MAPL: a Framework for Multiagent Planning with Partially Ordered Temporal Plans

Michael Brenner
Institut für Informatik, Universität Freiburg, 79110 Freiburg, Germany
brenner@informatik.uni-freiburg.de

Abstract

This paper discusses the specifics of planning in multiagent environments. It presents the formal framework MAPL ("maple") for describing multiagent planning domains. MAPL allows to describe both qualitative and quantitative temporal relations among events, thus subsuming the temporal models of both PDDL 2.1 and POP. Other features are different levels of control over actions, modeling of agents' ignorance of facts, and plan synchronization with communicative actions. For global planning in multiagent domains, the paper describes a novel forward-search algorithm producing MAPL's partially ordered temporal plans. Finally, the paper describes a general distributed algorithm scheme for solving MAPL problems with several coordinating planners. The different contributions intend to provide a simple, yet expressive standard for describing multiagent planning domains and algorithms that in the future might allow cross-evaluation of Multiagent Planning algorithms on standardized benchmarks.

1 Introduction and related work

By Multiagent Planning (MAP), we denote any kind of planning in multiagent environments, meaning on the one hand that the planning process can be distributed among several *planning* agents, but also that individual plans can (and possibly must) take into account concurrent actions by several *executing* agents. We do neither assume cooperativity nor competition among agents, nor do we impose any relation among planning and executing agents: in the general case, m planners plan for n executing agents. In the specific, yet common case of n agents, each having both planning and executing capabilities we speak of *autonomous agents*.

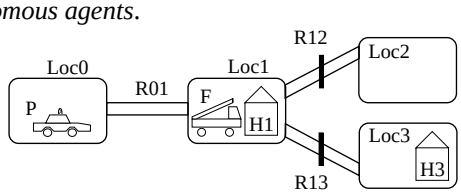

Figure 1: A multiagent planning problem

As a motivating example, fig. 1 shows a simple MAP problem as it appears in the RoboCupRescue simulation. There are two autonomous agents: police force P and fire brigade F. They have different capabilities: P clears blocked roads, F extinguishes burning houses, both can move on unblocked roads. Each action has a duration which may vary because of specific execution parameters (e.g. location distance, motion speed) and/or intrinsic unpredictability. For this example, we assume a duration of 30 to 180 minutes for *clear*, 1 to 4 hours for *extinguish*, and 2 to 4 minutes for *move*. The speed and thus the duration of *move* is controlled by each agent while the duration intervals for *clear* and *extinguish* can only be estimated. The agents' knowledge and goals are differing, too: P wants the roads to be clear, but is unaware of the state of all roads except $R01$. F wants all burning houses extinguished, knows that $H1$ and $H3$ are burning, but also that it cannot reach $H3$ because road $R13$ is blocked.

Even in this trivial example, we can make some general observations about planning in MAS that will motivate the concepts introduced in the rest of the paper.

(1) *Concurrent acting* is central to MAS (P can move to $Loc1$ and start clearing $R13$ while F is extinguishing $H1$). (2) *Metric time* is needed to realistically describe action durations and their relations. (3) Synchronizing on actions of unknown (at least to some agent) duration demands *qualitative* use of time (e.g. "after P has cleared $R13$"). A specific usage of qualitative time in MAP is (4) synchronization on communicative acts, for example "after P has informed me that $R13$ is now clear".

While many recent planning formalisms allow some degree of concurrency, most fail in providing either (2) or (3). PDDL 2.1, for example, supports metric time but enforces planners to assign exact time stamps and durations to all events [7]. In contrast, the concurrency model of [2] augments partial order plans with concurrency, thus allowing flexible, synchronized execution, but makes no difference between plans that take seconds and ones that take years. None of the planning models known to us features property (4), plan synchronization with communicative acts. To address problems (1)-(3), we will formally describe the Multiagent Planning Language MAPL ("maple") as an extension of PDDL in section 2. Synchronization with speech acts is an important concept of our distributed planning algorithm and will be presented in section 4. Fig. 2 shows part of a MAPL description for the Rescue domain. Fig. 3 shows a MAPL plan of agent F for the problem given in Fig. 1.

```
(:state-variables
  (pos ?a - agent) - location
  (connection ?p1 ?p2 - place) - road
  (clear ?r - road) - boolean)
(:durative-action Move
  :parameters (?a - agent ?dst - place)
  :duration (:= ?duration (interval 2 4))
  :condition
    (at start (clear (connection (pos ?a) ?dst)))
  :effect (and
    (at start (:= (pos ?a) (connection (pos ?a) ?dst)))
    (at end (:= (pos ?a) ?dst))))
```

Figure 2: Excerpt from a MAPL domain description

How can such a plan be found by an agent (or globally for all agents?) We see that while P can reach its goal using only his own actions, any plan reaching F's goal must involve actions of both agents. If F knows about P's capabilities, F can find such a plan. Even if F does not know P's actions and thus cannot solve the problem alone, this failure can be the trigger for cooperation and provide clues about *where* help is needed. We see that (5) the capability for single-agent synthesis of multiagent plans is a basic requirement for MAP. Section 3 presents a new single-agent algorithm for heuristic forward search in the space of MAPL (and thus also POP and PDDL) plans. It will not only find valid plans, but, in case of failure, also provide the information necessary to trigger coordination.

For the case of distributed planning, the example shows that cooperation is needed especially when (6) planners do not know how to reach their goals alone. However, most MAP re-

search has assumed that planners can find plans or sets of plans that are then coordinated or merged with those of others ([6]). Distributed hierarchical planning ([5]) is a special case thereof where the set of possible solutions is implicit in an abstraction hierarchy that is refined distributedly. But coordination is indispensable also when (7) individually valid plans are conflicting (e.g. two ambulance teams trying to rescue the same refugee). This problem resembles the detection of inconsistencies in Distributed CSPs [15]. The algorithm presented in section 4 is inspired by DCSP techniques. It addresses the problems (6) and (7) by introducing the concept of state variable responsibility.

In the following section, we will present the formal semantics of MAPL. Section 3 describes our single-agent planning algorithm which is then used in the distributed planning algorithm (section 4). In section 5, we will conclude with some remarks on past and future MAP research.

2 Multiagent plans

One main feature distinguishing MAPL from PDDL is the use of non-propositional state variables: in MAP we must dismiss the Closed-World Assumption (CWA) that everything not known to be true is false – the truth value might also be simply *unknown* to an agent. Although such belief states could be compiled to propositions (similarly to the removal of explicit negation in [8]) we will not only allow ternary state variables (with values *true, false* and *unknown*), but *n-ary state variables*. Among others, Geffner[9] uses the same concept and gives an extended formal description and justification. Still, compiling away state variables is possible and no planner is forced to internally use them.

Definition 1 *A* planning domain *is a tuple* $D = (T, O, V, type)$ *where* T *is a set of types,* O *a finite set of objects,* V *the set of state variables.* $type : O \cup V \rightarrow T$ *assigns a type to each object and state variable.* $dom : V \rightarrow \mathcal{P}(O)$ *with* $dom(v) := \{o \in O | type(o) = type(v)\} \cup \{unknown\}$ *gives the possible values for state variable* v*. A* state variable assignment *is a pair* $(v,o) \in V \times dom(v)$*, also written* $(v{=}o)$*.*

2.1 Events and actions

Definition 2 *An* event[1] e *is defined by two sets of state variable assignments: its* preconditions $pre(e)$ *and its* effects $\mathit{eff}(e)$*. For assignments* $(v = o)$ *in the preconditions* [*effects*] *of an event we will also write* $(v{==}o)$ $[(v{:=}o)]$*.*

Definition 3 *A* temporal constraint $c = (e_1, e_2, I)$ *associates events* e_1, e_2 *with an interval* I *over the real numbers, describing the values allowed for the temporal distance between the occurrence times* t_{e_1} *and* t_{e_2} *of the events:* (e_1, e_2, I) *is satisfied iff* $t_{e_2} - t_{e_1} \in I$*.* I *can be open, closed or semi-open.*

We will use the abbreviation $(e_1 \prec e_2) \in C$ for the expression $\forall I.\ (e_1, e_2, I) \in C \rightarrow I \subseteq \mathbb{R}^+$, i.e. e_1 occurs sometime before e_2. $(e_1 \preceq e_2) \in C$ is defined similarly for sub-intervals of $\mathbb{R}_0^+$.

Definition 4 *A* durative action *is a tuple* $a = (e_s, e_e, I, e_{inv})$ *where* e_s, e_e *are events (called the start and end event),* $I \subseteq \mathbb{R}^+$ *is an interval representing the temporal constraint* (e_s, e_e, I) *of the form* $e_s \preceq e_e$*, and* e_{inv} *is an event with* $\mathit{eff}(e) = \emptyset$*, called the* invariant event*. An* instantaneous action *is a durative action* $a = (e, e, [0,0], e_{inv})$ *where* $pre(e_{inv}) = \mathit{eff}(e_{inv}) = \emptyset$*. For a set of actions* Act*,* E_{Act} *denotes the set of start and events of actions in* Act*.*

[1] In this paper we assume *ground events* and actions. Instantiation of actions schemas includes instantiation of the state variable schemas (like `pos(?a)`) and requires some additions to the action preconditions and effects. For space reasons, we omit the description of the instantiation process from this paper.

There are two kinds of durative actions: those in which duration is controlled by the executing agent (e.g. reading a book) and those in which the environments determines the duration (e.g. boiling water). In the former case, the agent can *choose* the delay of the end event after executing the start event, in the latter case the end event may happen *at any time* during the interval given by the constraint. For any set of actions Act_a of an agent a we assume there is a *control function* $c_a : E_{Act} \rightarrow \{a, env\}$ describing whether the agent or the environment controls the occurrence time of an event. As agents can normally decide at least the start time of an action we assume that $c_a(e) = a$ for start events e_s. (A similar, more sophisticated concept is developed in [13].)

2.2 Mutex events and variable locks

Concurrency is a key notion in MAS. In Multiagent Planning it appears at two levels: as concurrent actions in a *plan* (or distributed over several plans by different agents) and as concurrent *planning*. Both levels are closely related: concurrency conflicts at the plan level must be detected and resolved during planning. For the plan level we define:

Definition 5 *Two events are* mutually exclusive (mutex) *if one affects a state variable assignment that the other relies on or affects, too.* $mutex(e_1, e_2) :\Leftrightarrow$
$(\exists (v{:=}o) \in \mathit{eff}(e1)\ \exists (v,o') \in pre(e_2) \cup \mathit{eff}(e_2)) \vee$
$(\exists (v{:=}o) \in \mathit{eff}(e2)\ \exists (v,o') \in pre(e_1) \cup \mathit{eff}(e_1))$

This definition corresponds to mutex concepts in single-agent Planning, e.g. in Graphplan[1]. From a Distributed Systems point of view, however, the mutex definition describes a *read-write lock* on the state variable v that causes the conflict. To solve lock conflicts during planning, section 4 introduces the concept of state variable *responsibility*.

2.3 Plans

Definition 6 *A* multiagent plan *is a tuple* $P = (A, E, C, c)$ *where* A *is a set of* agents, E *a set of events,* C *a set of temporal constraints over* E*, and* $c : E \rightarrow A$ *is the* control function *assigning to each event an agent controlling its execution.*

To simplify the next definitions we assume the set C to be always *complete*, i.e. $\forall e_1, e_2 \in E \exists I.\ (e_1, e_2, I) \in C$. This is no restriction because we can assume C to contain the trivial constraints $(e, e, [0,0])$ for all events $e \in E$ and $(e_1, e_2, (-\infty, \infty))$ for unrelated events $e_1 \neq e_2$.

Definition 7 *A set of temporal constraints* C *is* consistent *if* $\neg\exists e_1, e_2, \ldots, e_n.(e_1 \prec e_2) \in C \wedge (e_2 \prec e_3) \in C \wedge \cdots \wedge (e_n \prec e_1) \in C$*. A multiagent plan* $P = (A, E, C, c)$ *is* temporally consistent *if* C *is consistent.*

This is a reformulation of the consistency condition for Simple Temporal Networks (STNs) [4] as (E, C) is in fact an STN[2]. Using the Floyd-Warshall algorithm [3], consistency of an STN can be checked in $O(n^3)$. In planning, new events and constraints are repeatedly added to a plan while consistenty must be kept. To check this, we use an incremental variant of the algorithm (omitted from this paper) that checks for consistency violations caused by a constraint newly entered into the plan. This algorithm is in $O(n^2)$ (for every addition of a constraint).

Definition 8 *A multiagent plan* $P = (A, E, C, c)$ *is* logically valid *if the following conditions hold:*

1. *No mutex events* $e', e'' \in E$ *can occur simultaneously:*
   $\forall e', e'' \in E.mutex(e', e'') \rightarrow (e' \prec e'') \in C \vee (e'' \prec e') \in C$

[2] We are aware that STN consistency is not adequate for plans with uncontrollable action durations. We are working to integrate the concept of *dynamic controllability* into our framework [13].

For any assignment $(v == o)$ *in the precondition of any event* $e \in E$ *there is a* safe achieving event $e' \in E$*:*

2. $(e' \prec e) \in C \wedge (v := o) \in \mathit{eff}(e)$ *(achieving event)*
3. $\forall e'' \in E\ \forall (v := o') \in \mathit{eff}(e'').\ o' \neq o \ \rightarrow\ (e'' \prec e') \in C\ \vee\ (e \prec e'') \in C$ *(safety)*

Conditions 2 and 3 define plans as valid if there are no open conditions and no unsafe links, an approach well-known from partial order planning[12; 14]. Condition 1 (similarly used in GraphPlan[1]) describes threats caused by conflicting effects that do not necessarily cause unsafe links. This happens especially when events violate invariants of durative actions. Due to space restrictions extensive discussion of this topic must be omitted from this paper.

Definition 9 *A* planning problem *for an agent* a *is a tuple* $Prob_a = (Act, c_a, e_0, e_\infty)$ *where* Act *is a set of actions,* c_a *is the control function for* Act*, and* e_0, e_∞ *are special events describing the initial and goal conditions.*

Definition 10 *A multiagent plan* $P = (A, E, C, c)$ *is* valid *if it is both temporally consistent and logically valid. A plan* P *is a* solution *to a problem* $Prob_a = (Act, c_a, e_0, e_\infty)$ *of agent* a *if the following conditions are satisfied*

1. c *is consistent with* c_a*:* $c_a(e) = x \ \rightarrow\ c(e) = x$ *and* $\forall (e_s, e_e, I, e_{inv}) \in Act.$
   $[c(e_e) = a \ \rightarrow\ \forall (e_s, e_e, I') \in C.\ I' \subseteq I]\ \wedge$
   $[c(e_e) = env \ \rightarrow\ \forall (e_s, e_e, I') \in C.\ I' = I]$
2. $\forall (e_s, e_e, I, e_{inv}) \in Act.$
   $e_s \in E \ \rightarrow\ (e_e \in E \wedge e_{inv} \in E)\ \wedge$
   $(e_s \prec e_{inv}) \in C \wedge (e_{inv} \prec e_e) \in C$
3. *for* $C' = C \cup \bigcup_{e \in E} \{(e_0, e, \mathbb{R}^+), (e, e_\infty, \mathbb{R}^+)\}$
   $P' = (A, E \cup \{e_0, e_\infty\}, C', c)$ *is valid.*

So, a plan solves a problem if (1) it uses actions controlled by the agent as specified in the problem, (2) durative actions and their invariants are used as defined, (3) executing the plan in the initial state reaches the goals.

Note that the solution plan is not required to contain only actions from Act: a plan can solve an agent's problem even if it contains not a single action of that agent!

3 Single agent search for multiagent plans

In the following we look at a basic capability of agents: to plan alone (if possible). To that end we develop a forward search algorithm on partially ordered temporal plans.

A minimal condition for a plan to satisfy a set of goals is that every goal is achieved at some point in the plan and is not removed again after that point. Instead of testing goal satisfaction, we will use this condition to describe if actions can be added to a plan "at the end".

For a given plan $P = (A, E, C, c)$ we will use the following abbreviation: $\mathit{achieves}(e, (v,o)) :\Leftrightarrow (v,o) \in \mathit{eff}(e) \wedge$
$\forall e' \in E \setminus \{e\}\ \forall (v,o') \in \mathit{eff}(e').\ o' \neq o \ \rightarrow\ (e' \prec e) \in C$

Definition 11 *The* frontier F_P *of a plan* $P = (A, E, C, c)$ *is the set of achieved state variable assignments*
$F_P = \{(v,o) \mid \exists e \in E.\ \mathit{achieves}(e, (v,o))\}$

Corollary 1 *For each assignment* $(v,o) \in F_P$ *in a valid plan* P *there is a unique* achiever $e_{(v,o)}$ *with* $\mathit{achieves}(e_{(v,o)}, (v,o))$.

Definition 12 *The set of* enabled actions *for a given plan* P *and a set of possible actions* Act *is*
$\mathit{enabled}_{P,Act} = \{(e_s, e_e, I, e_{inv}) \in Act \mid \mathit{pre}(e_s) \subseteq F_P\}$

Enabled actions can be added to the plan "at the frontier", i.e. after all events achieving their preconditions. But there is potential for conflict: even if no event changes an assignment after the achiever, events later in the plan might "read" that assignment, i.e. it appears in their preconditions although not in their effects. If the newly added action changes the assignment it threatens the "reader" events. E.g. actions *extinguish* and *move* can both be enabled in plan frontier $F_P \ni (isAt(F) = Loc1)$, but both can only be applied by first apply *extinguish* and then add *move* after the reader event *extinguish*. By doing so, we automatically prevent the plan from becoming invalid as *extinguish* and *move* are mutex.

Algorithm 1 apply(a,P)

$E' := E \cup \{e_s, e_e\, e_{inv}\}$
$C' := C \cup \{(e_s, e_e, I), (e_s, e_{inv}, \mathbb{R}^+), (e_{inv}, e_e, \mathbb{R}^+)\}$
for all assignments $(v,o) \in \mathit{pre}(e_s)$ **do**
  $\mathit{readers} := \{e \in E \mid (v,o) \in \mathit{pre}(e)\}$
  if $(\mathit{readers} = \emptyset)$ or $(\neg\exists (v := o') \in \mathit{eff}(e_s) \cup \mathit{eff}(e_e))$ **then**
    // there are no readers of (v,o) *or* a *is a reader itself*
    $C' := C' \cup \{(e_{(v,o)}, e_s, \mathbb{R}^+)\}$
  else
    $C' := C' \cup \bigcup_{e \in readers} \{(e, e_s, \mathbb{R}^+)\}$
return $P' = (A, E', C', c)$

Algorithm 1 enters an enabled action a into a plan at the earliest possible position that causes no safety or mutex threats. Open conditions cannot be produced either[3] because the old plan was valid and the new action was enabled, i.e. preconditions satisfied. Without proof, we can therefore state:

Theorem 2 *If* P *is a valid plan and* $a \in Act$ *is enabled in* P*, i.e.* $a \in \mathit{enabled}_{F_P, Act}$ *then* apply(a,P) *returns a valid plan.*

We can now describe single-agent algorithms that search for (multi-agent) plans in the space of plans (like POP algorithms): for every state (=plan), *enabled* gives us a set of possible transitions (=actions) and *apply* describes the transition function from one state to a successor state. That means we can use any state-space search technique to find valid plans.

The simplest method, Breadth First Search, produces plans with minimal number of actions. For concurrent temporal plans this is unintuitive because concurrency and action duration is ignored. A better criterion for plan quality is *makespan* (minimal duration of the execution) of a plan. The calculation of a plan's makespan is a side product of the consistency check with the Floyd-Warshall algorithm: makespan corresponds to the length of the longest path in the plan assuming minimal possible duration for durative actions. An even more reasonable quality metrics for MAP is to assume *maximal* duration for *uncontrolled* actions. We will call this metrics *min-max makespan* and denote with $mmdur(a)$ the minimal (maximal) duration of a controlled (uncontrolled) action.

Quality metrics are used in a search method by sorting the search queue according to the metrics. This corresponds to A^* search with heuristic function 0. However, we need not do without heuristics. Goal distance can be estimated in various ways; the following method is based on the FF[11] heuristic.

To relax a planning problem we assume that assignments to state variables made anywhere in a plan remain true throughout the plan. Algorithm *r-apply(a,P)* is derived from *apply(a,P)* by simply not checking for "readers" and adding the new action after all its preconditions are achieved, i.e. by adding

[3] We assume $\mathit{pre}(e_e) \subseteq \mathit{pre}(e_s) \cup \mathit{eff}(e_s)$ and $\mathit{pre}(e_{inv}) \subseteq \mathit{pre}(e_s) \cup \mathit{eff}(e_s)$. Any other choice would make the semantics of durative actions problematic, e.g. non-terminating durative actions would be possible.

$(e_{(v,o)}, e_s, \mathbb{R}^+)$ to C for all $(v,o) \in pre(e_s)$. But Corollary 1 does not hold for relaxed plans: when mutex events are allowed there is not necessarily a *unique* achiever $(e_{(v,o)}$ for an assignment. To be sure to find the relaxed plans with minimal (min-max) makespan we must make sure that the *earliest achiever* is used when constraints with the precondition achievers of a new action are added to the plan. This can be accomplished by building the relaxed plan with a simple BFS-like algorithm (omitted for space reasons).

The "state" a relaxed plan P_P^R is built on is the frontier of a non-relaxed (partial) plan P. This way, the relaxed plan can be used to provide a heuristic evaluation of P. The min-max makespan of the relaxed plan including its non-relaxed prefix can then be used as an admissible heuristic for the goal distance of plan P. However, the heuristic is not very informative (the reason being that unnecessary actions in a plan can usually be executed concurrently with necessary ones and thus will not increase the makespan). We are currently experimenting with variants of this heuristic in combination with different forward search algorithms based on the techniques presented in this section.

For the purpose of this paper we can disregard the exact heuristic function for relaxed plans and use only one basic information it provides: the relaxed (un-)solvability of a problem. Like FF, our relaxed planning algorithm terminates if either no more new actions are applicable in P or if the goals are achieved at the *relaxed frontier* which is defined $F_P^R = \{(v,o) | \exists e \in E.\ (v,o) \in \mathit{eff}(e)\}$. It is clear that if a relaxed plan cannot be found then the non-relaxed problem is also unsolvable. In single-agent planning (e.g. [1], FF[11]), this property is used to prove a problem unsolvable and terminate search. In MAP, we will interpret it as a trigger for cooperation with other agents!

While our algorithm is proven sound by theorem 2, it is, in this simple form, not complete, i.e. for some MAPL problems the algorithm does not find a solution although it exists. The solutions missed can be characterized and found by an extended algorithm that we are currently developing. Discussion of this issue will be left to another paper.

4 Multiagent planning

In a multiagent environment, non-coordinating agents will probably run into one of the following problems: (1) they won't know how to solve their planning problem or (2) they will find a plan but run into execution conflicts (mutex or safety threats) with other agents' plans. In a competitive environment this might be acceptable or even intended; in all other cases it is not. For the rest of the paper we will therefore assume agents willing to coordinate at least as much as to produce a set of conflict-free plans to make plan execution predictable. This does not mean that one common, conflict-free plan must be built. The method proposed is intended to minimize synchronization of both planners and plans.

We will tackle the problem of unsolvability and the problem of conflicting plans with the same concept: responsibility for state variables. The agent responsible for a state variable is the one who is asked when another cannot reach a goal involving that variable. And the responsible agent will decide who is allowed to manipulate a variable when actions planned by two agents use a variable in a conflict-producing way.

Currently, MAPL and our algorithmic framework are only at the beginning of development. For now, we therefore make strong simplifying assumptions about responsibility:

(1) Responsibility for state variables is described by a function $resp : V \rightarrow A$. (2) The function $resp$ is fixed during the planning process. (3) The agent $resp(v)$ and only he must know controlled actions that manipulate v. (4) The agent responsible for a state variable v knows the initial value of v.

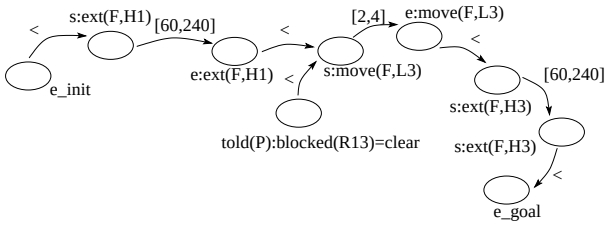

Figure 3: F's plan (including a communicative action by P)

In future work, these assumptions will be relaxed as follows: (1) We will allow a set of responsible agents negotiating about changes to the variable. (2) We will allow dynamic change of the responsible agent to better reflect who really is "using" the state variable during the course of the planning process. (3) (4) will both not be enforced any more, but be an effect of dynamic change of the responsible agent.

The responsible agent records the changes to "his" state variable in an *access history*. Thus he can give agents accessing the variable informations about the changes and they can create plans consistent with the access history.

Definition 13 *An* access history $H_v = (A, E_v, C_v, c_v)$ *is a plan where all preconditions and effects* $e_v \in E_v$ *use only assignments of one state variable* v.

H_v *is an access history* for *a plan* P *iff there is a bijective mapping* m *from* E_v *to the set* $\{e \in E | \exists o.(v,o) \in pre(e) \cup \mathit{eff}(e)\}$ *such that for all* $e_v, e'_v \in E$*:*

$(v,o) \in pre(e_v) \leftrightarrow (v,o) \in pre(m(e_v)) \wedge$
$(v,o) \in \mathit{eff}(e_v) \leftrightarrow (v,o) \in \mathit{eff}(m(e_v)) \wedge$
$(e_v, e'_v, I) \in C_v \rightarrow (m(e_v), m(e'_v), I) \in C \wedge$
$c_v(e_v) = c(m(e_v))$

By $H_{v;write}$ $[H_{v;read}]$ *we denote the sub-plan of* H_v *that contains only events* e *where* $\mathit{eff}(e) \neq \emptyset$ $[pre(e) \neq \emptyset]$.

Algorithm 2 MA Planning (agent a, current partial Plan P)

Received no message:
  P^R := relaxed plan upon current plan frontier F_P
  $unachievable = \{(v,o) \in pre(e_\infty) | (v,o) \notin F_P^R\}$
  if $unachievable \neq \emptyset$ **then**
    choose $(v,o) \in unachievable$
    investigate(v, o)
  else if found Plan P **then**
    if $\exists v$ accessed in P for which $H^r_{v;write}$ is unknown **then**
      send ask(v) **to** $resp(v)$
    else if $H^r_{v;write}$ is consistent with P $\forall v$ accessed in P **then**
      send tell(H_v) **to** $resp(v)$
      output P
      terminate (if not responsible for a state variable)

Received tell($H^r_{v;write}$) **from** agent $r = resp(v)$:
  choose (v,o) for which $\exists e \in H^r_{v;write}.\ (v,o) \in \mathit{eff}(e)$
  apply(P,$\mathrm{TOLD}_{v,o}$)

Received unachievable(v,o) **from** agent $r = resp(v)$:
  backtrack

Procedure investigate(v,o):
  if $(v == unknown) \in F_P^R$ **then**
    send ask(v) **to** $resp(v)$
  else if $\neg\exists a \in Act.\ (v,o) \in \mathit{eff}(e_s) \cup \mathit{eff}(e_e)$ **then**
    send askFor(v,o) **to** $resp(v)$
  else
    choose $(v',o') \notin F_P^R$ by regression on (v,o)
    investigate(v', o')

An agent using a fact in his plan need not know how, why or by whom it has been achieved. In temporally uncertain domains the agent must even plan not knowing *when* exactly the fact will become true. We achieve this by introducing *reference events* into a plan: One that will be only briefly mentioned is e_{tr}, the temporal reference point lying before all other events. All agents know e_{tr} and thus can describe *absolute times* as constraints with e_{tr}. The other type of reference event are the *communicative events* $\text{TOLD}_{v,o}$ where $pre(\text{TOLD}_{v,o}) = \emptyset$ and $\mathit{eff}(\text{TOLD}_{v,o}) = \{(v, o)\}$. By entering new information into the current plan with TOLD agents can use it like any effects of other events: as preconditions of new actions and as temporal reference in constraints. It is the latter use that is especially helpful: the TOLD event provides automatic synchronization with another agents plan. E.g. fig. 3 shows how the fire brigade synchronizes on the police clearing a road without knowing when or how this is done.

We will now explain the basic distributed planning algorithm for MAPL. For space reasons, alg. 2 shows only the reactions of a *planning* agent to various messages and his behavior when no messages are received. We have omitted the part of the algorithm describing the behavior of an agent *responsible* for a variable. Of course an agent may have both roles (or even responsibility for more than one variable). To simplify presentation, the algorithm is described using the nondeterministic *choose* operator that also defines backtrack points. These are beginning points for future research: how can the nondeterministic choices be heuristically guided?

The idea of the algorithm is simple: by building a relaxed plan, the planning agent detects goals involving state variables he does not know about, cannot manipulate or could if only some earlier condition were satisfied. He contacts the responsible agents to receive more information or delegate a subgoal concerning the variable. The responsible agent answers the question or adopts a temporary goal to help the asking agent. For this simplified version of the algorithm, we assume that the responsible agent can always help. In general, this is of course not true, and when the responsible agent fails to help asynchronous backtracking [15] is triggered; but this is out of the scope of this paper.

There are three possibilities for termination: either an agent can backtrack no more, or an agent not responsible for a variable has found a plan, or a responsible agent gets information that all other agents have terminated or found a plan so that he can terminate, too, without "shirking" his responsibility to inform other agents about his variable.

5 Conclusion and future work

We have presented a new representation for MAP domains and solutions along with basic single- and multi-agent planning algorithms. The algorithms have been implemented in Java; agents are modeled as threads communicating via pipes. The implementation is preliminary, in particular, we are still in development of a parser to automatically read in problems. At the moment, only hand-coded toy problems (like the Rescue example) are solved by the planners.

A lot of exciting work is possible now: we are already experimenting with different heuristics and search methods on all levels of the algorithms. Dynamic responsibility hierarchies will improve flexibility and performance of the algorithm (comparably to dynamic agent hierarchies in Asynchronous Weak-Commitment Search [15]). We will also develop a set of benchmark problems and apply the distributed algorithm in a realistic application (RobocupRescue).

MAP has been a topic of interest in AI for quite some time. However, not much work has been published, neither in the field of Multiagent Systems (MAS) nor in Planning; furthermore, what has been published is mostly stand-alone work that has not led to a steady development in MAP research. In our view, this is due to an unfavorable separation of the (single-agent) planning phase and the (multi-agent) coordination and execution phase that has lead AI Planning researchers to concentrate exclusively on the former while MAS researchers almost as exclusively deal with the latter. This separation is only possible with strong assumptions that narrow the generality of the proposed approaches, for example the assumption that actual planning of each agent can either be easily done before coordinating in a "classical" way or is eased by a given hierarchical task decomposition. It seems obvious to us that such a separation is artificial: planning and coordination should be interleaved to enable agents both to find plans at all and to detect probable execution conflicts of their (partial) plans as early as possible.

With PDDL 2.1, the AI planning community has only recently fully acknowledged the need for sophisticated models of concurrency (earlier exceptions include most notably work by M. Ghallab[10]). MAPL is an attempt to extend that representation in a way that allows flexible execution *after* and easy coordination *during* the planning process. The single- and multi-agent algorithms we propose are only first steps and hopefully not the only possible ways to synthesize MAPL plans. We hope that our representation will allow to conveniently describe largely differing MAP domains for which researchers can propose and cross-evaluate very different algorithmic approaches, thus promoting the field of Multiagent Planning.

References

[1] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artifi cial Intelligence*, 90(1-2), 1997.

[2] Craig Boutilier and Ronen Brafman. Partial order planning with concurrent interacting actions. *JAIR*, 2001.

[3] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1992.

[4] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artifi cial Intelligence*, 49, 1991.

[5] E. Durfee and T. Montgomery. Coordination as distributed search in hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, 1991.

[6] Eithan Ephrati and Jeffrey S. Rosenschein. Divide and conquer in multi-agent planning. In *Proc. AAAI-94*, 1994.

[7] Maria Fox and Derek Long. *PDDL 2.1: an Extension to PDDL for Expressing Temporal Planning Domains*, 2002.

[8] B. Gazen and C. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *ECP '97*, 1997.

[9] H. Geffner. Functional STRIPS: a more flexible language for planning and problem solving. In Jack Minker, editor, *Logic-Based Artifi cial Intelligence*. Kluwer, 2000.

[10] M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *Proc. of AIPS '94*, 1994.

[11] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14, 2001.

[12] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *Proc. IJCAI '01*, 2001.

[13] T. Vidal and H. Fargier. Handling contingency in temporal constraint networks. *Journal of Experimental and Theoretical Artifi cial Intelligence*, 11, 1999.

[14] Daniel Weld. An introduction to least commitment planning. *AI Magazine*, 15(4), 1994.

[15] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: a review. *Autonomous Agents and Multi-Agent Systems*, 3(2), 2000.