A Distributed CSP Approach for Solving Multi-agent Planning Problems

Oscar Sapena, Eva Onaindía, Antonio Garrido, Marlene Arangu
Dpto. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Valencia, Spain
osapena,onaindia,agarridot,marangu@dsic.upv.es

Abstract

Distributed or multi-agent planning extends classical AI planning to domains where several agents can plan and act together. There exist many recent developments in this discipline that range over different approaches for distributed planning algorithms, distributed plan execution processes or communication protocols among agents. One of the key issues about distributed planning is that it is the most appropriate way to tackle certain kind of planning problems, specially those where a centralized solving is infeasible. In this paper we present a new planning framework aimed at solving planning problems in inherently distributed domains where agents have a collection of private data which cannot share with other agents. However, collaboration is required since agents are unable to accomplish its own tasks alone or, at least, can accomplish its tasks better when working with others. Our proposal motivates a new planning scheme based on a distributed search of heuristic information and on a constraint programming resolution process.

Introduction

Distributed planning is the problem of finding a course of actions that will help a set of agents collectively satisfy certain desired goals. Due to an inherent distribution of resources such as knowledge and capability among the agents, an agent in a distributed planning system is unable to accomplish its own tasks alone, or at least can accomplish its tasks better when working with others (Durfee 2001). Distributed planning is still an open challenge, and there is an increasingly number of applications that can benefit from this research area: cooperative robotics (Wehowsky, Block, & Williams 2005) (Sirin *et al.* 2004), composition of semantic web services (Wu et al. 2003), manufacturing systems (Hahndel, Fuchs, & Levi 1996), etc.

The literature cites many reasons for which multi-agent planning is an interesting approach to pursue. One of these reasons is to split the problem into smaller subproblems which are usually easier to solve. This divide-and-conquer approach has been used in several distributed planning proposals (Rehak, Pechoucek, & Volf 2006) (Cox, Durfee, & Bartold 2005). In these approaches, multiple agents plan to achieve their individual goals independently and, then, these individual plans are merged into a global plan. The coordination/merging process is usually the most costly part since it is necessary to avoid cross-working or duplicating effort.

Another reason that often comes up is that of privacy (van der Krogt 2007). Especially in circumstances where the agents represent companies, sharing data with other parties is considered undesirable. At the same time, it is well recognized that cooperation may be mutually beneficial to all parties. In this paper we present a new planning framework aimed at solving problems of this type. Specifically, we will address problems with the following characteristics:

- *Distributed domains*. In these domains there exists an inherent distribution of resources such as knowledge and capability among the agents. This way, agents are clearly identified, so a problem decomposition stage is not required.
- *Privacy*. Agents maintain a set of private data, which are the beliefs that the agent will never share with other agents. The goals of an agent are also private, although it may require help from other agents to achieve them.
- *Collaboration*. In this framework, an agent often needs help from other agents to achieve the necessary conditions for executing an action. However, even though actions are jointly planned, they are individually executed, that is, agents do not get synchronized to carry out a same action jointly (for example, several robots pushing a single block together into a target area). This latest type of coordination is usually addressed in team-oriented planning approaches, where several agents collaborate to achieve a common global goal.

The presented approach is useful to solve planning problems in inherently distributed domains where a centralized solving process is not affordable. Additionally, agents have a collection of private data which cannot share with other agents so information exchange among them can only be achieved through the public or sharable data. Our proposal motivates a new planning scheme based on a distributed search of heuristic information and on a constraint programming resolution process. The overall approach is a distributed *CSP* resolution for solving the kind of problems that fit well a multi-agent planning paradigm.

The remainder of this paper is structured as follows. We begin by defining the problem characteristics and showing a representative problem example. Then, we present a general overview of our approach and we describe the main two stages of the planning algorithm. Finally, we show some experimental results and we present our conclusions and future work.

Problem definition

Our approach is particularly aimed at solving problems which develop in inherently distributed domains. In this type of problems, agents are clearly identified so it is not necessary to apply a problem decomposition because there exists a natural partition/distribution of the problem itself.

Definition 1. An *agent* is an entity with planning capabilities and thus we can specify an agent as a tuple $Ag = \langle Adj$, G, I, A, $m\rangle$ where:

- Adj is the set of adjacent agents. An agent ag' is adjacent to an agent ag, $ag' \in Adj(ag)$, if the public information of ag' is accessible from ag. This relationship is symmetric: $ag' \in Adj(ag) \iff ag \in Adj(ag')$. An agent can only collaborate directly with its adjacent agents, consulting and/or modifying their shared information.
- G represent the individual goals of the agent. These goals are not visible from other agents.
- I is a set of propositions that represents the agent's beliefs. This knowledge is classified in private (non-sharable) and public (sharable): $I = \langle Ip, Is\rangle$. The private information, Ip, is a set of propositions that are not accessible from other agents. On the contrary, interactions between agents are possible through their public repository: other agents can consult and add propositions to Is.
- A is the set of actions that can be applied in the domain. This information, which represents the agent skills, is not accesible from other agents. As a typical planning action, an action a is a triple $\langle pre(a), add(a), del(a)\rangle$, where the preconditions, $pre(a)$ and the effects, $add(a)$ and $del(a)$, are sets of propositions. These propositions can be agent beliefs (public or private) or public beliefs from adjacent agents:
  $$\forall p \in pre(a) \cup add(a) \cup del(a) \,/\, a \in A(ag),$$
  $$p \in I(ag) \vee p \in Is(ag') : ag' \in Adj(ag)$$
- m is the optimization function (or metric). The agent must try to achieve its goals with the minimum cost according to this function.

Definition 2. A *planning problem* consists of finding a (partially ordered) sequence of actions that leads the system from its initial state to a goal state. Formally, it is defined as a tuple $\langle Ag, I, G, A\rangle$, where:

- Ag is a set of agents.
- I is the problem initial state, which is the union of the agents' beliefs:
  $$I = \bigcup_{\forall ag \in Ag} I(ag)$$
  This initial state is globally consistent because information is not replicated in different agents:
  $$\forall ag, ag' \in Ag,\ I(ag) \cap I(ag') = \emptyset$$
- G is the problem goal, which is the union of the individual goals of all agents:
  $$G = \bigcup_{\forall ag \in Ag} G(ag)$$
- A is the problem actions, which is the union of the individual actions of all agents:
  $$A = \bigcup_{\forall ag \in Ag} A(ag)$$

Definition 3. A *global plan* is a partially ordered set of pairs (ag, a), where ag is an agent and a is an action of that agent ($a \in A(ag)$). The execution of a by ag causes a transition in the ag's state and, possibly, in the public beliefs of adjacent agents. Thus, a global plan is *valid* if the execution of all actions in the plan (by their corresponding agents and respecting the ordering constraints) on the initial state leads to a final state where all problem goals hold.

The cost of a global plan is computed as the sum of the cost of all actions in the plan, according to the metric function defined in their corresponding agent. Therefore, metric functions of all agents must be defined by the same measure unit and the same scale. This simplification facilitates the computation of the plan quality, but it may not be adequate for certain problems where the optimization function of one agent gets into conflict with the objective of another agent. This problem will be addressed in a future work to improve the applicability of our problem model.

There are many real problems that fit the distributed problem paradigm we have described in this section. In the following subsection, we show a simple example that we will use through the rest of the paper to illustrate our proposal.

Problem example

In this section, we show a simple example from a transportation and storage problem. In this problem we can identify two types of agents: warehouses and transport companies.

In each warehouse, there is a storage area and a loading area. In the storage area, packages can be stacked and unstacked onto a fixed set of pallets by means of a hoist. All the information about the storage area (packages, stacks, ...) in a warehouse is private, so it is not visible from other agents. Packages can also be moved from/to the loading area. The loading area of a warehouse is public, so packages in that area are visible from the adjacent agents of the warehouse. In general, the goal of a warehouse is to get a certain package distribution inside its storage area, and the metric function is to minimize the number of stack, unstack and move operations carried out to achieve its goal.

A transport company manages the transportation of packages between a set of warehouses located in the same geographic area. All the information about the truck fleet and the set of routes they use is private. Transport companies access the loading area of the warehouses to pick up and deliver packages. The optimization function in these companies is to minimize the total number of truck movements.

There are three agents in our example: two warehouse agents, *w1* and *w2*, and a transportation agent *t1*. In the initial state, agent *w1* has tree packages (*a*, *b* and *c*) and agent *w2* has two (*e* and *d*), organized as Figure 1 shows. The

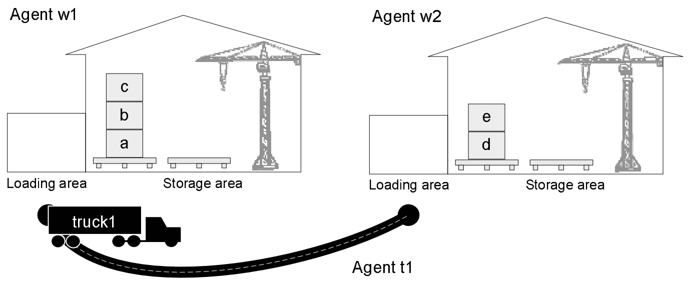

Figure 1: Initial state in the distributed planning example.

transportation agent has only one truck (*truck1*) and links both warehouses. Warehouse agents are only adjacent to the transportation agent, so warehouses cannot communicate information among themselves.

We use the following predicates in this example:

- `on ?pkg1 ?pkg2`, which states that package *pkg1* in on package *pkg2*.
- `at-la ?pkg`, which indicates that package *pkg* is in the loading-area of the warehouse.

The goal of agent *w1* is to get the package *b* on top of package *e*, that is, `on b e`. The position of the remaining packages in the warehouse is not relevant. The goals of agent *w2* are `on d a` and `on a c`. Agent *t1* has no individual goals.

The multi-agent planning approach that we propose in this paper can help agents in situations such as described in this example. It offers a way to co-operate while being in control of which information is shared and with whom.

Distributed planning scheme

Collaboration is required as agents usually cannot reach their individual goals without the help of the other agents. This help is provided through a set of abstract operations which we call *services*.

Definition 4. A *service* is an abstract operation that an agent offers to another agent to fulfill a set of public propositions, which we call the *service goal*. Internally, a service is a (partially ordered) set of actions (local plan), but how an agent provides a service is kept as private information.

Additionally, we define an *internal service* as an abstract operation that an agent computes for achieving its individual goals (in this case, the propositions in the service goal can be private). From now on, when we use the term service, we will refer to both services and internal services.

For providing a service, an agent may require the execution of one or more services from other agents. This way, the global planning problem consists of finding a partially ordered set of services that allows all agents to achieve their individual goals. The distributed planning process is started by the agents with individual goals to accomplish. The first step for these agents is to find out the pieces of information, which we call *requirements*, that they require from other agents in order to achieve their goals.

Definition 5. A *requirement* is a public proposition or a disjunction of public propositions that an agent needs from other agents to achieve its goals or to provide a service.

This way, a service can be seen as a planning operator, where the service requirements correspond to the operator preconditions and where the service goal corresponds to the operator effects.

In the proposed example, agent *w2* has to stack package *d* on top of *a* and package *a* on top of *c*, but packages *a* and *c* are not in the warehouse. Therefore, the only way to achieve its goals is to get packages *a* and *c* in its loading area. Then, the requirements are `at-la a` and `at-la c`. In the proposed example, requirements are always single propositions, but in other problems they can be disjunction of propositions. If, for instance, warehouse *w2* had two different loading areas ($la1$ and $la2$), then the requirements for reaching its goals would be: (`at-la1 a` $\vee$ `at-la2 a`) and (`at-la1 c` $\vee$ `at-la2 c`).

Agents with requirements to fulfill have to ask their adjacent agents for help. The first stage of the planning algorithm is a message exchange process to find out what services agents can provide and for computing an estimated cost of these services. Once the set of available services has been established, the global problem is solved through a collaborative planning process, which is modeled as a distributed *CSP*. Both planning stages are described in detail in the following two sections.

Cost estimate of the services

In this stage, agents send messages to their adjacent agents to request services for satisfying their requirements. An agent that needs a proposition sends a service request message to its adjacent agents asking for the cost of that service. Only one proposition is requested in a single message. If a requirement contains disjunctive propositions, then these propositions are individually requested in separated messages. These service request messages contain the following information:

- a_{orig}: the requesting agent.
- a_{dst}: the target agent.
- p: the requested proposition.
- R: the message route, which is the sequence of agents the message has passed by (the target agent is not included). This information avoids infinite message loops.
- Id: the message identifier. Each time a message is propagated, a number is added at the end of the Id. This number is the same for all propositions in a requirement (that is, for disjunctive propositions) but different for propositions in different requirements.

Let's suppose that an agent has the following requirements: ($p1 \vee p2$) and $p3$. The message identifier, for example, will be "1" for requesting $p1$ and $p2$ (in separated messages), and "2" for requesting $p3$. Through the message identifier and the message route parameters, an agent that receives several messages can easily find out if all the requested propositions are required or if some of them are disjunctive, that is, alternative ways to satisfy a requirement.

When an agent that receives a service request cannot provide the service, it returns an infinite cost as a reply. Otherwise, the agent must:

- Analyze the necessary requirements to provide the service.
- Ask its adjacent agents for help if required.
- If the requirements can be achieved, compute a plan to estimate the cost of the service. This cost is sent back as reply to the service request.

Additionally, if an agent receives two single different messages with non-disjunctive requested propositions, it automatically creates a new service for achieving both propositions together. This combination of propositions is done because achieving several proposition altogether is usually less costly than handling each subgoal independently and thus it will positively affect the plan quality.

However, since it is not affordable to compute all possible combinations, we will only consider the number of 2-combinations from a set with n non-disjunctive propositions $(n*(n-1)/2)$ plus one k-combination for $k > 2$. This calculation makes about $n^2/2$ number of combinations to study; this relaxation provides a good trade-off between computational cost and quality.

This message exchange process requires the agents to keep some information about the received messages and the requested and provided services. For each agent, the stored information is the following:

- Messages database (*MsgDB*): in this database the agent stores the received service request messages (the format of these messages was described above).
- Service database (*SerDB*): it stores the offered services. Tuples are in the form $\langle G$, a_{orig}, $NecReq$, $MinReq$, $minReqCost$, $Plan$, $cost\rangle$, where:
  - G is the service goal, that is, the conjunction of propositions the service achieves.
  - a_{orig} is the agent that requested the service.
  - $NecReq$ is the set of requirements needed to provide the service.
  - $MinReq$ is a conjunction of propositions that satisfy the service requirements ($NecReq$) with the minimum cost. For each requirement with disjunctive propositions, the alternative with the minimum cost is selected.
  - $minReqCost$ is the estimated cost to achieve $MinReq$.
  - $Plan$ is the internal plan that allows to achieve G, assuming that the requirements hold.
  - *cost* is the estimated service cost, which corresponds to the $Plan$ cost, computed according the metric function of the agent.
- Requirements database (*ReqDB*): this database stores the service request replies, that is, the cost of the services requested to other agents. This information is stored in tuples of the form $\langle G, cost, a_{dst}\rangle$, where:
  - G is the service goal.
  - $cost$ is the service cost, that is, the estimated cost to achieve the propositions in G.
  - a_{dst} is the agent that provides this service.

Algorithm in Figure 2 shows the behaviour of agent a_{dst} when it receives a service request from agent a_{orig}. In this algorithm, we have used the following functions:

- *Unreachable*(p): returns *true* if no action allows the agent to achieve proposition p, regardless of the preconditions of that action holds or not.
- *ComputeNecessaryRequirements*(p): this function returns the necessary requirements to achieve p.
- *RequirementsAchieved*($NecReq$): returns *true* if $NecReq$ can be satisfied.
- *ComputeMinimumRequirements*($NecReq$): returns the set of propositions that satisfies $NecReq$ with the minimum cost.
- *ComputePlan*(S, G): returns a plan to achieve the set of propositions G from the initial state S.
- *PlanCost*($Plan$): returns the cost of $Plan$ according to the defined problem metric.
- *Conjunctive*($p1$, $p2$): returns *true* if both propositions are conjunctive. This can be easily computed through the route and the identifier of their respective messages.
- *ServiceCombination*(G): this method computes the cost of a service that achieves all propositions in G. Therefore, the minimum requirements and a new plan to achieve G must be computed. If this service is less costly than achieving all propositions in G separately, then a message containing this information is delivered to the requesting agents.

At the end of this process, each agent has a list of services with an estimated cost for each one of them. Since the purpose of these services is to help other agents achieve their requirements, each agent must compute a set of internal services to achieve its own goals. Agent $w1$, for example, will compute an internal service for satisfying its goal '*on b e*'. Combinations with other services are also studied, in the same way as shown in Figure 2.

Tables 1, 2 and 3 show the services offered by agents *w1*, *w2* and *t1* respectively. The first column assigns a number to each service. The second column shows the service goal: each proposition is preceded by the requesting agent. The third column indicates the service cost, without taking into account the cost of the requirements. The last column shows the necessary requirements to provide the service: each proposition is preceded by the agent that can achieve it with the minimum cost.

Collaborative planning

In the literature we can find some proposals for solving collaborative planning tasks (Cox, Durfee, & Bartold 2005) (Rehak, Pechoucek, & Volf 2006). These works address

```
double ServiceRequest ($a_{orig}$, $a_{dst}$, $p$, $R$, $Id$) begin
  // Check if this service was previously computed
  if $\exists\, T \in SerDB$ / $T$.G = $\{p\}$ then
    return $T.cost + T.minReqCost$
  // Necessary requirements computation
  if Unreachable($p$) return $\infty$
  $NecReq \longleftarrow$ ComputeNecessaryRequirements($p$)
  // Ask for help to achieve the new requirements
  $\forall\, a \in$ Adj($a_{dst}$) $\wedge\, a \notin R$ do
    $\forall\, r \in NecReq$ /
    $\nexists\, T \in ReqDB : T.G = \{r\} \wedge T.a_{dst} = a$ do
      cost $\longleftarrow$ ServiceRequest($a_{dst}$, $a$, $r$, $R \otimes a_{dst}$,
                $Id \otimes newIdPart$)
      if cost $\neq \infty$ then
        $ReqDB = ReqDB \cup \langle\{r\}, cost, a\rangle$
  // Service cost
  if $\neg$ RequirementsAchieved($NecReq$) return $\infty$
  $MinReq \longleftarrow$ ComputeMinimumRequirements($NecReq$)
  $minReqCost = \sum_{\forall r \in MinReq}$ min($T$.cost),
                $\forall\, T \in ReqDB$ / $r \in T.G$
  $Plan \longleftarrow$ ComputePlan($CurrentState \cup MinReq$, $\{p\}$)
  $planCost \longleftarrow$ PlanCost($Plan$)
  $SerDB = SerDB \cup \langle\{p\}, a_{orig}, NecReq, MinReq,$
            $minReqCost, Plan, planCost\rangle$
  // Combination of services
  $C = \{p\}$
  $\forall\, T \in MsgDB$ / Conjunctive($T.p$, $p$) do
    $C = C \cup \{T.p\}$
    ServiceCombination($\{T.p\} \cup \{p\}$)
  if $|C| > 2$ then ServiceCombination($C$)
  return $planCost + minReqCost$
```

Figure 2: Service request processing.

a different problem than ours, since they follow a divide-and-conquer approach, but the key idea is the same: using *POP* (*Partial Order Planning*) techniques (Penberthy & Weld 1992). *POP* techniques are very appropriate for distributed planning since no explicit global state is required.

Instead of developing a new distributed *POP* algorithm, we have chosen to convert the planning problem into a distributed constraint satisfaction problem (*disCSP*). The reasons for this decision are:

- There are many distributed *CSP* algorithms (Yokoo & Hirayama 2000) and some available platforms, such as *DisChoco* [1].
- A planning problem can be easily formulated as a *CSP* and experimental results show that this approach can be very competitive (Vidal & Geffner 2006).

The goal of this process is to obtain a final global plan, which is a partially ordered list of services that each agent will have to carry out. In order to establish the order between these services, each service will have an associated starting time. The steps for obtaining this global plan are described in the following subsections.

[1] Available at *http://www.lirmm.fr/coconut/dischoco/*

Table 1: List of services of agent *w1*.

#	Service goal	Cost	Requirements
1	$\{t1$:at-la a$\}$	6	$\emptyset$
2	$\{t1$:at-la c$\}$	2	$\emptyset$
3	$\{t1$:at-la a, $t1$:at-la c$\}$	6	$\emptyset$
4	$\{w1$:on b e$\}$	6	$\{t1$:at-la e$\}$
5	$\{t1$:at-la a, $w1$:on b e$\}$	8	$\{t1$:at-la e$\}$
6	$\{t1$:at-la c, $w1$:on b e$\}$	6	$\{t1$:at-la e$\}$
7	$\{t1$:at-la a, $t1$:at-la c, $w1$:on b e$\}$	8	$\{t1$:at-la e$\}$

Table 2: List of services of agent *w2*.

#	Service goal	Cost	Requirements
1	$\{t1$:at-la e$\}$	2	$\emptyset$
2	$\{w2$:on d a$\}$	6	$\{t1$:at-la a$\}$
3	$\{t1$:at-la e, $w2$:on d a$\}$	6	$\{t1$:at-la a$\}$
4	$\{w2$:on a c$\}$	4	$\{t1$:at-la a, $t1$:at-la c$\}$
5	$\{w2$:on a c, $w2$:on d a$\}$	8	$\{t1$:at-la a, $t1$:at-la c$\}$
6	$\{t1$:at-la e, $w2$:on a c$\}$	6	$\{t1$:at-la a, $t1$:at-la c$\}$
7	$\{t1$:at-la e, $w2$:on a c, $w2$:on d a$\}$	8	$\{t1$:at-la a, $t1$:at-la c$\}$

Table 3: List of services of agent *t1*.

#	Service goal	Cost	Requirements
1	$\{w1$:at-la e$\}$	2	$\{w2$:at-la e$\}$
2	$\{w2$:at-la a$\}$	1	$\{w1$:at-la a$\}$
3	$\{w1$:at-la e, $w2$:at-la a$\}$	2	$\{w2$:at-la e, $w1$:at-la a$\}$
4	$\{w2$:at-la c$\}$	1	$\{w1$:at-la c$\}$
5	$\{w1$:at-la e, $w2$:at-la c$\}$	2	$\{w2$:at-la e, $w1$:at-la c$\}$
6	$\{w2$:at-la a, $w2$:at-la c	1	$\{w1$:at-la a, $w1$:at-la c$\}$
7	$\{w1$:at-la e, $w2$:at-la a, $w2$:at-la c$\}$	2	$\{w2$:at-la e, $w1$:at-la a, $w1$:at-la c$\}$

Selecting the agent priority

In this type of decentralized algorithms is necessary to establish an order/priority between the participating agents. Moreover, this order substantially affects the performance of the search.

Experimentally, we have observed that the most efficient order assignment is to set the highest priority to the agent with the lower number of services (internal services are not considered). In the proposed example, $w2$ is the agent with highest priority since it only has one non-internal service: the service #1. The rest of services are internal.

The next agent in the priority order is the agent that provides a higher number of services to agents with an already assigned priority, an so on. In the proposed example, $t1$ is the agent that provides more services to $w2$. Finally, $w1$ will be the agent with the lowest priority.

The reasons for these ordering criteria are the following:

- The agents with a higher number of services and, consequently, with a greater number of variables and constraints, have low priorities. This way, the agents that have to make a greater computation effort are the last in the *CSP* solving process, thus minimizing the number of required backtracks.
- Following an assignment order according to the services' causal links, we maximize the number of shared variables and constraints between two consecutive agents. Then, when an agent communicates its partial solution to the next agent, it is possible to prune the domains of the variables efficiently.

Formulating the problem as a *CSP*

Each service can be easily translated into a *PDDL* operator: the requirements correspond to the operator's preconditions and to the delete effects (if they do not hold after the service execution), the service goals correspond to the add effects, and the service cost can be modeled as a numeric fluent. For example, the third service of agent $t1$ (see Table 3) is translated as follows:

```
(:action t1-Service3
  :parameters()
  :precondition (and (w2-at-la e)
                     (w1-at-la a))
  :effect (and (not (w2-at-la e))
               (not (w1-at-la a))
               (w1-at-la e) (w2-at-la a)
               (increase (cost) 2)))
```

Based on the works of (Vidal & Geffner 2006) and (Refanidis 2005), we can translate these planning operators/services into a *CSP* formulation. For this formulation, we have defined the following integer variables:

- $Inplan(o) \in [0,1]$, represents wether the operator o is in the final plan (value 1) or not (value 0).
- $Start(o) \in [0,\infty]$, represents the start time of o.
- $Support(p, o) \in O$, where O is the set of operators that can provide proposition p for o. Thus, these variables represent the causal links between the agent services.
- $Time(p, o) \in [0,\infty]$, is the time when the causal link $Support(p, o)$ happens.

If an operator o is included in the plan ($Inplan(o) = 1$), then the following constraints must hold:

- The operator that produces p for o must be in the plan: $Support(p, o) = o' \longrightarrow Inplan(o') = 1$.
- Preconditions of o must hold before its start time: $Time(p, o) \leq Start(o)$.
- The duration of an operator is one time unit (we are not working with durative actions), so its effects are achieved one unit of time after the operator's start: $Support(p, o) = o' \longrightarrow Time(p, o) = Start(o') + 1$.
- *Mutex* relationship between an operator o that requires p and other operator o' in the plan that deletes p: $Start(o) \neq Start(o') + 1$.
- Threat resolution by promotion or demotion when an operator o' in the plan deletes a proposition p that is required by o: $(Start(o') + 1 < Time(p, o)) \vee (Start(o) < Start(o'))$.

Each agent is in charge of dealing with the variables and constraints that are related to its own services. However, there are some variables and constraints that must be shared between two agents. This occurs when the results of a service provided by an agent are required by other service of another agent. In this case, the agent with the highest priority will assign a value to the shared variables, whereas the other agent will check the shared constraints.

Heuristics

The formulation of a planning problem as a *CSP* requires the definition of a great number of variables and constraints. For instance, there are 227 variables and 781 constraints in the proposed example (57, 86 and 84 variables and 182, 297 and 302 constraints for agents $w2$, $t1$ and $w1$, respectively). In order to improve the efficiency of the *CSP* solver, we have defined some additional constraints and a specific value selection heuristic for the $Inplan$ variables.

One reason for the problem complexity is the high number of services that arise from the requirement combinations: if two conjunctive propositions, $p1$ and $p2$, have been requested to an agent, that agent automatically computes a new service that jointly achieves $p1$ and $p2$. Evidently, if the service to achieve p is included in the plan, the service to achieve $p1 \wedge p2$ will not be included (and the other way around), since both services represent alternative ways to attain p. To model this fact in the *CSP* formulation, we have included the following constraints: if two operators in an agent, o and o', produce a proposition p, then both operators are mutually exclusive in the plan: $(InPlan(o) = 1 \longrightarrow InPlan(o') = 0) \wedge (InPlan(o') = 1 \longrightarrow InPlan(o) = 0)$. These additional constraints substantially improve the solving process performance.

The most costly part for the *CSP* solver is to find out what operators must be included in the plan, that is, to make a feasible value assignment for the $Inplan$ variables. For this reason, each agent tries to make a good initial value assignment for these variables:

- A set with the requested propositions is computed for each adjacent agent. The agent itself is also considered as an adjacent agent as internal services can be seen as self-requests.

  For agent $w2$ in the proposed example, these sets are the following:

  Requests of agent $t1$: {`at-la e`}

  Self-requests: {`on d a, on a c`}
- A set of services/operators is computed to satisfy each set of requested propositions with the minimum cost.

  Following the example, these sets of services for agent $w2$ are the following:

Table 4: Final plan for the proposed example.

Time	Agent	Service	Description
0	$w1$	#3	Move packages a and c to the loading area
0	$w2$	#1	Move e to the loading area
1	$t1$	#7	Transport a and c to $w2$ and e to $w1$
2	$w2$	#5	Stack a on c and d on a
2	$w1$	#4	Stack b on e

Services for the requests of agent $t1$: {service #1}

Services for the self-requests: {service #5}

The $Inplan$ variables for the operators corresponding to these services (see Table 2) are initially set to 1, and the rest of operators are set to 0.

This value selection heuristic increases the solving process performance outstandingly, above all when the selected operators are finally included in the plan.

Distributed solving process

In the literature we can find several distributed *CSP* algorithms: asynchronous backtracking (*ABT*), asynchronous weak-commitment search (*AWC*), distributed breakout, etc. (Yokoo & Hirayama 2000). However, we are not interested at present in solving the problem the most efficiently as possible, but in demonstrating that out approach is viable for solving this type of problems. For this reason, we have implemented a simple sequential forward checking algorithm. As a future work, we want to use a more sophisticated algorithm in order to decrease the number of messages that the agents need to exchange during the search.

Plan execution

At the end of the process each agent knows what services has to execute and at what time. Table 4, for instance, shows the final plan obtained for the proposed example. This way, the plan execution becomes an easy task.

However, the final plan might not be completely executable in certain cases. This is because the computation of the estimate cost of the services is based on the problem initial state, which changes through the plan execution. Therefore, it is possible that some services that were initially available cannot be executed later (after the execution of other services).

If an agent must execute a service according to the global plan, but it is not possible to achieve the goals of that service in the current state, then it is necessary to calculate a new global plan: the planning process is repeated again but starting from the current situation. This solution does not avoid the problem of dead-ends, which may appear in some non-reversible domain. This issue will be addressed in a future work.

Table 5: Obtained results for an increasing number of agents.

#Ag	#Serv	#Vars	#Constr	#Msg	#Time
3	21	149	781	872	0.22
4	28	198	805	8641	0.73
5	29	207	831	844	0.41
6	33	233	954	15263	2.86

Results

For the evaluation of our approach, we have defined four problem examples with an increasing number of agents. The first problem, with three agents, corresponds to the proposed problem example (see Figure 1). The second problem adds a new warehouse, $w3$, with an individual goal to achieve. The third problem adds a new transport company, $t2$. The last problem, with six agents, includes a new warehouse, $w4$ with another individual goal to solve.

The obtained results are displayed in Table 5. The first column indicates the number of agents in the problem. The second column shows the total number of services provided by the agents. The third and the fourth column show the total number of variables and constraints in the *CSP* formulation of the problem, respectively. The fifth column shows the number of messages exchanged during the *CSP* solving process. The last column shows the time in seconds for obtaining the first solution. The distributed *CSP* solver has been executed on a single computer (Pentium 4 - 3Ghz.), so we have not considered message delays for the agents communication.

The goal of this evaluation is not to show a broad range of benchmark results (since the whole process is not fully automated yet), but to demonstrate that our approach is valid for solving this type of problems. In spite of the current implementation can be improved in many ways, the low number of messages exchanged and the small computation times indicate that our approach can be successfully used for solving small and medium multi-agent planning problems.

Thanks to the value selection heuristics, the plan quality obtained in the first solution found is usually very good. In the proposed example, for instance, the first solution found is the optimal one. In the current implementation of the *CSP* solver, the search is stopped when the first plan is found. However, this behaviour can be easily changed to allow the search to continue until a deadline is reached or even until the optimal solution is found.

Conclusions and future work

In this paper, we have presented a new approach for multi-agent planning, based on the extraction of heuristic information and the problem formulation as a *CSP*. Unlike other existing proposals that follow a divide-and-conquer approach, we focus our work on inherently distributed problems in which agents are clearly identified. In this type of problems, privacy is usually a key issue: agents keep private information that they do not want to share with other agents. At the same time, some information must be shared to allow the cooperation between the agents, which is required to reach their goals.

In our approach, the collaboration and the privacy between agents is achieved through the definition of services, which are abstract operations provided by the agents. This way, the planning process consists of two sequential steps:

- A message exchange process where agents request the information that they need to other agents and where the set of available services is obtained. The cost of each service is used as heuristic information for the search process.
- The problem is formulated as a distributed constraint satisfaction problem and solved with *CSP* techniques.

Some preliminary results show that our approach can be successfully used for solving small and medium size problems. Moreover, the quality of the solution is usually very good.

As a future work, we want to fully automatize the planning process, using a multi-agent platform such as, for example, *JADE* (Bellifemine, Caire, & Greenwood 2007). This will allow us to test our algorithm in a wide range of benchmark domains. On the other hand, there are some improvements that can be introduced in the algorithm, such as the use a more efficient distributed *CSP* solving method and the computation of new heuristics for the variable and the value selection. Finally, there are two key points that will be addressed in future works to improve the applicability and utility of our proposal: the issue of the global plan quality, allowing to define conflicting optimization functions for the agents, and the prevention of possible dead-ends during the planning process.

Acknowledgements

This work has been partially supported by the Spanish government project MCyT TIN2005-08945-C06-06 (FEDER) and by the Valencian government project GV06/096.

References

Bellifemine, F.; Caire, G.; and Greenwood, D. 2007. Developing multi-agent systems with JADE. *Wiley Series in Agent Technology*.

Cox, J.; Durfee, E.; and Bartold, T. 2005. A distributed framework for solving the multiagent plan coordination problem. *AAMAS'05* 821–827.

Durfee, E. 2001. Distributed problem solving and planning. *Mutli-agents Systems and Applications* 118–149.

Hahndel, S.; Fuchs, F.; and Levi, P. 1996. Distributed negotiation-based task planning for a flexible manufacturing environment. *Distributed Software Agents and Applications, LNAI* 1069.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning* 103–114.

Refanidis, I. 2005. Stratified heuristic POCL temporal planning based on planning graphs and constraint programming. *Proc. ICAPS-2005 Workshop on Constraint Programming for Planning and Scheduling*.

Rehak, M.; Pechoucek, M.; and Volf, P. 2006. Distributed planning algorithm for coalition logistics in semi-trusted environment. *Proceedings of the IEEE Workshop on Distributed Intelligent Systems* 265–272.

Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. HTN planning for web service composition using SHOP2. *Journal of Web Semantics* 1(4):377–396.

van der Krogt, R. 2007. Privacy in multiagent planning: A classical definition with illustration. *AAMAS 2007 Workshop on Coordinating Agents Plans and Schedules*.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal pocl planner based on constraint programming. *Artificial Intelligence* 170(3):298–335.

Wehowsky, A.; Block, S.; and Williams, B. 2005. Robust distributed coordination of heterogeneous robots through temporal plan networks. *Proceedings of the ICAPS Workshop on Multi-agent Planning and Scheduling*.

Yokoo, M., and Hirayama, K. 2000. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems* 3:185–207.