any constraints on end time of a task, so it can not express all of the relations in Allen's algebra. On the other hand we believe our approach is easier to track and expressive enough for many practical problems.

Dechter and Meiri can reason about metric time constraints and propose an algorithm that can solve simple temporal constraint satisfaction problems in polynomial time.

Bacchus suggested a simplistic approach to generate plans that include actions with same time stamp but in fact the plans are sequential because the ordering of the actions with same time stamp is important. The actions in this approach can have instant effects that are used to control concurrency and delayed effects to represent the actions that have a duration greater than one.

TaL planner is extended to reason about time, concurrency and resources. It prunes the search space using control rules written in temporal logic. They define two actions as mutually exclusive if the effects of them conflict at some point that these two actions overlap. Tal planner can perform concurrent numeric computation only on resources.

Smith and Weld extends the definition of mutual exclusion for actions that can have durations. TGP uses a more generalized planning graph that can handle actions with durations and employs the extended mutual exclusion reasoning when searching for a plan. TGP actions have preconditions that hold through out the execution and effects that are guarantied to be true at the end of action. TGP does not allow intermediate effects. One can argue that preconditions holding during the actions may be too restricted.

Conclusion

In this paper we have presented a formalism to explicitly represent time in HTN planning. Based on this formalism we defined TimeLine a sound and complete HTN planner that can reason about time. Our experiments concluded that our approach is feasible and worth future study.

The formalism we present is expressive enough to represent most of the practical problems and yet still not complex. We do not require the specification of any resource usage in any level of the task abstraction. Instead we define concurrent update rules for numeric state variables that can represent these recourses.

A future study may concentrate on reducing the backtracking points in the implementation. Number of backtracking becomes a real problem as the problem size and concurrency level increases. A better implementation may be backtracking to representative time points instead of backtracking all time points.

References

Allen,J. 1983. Maintaining knowledge about temporal intervals. Communication of the ACM 26(11):832-843

Simith D.,Weld, D. 1999. Temporal Planning with Mutual exclusion reasoning. (IJCAI-99)

Kohler,J. 1998. Planning under resource constraints.(Proc. ECAI-98)

Karlsson, L..Gustavfsson, J., Doherty, P. 1998. Delayed effects of Actions .(ECAI-98)

Fox, M.,Long, D. 2001. PDDL2.1: An Extension to PDDL for expressing Temporal Planning Domains.

Kvarnstrom, J., Doherty, P. 2001. TAL planner: A Temporal Logic-based Forward Chaining Planner. Annals of Mathematics and Artificial Intelligence.

Erol, K., Hendler, J., Nau, D. 1994. Semantics for Hierarchical Task-Network Planning. Tech. Report CS TR-3239, UMIACS TR-94-31, ISR-TR-95-9, University of Maryland, March, 1994a.

Erol, K., Hendler, J., Nau, D. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In *Proc. Second International Conf. on AI Planning Systems (AIPS-94*), June, 1994b, pages 249-254.

Munoz-Avila, H., Aha, D., Breslow L., Nau, D. HICAP: an interactive case-based planning architecture and its application to noncombatant evacuation operations. In IAAI-99,1999, pages 870-875.

Dechter, R,;Meiri, I.; and Pearl, J. 1991.Temporal constraint networks. Artificial Intelligence 49:61-95

Rintanen, J., Jungholt, H. Numeric state variables in constraint-based planning.

```
TimeLine (N, A, S, T)

1 If N and A are empty
2   return  empty  plan;
3 else
4 if there is a task t such that time constraints
  on start time of t is satisfied at T
5    If t is a simple task
6     Choose applicable operator instance o for t
7     end_time = time of the latest effect in o's
                 effects
8     A'=A \bigcup effects of o
9     N'=reduce(N,t,empty tasknetwork,T,end_time)
10    P' = TimeLine(N',A',S,T)
11    return  (t [T, end_time])+ P'
12   else if ti is a composite task
13    Choose a T-executable reduction R for t,
      R contains subtasks t_1 to t_n
14    end_time=(max(end t_1)(end t_2)..(end t_n))
15           N' = reduce(N, t, R, T, end-time)
16    return TimeLine(N',A,S,T)
17   end if
18 else
19 if A is empty and there is not a task t
   such that time constraints on start time of t
   is satisfied at T' > T
19      Chose a task t such that time constraint on
       starting time of t is satisfied at T.
20     Go to line 5
21 else
22    Choose ti be the progress task or a task in
      N such that time constraints on its start
      time are satisfied.
23    if ti is the progress task
24       (S',A') = Exec(S,A,T+1)
25       if S' is a valid state
25          return TimeLine(N,A',S',T+1)
26       end if
27    else
28       Go to line 5
29    end if
30 end if
31  End TimeLine
```

Figure 6: Pseudo code of TimeLine algorithm

Theorem 1: If one of the non-deterministic traces of TimeLine(N,A,S,T) returns a solution P, then P is a solution to the planning problem (N,A,S,T).

Proof : TimeLine returns a solution at

- *Line 2*: This line is executed when both N and A are empty and returned solution is an empty plan. This is Case 1 of the definition for solution planning problem.
- *Line 11*: If line 5 is executed after line 4 this case corresponds to Case 2.1 of solution definition. If line 5 is executed after line 20 this case corresponds to Case 3.1 of solution definition. If line 5 is executed after line 28 this case corresponds to Case 4.2 of solution definition);
- *Line 16*: If line 12 is executed after line 4 this case corresponds to Case 2.2 of solution definition. If line 12 is executed after line 20 this case corresponds to Case 3.2 of solution definition. If line 12 is executed after line 28 this case corresponds to Case 4.3 of solution definition;
- *Line 25*: The solution returned in this line corresponds to Case 4.1 of solution definition.

Implementation and experiments

We have implemented deterministic version of TimeLine algorithm. We have tested our implementation on the Logistic domain which is naturally concurrent and easily extendible to include numeric state variables.

The problems we used in our experiments were based on 30 problems used in AIPS98 planning competition. Basically we tried to create same set up that is defined in (Kvarnstrom, J. and Doherty, P. 2001). We set the space capacity for trucks to 5 and for airplanes to 25. We randomly generated the package sizes between 1 and 3. We also randomly defined distance between two locations in the range 1 to 25. We have create 20 random instanced for each of the 30 problems. We ran TimeLine on 20 problems instances then take the average and we did this for 30 problems.

We ran the experiments on a Pentium III-600 machine with 128 memory and Windows 98 operating system running on it. We compared our results with the published results of TAL planner and verified the feasibility of our approach. In fact most of the cases TimeLine performed better than TAL planner. Considering the configuration differences between the machines (TAL planner experiments performed on Pentium II-333) and the problems, this performance difference may not illustrate a great deal. As we can see from the preliminary results our approach is feasible and worth for future study.

No	TaL Planner	TimeLine		No	TaL Planner	TimeLine
1	270	591		16	10004	1455
2	811	644		17	2895	1248
3	2063	1165		18	21080	5047
4	5889	1412		19	18466	4649
5	541	601		20	37815	6317
6	6729	1877		21	39436	3224
7	1061	643		22	71402	20107
8	5658	1163		23	2434	3354
9	9594	2303		24	39096	1455
10	5738	3731		25	146921	10559
11	911	646		26	83960	22369
12	14871	1084		27	72814	9388
13	16524	2016		28	670284	24974
14	6800	2218		29	34550	21261
15	1512	3850		30	312099	9023

Table 1: Average time in miliseconds to solve problems

Related Work

Allen's interval algebra defines the relations on two tasks using their end points. Our approach does not define explicitly

- If C is a time constraint in R and C refers to *(end t1)* then R' contains a constraint C' such that C' is same as C except *(end t1)* is replaced by *time$_{end}$*.

```
R =((:t1 (load-airplane package plane airport1)
    :t2 (unload-airplane package plane airport2))
    ((= (start t1) now)(>= (start t2) (end t1))))
    )

r =((:t3 (!fly-airplane plane airport3 airport1)
     :t4 (!load-airplane package plane airport1))
    ((= (start t4) (end t3) ) ) )

R'=((:t2(unload-airplane package plane airport2))
     :t3 (!fly-airplane plane airport3 airport1)
     :t4 (!load-airplane package plane airport1))
    ((>= (start t2) (max (end t3) (end t4)))
     (= (start t4) (end t3) ) ) )
```

Figure 5: Example for reducing a task network

When we are talking about a simple task we can easily point the start and end time of it. Basically the time when matching operator is applied is its starting time and the time for the last effect of that operator is the end time. However this can not be directly applied to composite tasks. What happens if at time T a composite task t is decomposed into n subtasks using an applicable method and none of its subtasks start at time T. In such a case it is does not make sense to say that starting time for t is T. On the other hand if at least one of its subtasks can start at T then we can safely say that starting time of t is T. This leads to the definition of T-executable reduction.

Definition 12: Let S be the state at time T and A be an agenda after T. Let t be a composite task, r be a reduction of t at time T and t_i be a task in r such that time constraints on start time of t_i are satisfied at time T, then T-executable reduction R for $t$is defined as:

- If t_i is a simple task and it is T-executable in S with A then R is equal to r and R has an additional time constraint C (= (start t$_i$) T) if r does not contain C.
- If t_i is a composite task and it is R' is a T-executable reduction of t_i that with tasks $t_{i1}..t_{in}$ then R is equal to reduce(r ,t$_i$, R', time$_{start}$, time$_{end}$) where time$_{start}$ is equal to T and time$_{end}$ is equal to (max (end t$_{i1}$) (end t$_{i2}$) .. (end t$_{in}$)).
- Let T be the time and S be the state at T and A be an agenda after T. A composite task t is T-executable at time T in state S with A if there exists a t-executable reduction for t.

Plan and Planning Problem

We now define what is a plan, a planning problem and what is a solution to the planning problem.

Definition 13: Let T be the time, S be the state at T and A is an agenda after T. The effect of achieving a *progress task* at time T is defined as Exec(A,S,T+1)

Definition 14: A plan is a list of (task$_1$ [time$_{start}$, time$_{end}$]) where task$_1$ is a ground simple tasks and time$_{start}$ and time$_{end}$ are the start and end times of task$_1$.

Definition 15: A planning problem P is a tuple (N,A,S,T) where N is a task network, A is an agenda after T, S is a state and T is the current time.

Definition 16: Let P be a planning problem (N,A,S,T) then a solution Π of problem P is defined as follows:

- *Case 1*: If both N and A are empty then Π is an empty list.
- *Case 2*: If there exists a task $t_i$in N such that the time constraints on start time of t_i are satisfied at T and there exists an equality constraint for start time of t_i and
  - *Case 2.1*: ti is a simple task. Let O be an applicable operator instance of ti and time$_{end}$ be the time of latest effect in O. Let A' and N' be defined as:
    A' = A ∪ effects of O
    N' = reduce(N ,ti, empty task network, T, time$_{end}$)
    Let Π' be the solution to the problem (N',A', S, T) then Π = (ti [T, time$_{end}$]) + Π'
  - *Case 2.2*: ti is a composite task. Let R be an T-executable reduction of ti and ti1 .. tin are the tasks in R. Let time$_{end}$ be(max (end t$_{i1}$) (end t$_{i2}$) .. (end t$_{in}$)) Let N' be reduce(N ,ti, R, T, time$_{end}$) then Π is the solution to the problem (N',A, S, T).
- *Case 3*: If A is empty and there is no task t in N such that the time constraints on start time of ti are satisfied at T'¿T. Then let ti be a task in N such that t_i's time constraints are satisfied at T.
  - *Case 3.1*: ti is a simple task. Then Π is defined as in *Case 2.1*
  - *Case 3.2*: ti is a composite task. Then Π is defined as in *Case 2.2*
- *Case 4*: Let ti be the progress task or a task in N such that time constraints on its start time are satisfied. Then
  - *Case 4.1*: ti is progress task. Then let S' and A' be defined as
    (S',A') = Exec(S,A,T+1)
    If S' is a valid state then Π is a solution to problem (N,A',S',T+1);
  - *Case 4.2*: ti is a simple task. Then Π is defined as in *Case 2.1*
  - *Case 4.3*: ti is a composite task. Then Π is defined as in *Case 2.2*

Algorithm

We now define the TimeLine algorithm that finds a solution to the planning problem (N,A,S,T) as defined in pervious section. We also state and prove the soundness of the algorithm.

The pseudo-code for the algorithm is presented in Figure **??**. TimeLine is non-deterministic straight forward implementation of the solution defined for a planning problem in the previous section.

we do not allow the following (start t1) ¡ (end t2) or (start t1) $\geq$ ((start t2) - 3) or (start t1) = (end t2 - 5). All of these constraints require t1 to start before some time that we do not know in advance and by the time these points become known these constraints are either satisfied or not and there is no time point after that that can satisfy these constraints. We find these constraints hard to trace therefore do not make use of them. For similar reasons we do not define constraints on end time of tasks for example (end t1) $\geq$ (end t2).

Definition 7: Given a task t_1 and another task t_2 , a time constraint on start time of t_1 is one of the following expressions:

- *(= (start t_1) bound)* where *bound* can be either *(start t_2)* or (end t_2) or "*now*", which means current time, or a nonnegative integer c or (+ base delay) in which *delay* is a nonnegative integer and *base* is either *(start t_2)* or *(end t_2)* or "*now*".
- (>= *(start t_1) bound*) where bound is as defined above
- (>= *(start t_1) (max $bound_1$ $bound_2$ .. boundk)* where $bound_i$ is as defined above. This is a short hand notation for a list of time constraints of (>= (start t_1) $bound_i$) where i is in [1,k]

Time constraints of the first type are called equality constraints where as second and third types are called greater than constraints. The following time constraints are satisfied at time T

- (= (start t_1) bound) where *bound* is either "*now*" and current time is T or a nonnegative integer c that is equal to T.
- (>= (start t_1) bound) where *bound* is either "*now*" and current time is T or a nonnegative integer c that is less than or equal to T.
- (>= (start t_1) (max $bound_1$ $bound_2$ ..$bound_k$) where for every i, $bound_i$ is nonnegative integer that is less than or equal to T.

Given a set of time constraints U on start time of task t_1, U is satisfied at time T if all of the following holds;

- U contains at most one equality constraint C, and C is satisfied at T.
- All of the greater than constraints in U are satisfied at T.

Definition 8: Task Network is a list of tasks $(t_1..t_n)$ and a list of time constraints on the start time of these tasks such that end points of a task referenced in a time constraint is in the list of tasks $t_1..t_n$.

Definition 9: A method has the following form
(:method head precondition subtasks)
where *head* is a composite task, *precondition* is a conjunctive expression and *subtasks* is a task network.

Figure 4 includes two, methods we defined for our extended logistic domain. First method decomposes *air-deliver* task into two subtasks which are labeled as t_1 and t_2. According to the time constraints t_1 should start immediately and t_2 can start after t_1 ends. The second method decomposes the task unload-airplane-at into two subtasks. There is no time constraint for t_3 but t_4 should start when t_3 ends.

```
(:method (air-deliver ?obj ?airport-from ?city)
 ;;PRECONDITION
 ((obj-at ?obj ?airport-from)
  (in-city ?dest ?city)
  (airport ?dest)

  (= (airplane-space ?plane) ?space)
  (volume ?obj ?vol)
  (call >= ?space ?vol))
;; SUBTASKS
 ((:t1 (load-airplane ?obj ?plane ?airport-from)
   :t2 (unload-airplane-at ?obj ?plane ?dest))
  ((= (start t1) now) (>= (start t2) (end t1))))
 )

(:method load-airplane ?obj ?plane ?airport)
;;PRECONDITION
 ((not(moving ?plane ?dest))
 (airplane-at ?plane ?somewhere)
 (different ?somewhere ?airport)
 (= (airplane-user ?plane) ?user)
 (call < ?user 1)
 )
;; SUBTASKS
 ((:t3 (!fly-airplane ?plane ?somewhere ?airport)
   :t4 (!load-airplane ?obj ?plane ?airport 0))
  ((= (start t4) (end t3) ) ) )
)
```

Figure 4: Method for air-deliver task in extended logistic domain

Definition 10: Let S be the state at time T, t be a composite task and M be a method. Let *mgu* be the most general unifier that unifies the head of M and t. Then M^{mgu} is an *applicable* method instance for t in state S at time T if the precondition of M^{mgu} is satisfied in S. If α is a list of bindings for the free variables in precondition of M^{mgu} such that precondition of M^{mgu} is satisfied then $(subtasks^{mgu})^{\alpha}$ is a *reduction* of $t$at time T.

The idea behind reducing a task network is to replace a task in the network with one of its reductions and update all the time constraints that refer to the old task to include references to new tasks. Figure 5 gives an example task network R which is reduced to R' using the method for *load-airplane* task defined in Figure 4.

Definition 11: Let R be a task network and t be a task in R. Let r be a task network with tasks $t_1..t_n$ then *reduce(R, t, r, $time_{start}$, $time_{end}$)* is a new task network R' satisfying the following:

- R' contains all tasks in r and all tasks in R except t
- R' contains all time constraints in r and in R except the constraints that are on start time of t and those refer to end points of t.
- If C is a time constraint in R and C refers to *(start t)*, then R' contains a constraint C'such that C' is same as C except *(start t)* is replaced with $time_{start}$.

state. If in an agenda A every t is grater than T then A is an agenda *after* T.

An effect e_1 that is promised to be true at time T is *consistent* with an agenda A iff A does not contain an effect e_2 that is promised to be true at time T and e_2 is mutually exclusive with e_1.

Figure 3 shows the load operator we defined for our extended logistic domain. There may be a state that satisfies the preconditions of both *load* and *drive* operators. It is obvious that for a truck these two operators should not overlap on the timeline. To handle this case we should extend the definition of applicability for an operator, to include consistency with the current agenda. Therefore if a *drive* operator on a truck is scheduled at time T, *load* operator on the same truck should not be applicable. We should allow concurrent *load* operators unless the cumulative numerical effects of these load operators lead to an out of range value for a numeric state variable in the future states. It is not always possible to detect these inconsistencies by looking at the agenda, since there may be additions to agenda and that can fix what seems to be a problem. However it is easy to check for the state just after the current one. Let's say in the current state truck B has 5 units of space available. If we schedule 6 concurrent loading into B now, we can immediately see that (given that all packages have positive sizes) one unit time later we will ran out of space. No unload operation can fix this problem because the value should always stay in the range no matter in what order these effects are carried on.

```
(:operator
 (!load ?obj ?truck ?loc)
  ( (not (moving ?truck ?dest))
    (obj-at ?obj ?loc)
    (truck-at ?truck ?loc)
    (= (truck-space ?truck) ?space)
    (volume ?obj ?vol)
    (call >= ?space ?vol) )
  (([1]((+=(truck-user ?truck) 1)
        (-=(truck-space ?truck) ?vol)
        (not (obj-at ?obj ?loc))))
  ([1,2]((truck-at ?truck ?loc)))
  ([2]((-= (truck-user ?truck) 1)
       (in-truck ?obj ?truck)))))
```

Figure 3: Load operator for extended logistic domain

Definition 5: Let S be the state for time T, A be an agenda after T, t_1 be a simple task and O be an operator. Let *mgu* be the most general unifier that unifies with the head of O and t_1. Then O^{mgu} is an *applicable operator instance* for t_1 at time T in state S with agenda A iff the following holds

- There is a satisfier α for the precondition of O^{mgu} in S.
- None of effects in effect-list of $(O^{mgu})^\alpha$ with same time is mutually exclusive with each other.
- All of the effects of $(O^{mgu})^\alpha$ are consistent with A
- All the numeric variables stay in the range at time $T + 1$.

Let S be the state at time T and A be the agenda in which all of the effects are after T. A simple task t_1 is *T-executable* if there exists an applicable operator instance for t_1 in S with A.

The purpose of agenda is to keep track of changes that will be made to future states. Given the current time, state and the current agenda successor states can be generated by performing the effects in the temporal order.

Definition 6: Let S be the state at time T and A be an agenda after T. Let $e_1..e_n$ be the effects in A that are promised for time T' where T' is T+1 then *Exec(A,S,T')* creates a new state S' in and a new agenda A' with the following properties:

- Let p be *(= numeric-variable ex-value)* and S contains p. If there is an effect e_i that assigns value *new-value* to *numeric-variable* then S' does not contain p and S' contains (= *numeric-variable new-value*)
- Let p be *(= numeric-variable ex-value)* and S contains p. Let E^+*(numeric-variable)* be subset of e_1 to e_n such that every $e_i \in E^+$*(numeric-variable)* increases the value of numeric-variable. E^-*(numeric-variable)* is defined similarly for the effects that decrease the value of *numeric variable*. *Total increase* is sum of the increase amounts of effects in E^+. *Total decrease* is sum of the decrease amounts of the effects in E^-. Then S' does not contain p and S' contains (= numeric-variable new-value) where *new-value* is equal to *ex-value* + *total increase* – *total decrease*. If *ex-value* + *total increase* or *ex-value*– *total decrease* is not in the range defined for *numeric-variable* then S' is an invalid state.
- Let p be an atom of the form *(p-name arg*$_1$ *arg*$_2$ .. *arg*$_n$*)* and *p-name* is not +=, -= or =. If p is in S and there is an e_i such that e_i is *(not p)* then S' does not contain p, if there is no such e_i S' contains p. If there is an e_i such that e_i is p then S' contains p.
- A' is same as A except A' does not contain effects e_1 .. e_n.

Time Constraint, Task network, Method

End points of a task t are the start and end times of t which we represent as *(start t)* and *(end t)* respectively. End time of a simple task is the time of last effect in the operator instance that is chosen to achieve that task. Therefore once the operator is chosen, end time of the simple task is known. End time of a composite task is maximum of the end times of the subtasks in the decomposition of the method chosen to achieve this composite task. End time of a composite task becomes known when all of its subtasks end times are known. We use the end points of tasks to define temporal constraints. We concentrated on constraining the start time of tasks explicitly. We can define both metric and qualitative constraints. For example if t1 and t2 are two tasks the following are the time constraints on start time of t1; (start t1) $\geq$ ((end t2) + 5) or (start t1) = ((start t2) + 4) or (start t1) $\geq$ (start t2) While we are handling the general cases, there are some combinations that we don not allow in our time constraint definition. For example if t1 and t2 are two tasks assignment operations on the same variable at the same time, even though the assigned values are same. Concurrent increase and decrease operations on the same numeric variable can be permitted as long as the value of the variable stays in the defined range in all intermediate states produced by any permutation of these operations. Since addition and subtraction are commutative operations, the result of the any permutation will be the same. To ensure that the value always stays in the range, it is enough to check pessimistic cases in which all increase or all decrease operations are performed first.

State, Agenda, Operator

Definition 1: State is a collection of positive ground atoms of the form $(p\ t_1\ t_2\ ..\ t_n)$ where p is the predicate name and t_1 to t_n are argument terms. Value of a numeric state variable is represented by an atom of the form *(= variable value)* where *variable* is the numeric state variable and *value* is the value of the *variable* in this state. A valid state can not contain both (= *variable value1*) and (= *variable value2*).

Main elements of HTN planning are simple tasks and composite tasks. Operators define a set of changes in the current state in order to achieve simple tasks. Composite tasks can be achieved by decomposing them into subtasks and then achieving these subtasks. Methods define decompositions for composite tasks.

Classical HTN operators have a precondition to hold in the state just before the operator is applied. Operators have effects which will be true in the next state of the world. We extend this definition to represent operators that may have a duration of more than one unit time. We do not require the precondition of an operator to hold through out the execution. Effects of an operator can not change the state in which it's precondition is evaluated. We eliminated instant effects because they make it hard to trace the deleted precondition interactions. This way we always evaluate the preconditions in a stable state. We let the operators have effects at intermediate time points, not only at the end so the operators may represent gradual changes in the successor states. *Effects* are the promises that will be true in a successor state. Effects may assign a value to a numeric state variable, increase or decrease the value of a numeric state variable, add or delete an atom in or from the state.

Definition 2: An operator has the following form

(:operator head precondition effect-list)

where head is a simple task, precondition is a conjunctive expression and effect-list is a list of timed effects. Timed effects can be in one of the following forms:

([time$_1$*]*$(e_1 \ldots e_n)$)or *([time*$_2$*,time*$_3$*]*$(e_1 \ldots e_n)$)

where e_i's are effects and the intended meaning of first form is e_i's will be true at (*start time of operator* + *time*$_1$). The intended meaning of the second form is e_i's will be true in the states associated with inclusive time interval [*start time* + *time*$_2$, *start time* + *time*$_1$]. In this notation *time*$_1$, *time*$_2$ and *time*$_3$ should be positive integers or numeric expressions. If the result of the numeric expression is not an integer we take the ceiling of the result. More over *time*$_3$ should be greater than or equal to *time*$_2$.

```
(:operator (!drive ?truck ?loc-from ?loc-to)
 ;;PRECONDITIONS
 ( (not (moving ?truck ?dest))
   (= (truck-user ?truck) ?user)
   (call = ?user 0)
   (truck-at ?truck ?loc-from)
   (distance ?loc-from ?loc-to ?dist)
   (assign ?duration (call ceil (call / ?dist 2))
  ))
 ;;EFFECTS
 (([1]
   ((=(truck-user ?truck) 1)
    (=(truck-arrives ?truck ?loc-to)
                          (call - ?duration 1))
    (moving ?truck ?loc-to)
    (not (truck-at ?truck ?loc-from))))
 ([2,?duration]
    ((-= (truck-arrives ?truck ?loc-to) 1)))
 ([?duration ]
    ((= (truck-user ?truck) 0)
     (not (moving ?truck ?loc-to))
     (truck-at ?truck ?loc-to))))
)
```

Figure 2: Drive operator for extended logistic domain

Figure 2 shows *drive* operator we defined for logistic domain in which we added some numeric state variables. The precondition of the operator states that number of users that are working on this truck should be zero and truck should not be in motion. *Assign* statement in the precondition simply binds the value of its second term to its first term. In this case ?duration is assigned to travel time between ?loc-from and ?loc-to when the truck speed is 2. One unit time later the state and current location of the truck is updated also the number of truck users for this truck is set to 1 and a counter that shows the time left to arrive ?loc-to is initialized. After that at every clock tick this counter is decreased by one. Finally the state and the location of the truck is updated . We also decrease the number of users for this truck. One thing to notice is we assign the value of (truck-user ?truck) to one at the beginning instead of increasing it by one. That is because we want two overlapping drive operations on same truck to be mutually exclusive.

Definition 3: Two effects e_1 and e_2 are mutually exclusive if any of the following holds:

- if e_1 and e_2 are logical negations of each other
- if e_1 assigns a value to a numeric state variable v and e_2 assigns or increases or decreases the value of v
- if e_1 decreases or increases the value of a numeric state variable v and e_2 assigns a value to v.

Since we have delayed effects that may appear sometime in the future we need a structure that remembers all of the promises toward future states.

Definition 4: Agenda is a collection of pairs (t, e) where e is an effect and t is the time when e will be true in the

TimeLine: An HTN Planner That can Reason About Time

Fusun Yaman and **Dana S. Nau**

Department of Computer Science
University of Maryland
College Park, Maryland 20742
{fusun, nau}@cs.umd.edu

Abstract

In this paper we present a formalism for explicitly representing time in HTN planning. Actions can have durations and intermediate effects in this formalism. Methods can specify qualitative and quantitative temporal constraints on decompositions. Based on this formalism we defined a planning algorithm TimeLine that can produce concurrently executable plans in the presence of numeric state variables. We state and prove the soundness of the algorithm. We also present the experimental results of the TimeLine implementation that shows the feasibility of our approach.

Introduction

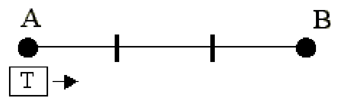
Actions with different durations, simultaneous action execution and reasoning with metric quantities are three characteristic of real-world planning problems. Recently studies on artificial intelligence planning concentrated on developing formalisms for representing time and creating temporal plans. The planning domain definition language (PDDL 2.1) for AIPS 2002 planning competition can define actions with durations, and address the concurrency issues in the presence of numeric state variables.

The difficulty aroused with concurrency is to control the overlapping action executions. The problem gets more complicated when there are limited number of shared resources. When resources are identified and resource needs for every action are explicitly defined, then two actions with conflicting resource requirements can be defined as mutually exclusive. In this approach the search space can be pruned effectively if it's accompanied by good resource management techniques. The more general case is when there are numeric state variables that can be updated concurrently. Numeric state variables can be used to represent resources but not every numeric variable can be seen as a resource.

Numeric computations and time can be handled easily by HTN planners. For this reason HTN planners are conveniently used for practical applications. In this paper we present a formalism for explicitly representing time in HTN planning. Actions can have durations and intermediate effects in this formalism. Methods can specify qualitative and quantitative temporal constraints on decompositions. Based on this formalism we defined a planning algorithm that can produce concurrently executable plans in the presence of numeric state variables. We state and prove the soundness of the algorithm. We also present the experimental results of the implementation that shows the feasibility of our approach.

Formalism

Performing numerical computations is an important issue for real-world problems. Some HTN planners like SHOP have already incorporated this functionality. Resources generally represent some features in the domain that are limited in number, like space available in a truck. Even though numeric state variables can be used to represent these resources, the opposite need not to be true. For example, let's say the distance between two cities A and B is 6 units and there is a truck T that has a speed of 2 units per unit time. If T is at A and will travel to B then as T moves, the distance between A and the current location of T increases (see dist(A,T) in Figure 1). Similarly travel time left to B is a numeric variable (see timeTo (T,B) in Figure 1). We believe these two numeric variables do not represent any resources. Therefore, instead of identifying the resources and defining operations on these resource, we will go with the more general way and define concurrent update rules for numeric state variables.



Time	0	1	2	3
Dist(A,T)	0	2	4	6
TimeTo(T,B)	3	2	1	0

Figure 1: Dist(A,T) is distance between A and current location of truck T, TimeTo(T,B) is time left to reach B

The value of a numeric variable can be assigned to a constant, decreased or increased by constant amount. We define assignment operations on the same variable at the same time, as mutually exclusive updates. Therefore we don't allow two