate only through message passing. This means that computation is actually not performed in parallel. Aside from *Planning-First*, we implemented another similar algorithm, which we refer to as *Planning-Last*. Here, the domains of the variables are generated a priori, and internal planning is performed as a unary constraint mechanism, filtering these domains. In essence, before search starts, the agent constructs the complete domains of its *ActVar*, *TimeVar* and *ReqVar*, with all possible actions sequences, time sequences and requirements respectively. When assigning ActVar, internal planning is used to check consistency with landmarks, filtering out inconsistent action sequences. Distributed search and internal planning were implemented exactly as in *Planning-First*.

Performance evaluation of *DisCSP* solver algorithms is usually done using two independent measures [10] – computational effort in the form of non-concurrent constraint checks performed, and communication load in the form of total number of messages passed. Performance of planning algorithms, however, is often measured by the runtime of the planner and by the number of nodes expanded. Since there is no obvious overlap between the two cases of performance evaluation, and because we compare our algorithm to *FF*, which is not a constraint based planner, we use running time as our main measure of efficiency.

We had two main issues on our agenda. First, verifying that *Planning-First* can show scalability beyond *FF* in certain loosely coupled domains. Second, getting an idea of how *DisCSP* heuristics, empowered with planning obtained knowledge, can affect the efficiency of the search. For each domain, we evaluated performance on a series of problem instances. The number of agents grew between two instances and with it the problem's size (propositions and executable actions). Table 1 depicts the running time (in sec.) of planners obtained over the three domains. The solution's δ is given in brackets. Empty entries in the table denote instances that were not solved due to exceeding the memory limit.

The results in Table 1 give us a clear picture of where *Planning-First* performs well. The Logistics domain is very tightly-coupled, where, on average, 64% of an agent's actions are public. A truck agent, for example, has only two or three private (mostly drive) actions, but has many public load/unload actions since the majority of locations are shared by more than one agent. This results in many agent coordination points, which increases δ (on average > 3), making *Planning-First* ineffective[1].

An agent in the Rovers domain has fewer public actions (46% on average), as well as a very complex internal planning problem, including navigation, soil and rock sampling, and image capturing. Its coordination points (average $\delta \approx 2.5$) include locations which are accessible to multiple Rovers, and communicating its data to the Lander. From the results we see that as the number of agents rises (and the problem's size increases

exponentially), *Planning-First's* agents are still solving relatively easy internal planning problems, while *FF* has difficulty in handling the size of the problem in its entirety. The Satellites domain has even less interaction between agents (15% of actions are public, $\delta \approx 1.5$), where coordination is limited to scheduling observation tasks shared by multiple satellites. The agent's internal planning includes positioning, calibration and image capturing. Here, as in the Rovers domain, *Planning-First* clearly shows scalability beyond *FF*, confirming our initial intuition. It should be noted that both algorithms found only optimal solutions with respect to the number of actions.

Table 1. Runtimes (sec.) of planners across the test domains.

Task no.	No. of Agents	Planning-First	Planning-Last	Fast-Forward
LOGISTICS				
1	3	0.3 (δ=3)	0.3	0.1
2	3	3.4 (3)	2.5	0.6
3	4	4.9 (3)	5.8	0.2
4	5	13.6 (3)	22.6	0.4
5	6	51.1 (4)	64	5
6	7	549 (4)	659.8	48.3
7	8			235.4
ROVERS				
1	3	4.2 (3)	5.9	34
2	4	34.3 (2)	37	328
3	5	122.6 (3)	205.3	1165.1
4	6	697 (2)		5245.2
5	7	2254 (2)		
SATELLITES				
1	2	0.1 (2)	0.2	0.1
2	4	0.2 (2)	0.5	0.3
3	6	0.6 (2)	30.4	1.4
4	8	1.5 (1)	60.7	6.5
5	10	4.7 (1)		42.1
6	12	10.9 (2)		153.2
7	14	20 (2)		521.9
8	16	99.5 (2)		1703.4
9	18	605.1 (1)		

Our results hint that the percentage of the public actions among an agent's actions is informative regarding the coupling level of a system, quite possibly because it is correlated with the number of interaction points required in a solution, as captured by δ. A high δ value therefore limits the scalability and effectiveness of *Planning-First*. As for *Planning-Last*, it is clearly not scalable due to its memory consuming exhaustive domain generation. Once either δ rises or there are many agents, the high number of possible coordination points results in monstrous domain sizes that are out of any *DisCSP* solver's reach.

The results depicted in Figure 1 compare the widely used *DisCSP* variable ordering heuristic, *Minimal-Domain* (guided by the fail-first principle) with our *goal-achieving most-constrained* heuristics. Here, we use classical *DisCSP* solver evaluation measurements – NCCCs and messages passed. The values are averages over the three domains experimented on.

These results make it clear that our heuristic overwhelmingly dominates *Minimal-Domain*. Although it is almost ideal when it comes to randomly generated, structure-less problems, *Minimal-*

---

[1] It should be noted that this is mostly a reflection of the actual logistics problem as formulated in the IPC. Here, the local state of each agent is quite limited, and the picture could change if each agent had more complex internal route planning problems.

Domain's "blindness" to planning information gives our heuristic a clear advantage. Prioritizing goal achieving agents (regardless of domain size) is crucial for *DisCSP*s with such large domains.
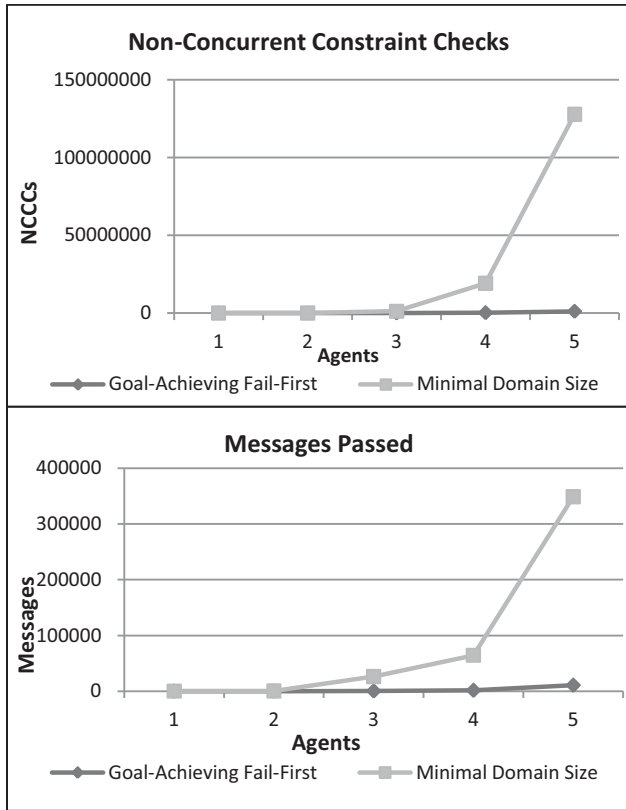

Figure 1. How ordering the agents by their ability to achieve goals affects *Planning-First*'s computational effort and communication load.

5. CONCLUSIONS AND FUTURE WORK

We have shown that BD's *Planning as CSP+Planning* methodology can be extended into a practical, fully distributed planner. The powerful theoretic guarantees of this methodology can, with correct utilization of planning tools and techniques, be translated into an efficient algorithm. This required: decreasing domain sizes by careful formulation of the variables in the CSP and by using domain-specific pruning techniques, such as filtering plans that do not contain essential actions; and adaptation of classical techniques and heuristics in order to take into account problem structure. Our approach yields promising results even when implemented on a centralized machine, provides novel distributed planning techniques that are competitive with state-of-the-art classical planning techniques, and supplies a nice example of how current *DisCSP* techniques can be enhanced to take into account the structure of a particular problem domain. Our work also introduces the world of MA-Planning to *DisCSP* researchers, presenting non-random, structured and well motivated domains for testing their algorithms and heuristics.

Our work is also of general interest to the distributed AI community since it provides a nice proof of the validity of some of the original ideas that motivated work in this area. That initial work was motivated not only by the desire to deal with systems consisting of multiple real agents, but also by the belief that

distribution of tasks, modularity of design, and specialization, are powerful algorithmic tools for building complex, efficient agents. In the realm of system design, the empirical evidence we have shown on the utility of reducing the percentage of public actions could help design MA systems that are easier to plan for, by defining clear separation of capabilities between agents. Of course, other considerations, such as robustness could call for some amount of duplication, but the two aims are not necessarily contradictory, as it may be possible to modify the planning algorithms to make smart use of duplication (e.g., by prioritizing).

Future work should check ways to systematically generate plans, avoiding the need to save previously tried action assignments, and find methods for handling problem instances with larger δ values.

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In AAAI, 809–814.

[2] Brafman, R. I., and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In Proc. of the 18th ICAPS, 28-35.

[3] Hoffman, J. and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. In Journal of Artificial Intelligence Research, Volume 14, 253 - 302.

[4] Fikes, R. Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. In Proc. of the 2nd International Joint Conference on Artificial Intelligence, 608-620.

[5] Yokoo, M., Durfee, E., Ishida, T., Kuwabara, K. 1998. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. In IEEE Trans. on Data and Kn. Eng., 10(5), 673-685.

[6] Haralick, R.M., Elliott, G.L. 1980. Increasing tree search efficiency for constraint satisfaction problems. In Artificial Intelligence 14, 263–314.

[7] Zhu, L., Givan, R. 2004. Heuristic Planning via Roadmap Deduction. In IPC-4, 64-66.

[8] Zivan, R. and Meisels, A. 2007. Asynchronous Forward-checking for DisCSPs. In Constraints, 12, 131-150.

[9] Prosser, P. 1993. Hybrid Algorithms for the Constraint Satisfaction Problem. In Computational Intelligence, Volume 9, Number 3, 268-299.

[10] Meisels, A., et. al. 2002. Comparing performance of Distributed Constraints Processing Algorithms. In Proc. DCR Workshop, AAMAS, 86-93.

[11] Kishimoto, A., Fukunaga,A. and Botea, A. 2009. Scalable, Parallel Best-First Search for Optimal Sequential Planning. In Proc. of the 19th ICAPS, 201-208.

[12] The international Planning Competition, ICAPS, http://ipc.informatik.uni-freiburg.de/