TLplan-C: An Extended Temporal Planner for Modeling Continuous Change

Serdar Kecici and **Sanem Sariel Talay**
Department of Computer Engineering
Istanbul Technical University, Istanbul, Turkey
{kecici,sariel}@itu.edu.tr

Abstract

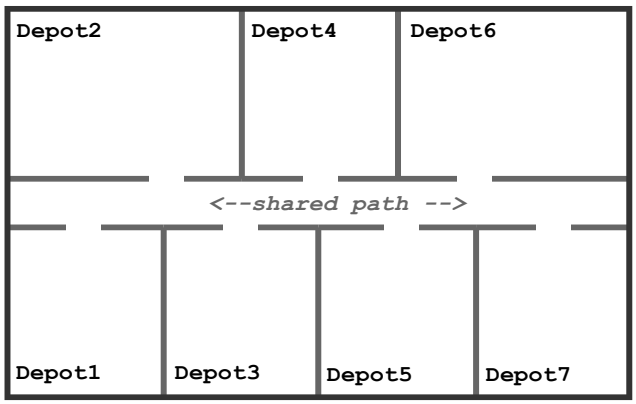
This paper presents realistic planning domains where changes cannot be represented by only instant updates of world states but representation of continuous change is also required. Since existing temporal planners cannot model such planning domains, they fail to find optimal plans for these domains. We propose a continuous model to represent continuous change in complex planning domains. Then, we present TLplan-C as an extension to an existing forward chaining temporal planner: TLplan. TLplan-C uses the new continuous model and applies a backward branching mechanism to schedule actions on the continuous timeline. These properties of the planner ensure optimality of the constructed plan, hence, completeness on the continuous model. Multi-depot multi-vehicle storage domain as a sample planning domain is presented to analyze the performance of TLplan-C.

Introduction

Recently, considerable progress has been made to ensure efficiency of planners in solving more complex problems. Domain definition languages are extended in order to widen the scope of the planning problems. Optimization metrics are defined for the planning problems to evaluate the quality of the constructed plans. Due to the real-world constraints, these metrics are designed to consider both action costs and plan duration. Total duration of the plan is also an important metric. This fact necessitates integration of planning and scheduling.

(Smith *et al.*2000) classifies planners in relation to the applied scheduling methodology in three groups: stratified, interleaved and homogenous planning and scheduling. In planners which applies stratified planning and scheduling, action choices are made first, and then, the resulting plan is scheduled in a separate phase. Crickey3 (Coles *et al.* 2008) uses this strategy through specifying the action choices by a classical planner, and then scheduling the selected actions. Although the planner is guided by the scheduler for selecting schedulable actions in the constructed plan, the final plan is not guaranteed to be optimal. This is due to the fact that makespan optimal plans depend on the selection of actions considering the schedule of these actions. Instead of separately executing planning and scheduling phases, these phases may be *interleaved,* in which essential constraints such as action ordering decisions are made while actions to be executed are chosen. Partial order planners like HSTS (Muscetolla 1993) lxtet (Ghallab & Laruelle1994) use this strategy. In these planners, a plan is constructed by incremental selection of actions together with a set of ordering constraints. Although the planning phase is not totally independent from the scheduling phase, these planners do not maintain absolute state information in search nodes during the search. This prevents solving complex problems where action choices need perfect knowledge of the state information. In homogenous planning and scheduling approach, the whole problem is formulated as a composite planning and scheduling problem, and the entire problem is solved in a uniform way in which the timing of actions is directly considered during planning. Forward chaining temporal planners use this strategy and use the advantage of providing complete state information. TLPlan (Bacchus &Ady 2001) which is an extension of the system introduced by (Bacchus & Kabanza1998) is capable of modeling and solving planning problems with metric quantities and actions with varying durations. Sapa (Do & Kambhampati 2003), another forward chaining planner, uses a similar search procedure to TLPLAN and defines domain independent heuristics to control its search instead of domain specific heuristics and temporal control formulas. In both of these planners, the scheduled time of an action is determined when the action is inserted into plan. Each search node maintains its corresponding world state in which the timing information is also included. The transition to a new world state is provided by applying an action, or advancement of the time value of the world. Forward chaining temporal planners can construct optimal plans while minimizing action costs or total duration (makespan) of the plan.

While integrated planning and scheduling techniques provide improvements on the solution quality, they have poor performance on the realistic problems where changes cannot be represented by only instant updates of the world states. Effects of an action in the real world may involve continuous change which should be considered by the planner in order to construct a valid plan. For instance, in an environment with mobile agents, while choosing a move action, the planner should be aware of time dependent continuous change in the positions of the other mov-



Figue 1: Storage domain

ing agents, in order to come up with a valid plan without collisions.

Some of the earlier studies have investigated planning in domains with continuous linear change. TM-LPSAT (Shin & Davis 2005) as an extension to SAT-based planning framework handles concurrent actions with continuous change. COLIN (Coles et al. 2009) is another effective planner which plans with mixed discrete-continuous numeric changes. However, continuous change in both systems represents increase or decrease of singular values, which is appropriate for modeling consumable resources but not adequate to handle multiple interacting changes such as movements of several agents in a shared path.

In this paper, we introduce planning domains including actions with continuous change effects, and we propose an optimal temporal planner TLPlan-C assisted by a continuous model which is able to represent continuous change. Our approach extends forward chaining temporal planning algorithm to provide completeness and optimality in domains with continuous change. We propose our planner as an extension to TLPlan system and investigate results for an example domain.

In the following sections, we describe the example domain and review the TLPlan approach. We proceed by describing the continuous model and TLPlan-C algorithm. Finally, we discuss optimality of TLPlan-C and present its solution in the example problem and evaluate its plan quality compared to that of TLPlan.

Planning Domains with Continuous Change

In this paper, we focus on solving complex planning problems where the domain includes continuous change that is required to be handled. Some definitions are given here to formulate the overall problem.

Definition 1: (discrete-change) A discrete change is an effect of an action that appears *at-start* or *at-end* of the action and reveals an instant change of the state such as addition/deletion of predicates.

Definition 2: (continuous-time variable) A continuous-time variable y is a variable whose value is updated in continuous time.

Definition 3: (continuous-change) A continuous change refers to a linear change in the value of a specific *continuous-time variable*. It can be expressed by a bounded linear function of time.

Definition 4: (action) An action correspond to a transition of world states and is represented by a condition, several effects and a positive duration.

Definition 5: (discrete-effect action) A discrete-effect action is an action which has only *discrete-change* effects.

Definition 6: (continuous-effect action) A continuous-effect action is an action which has one or more *continuous-change* effects.

Definition 7: (validation rule) *Validation rule* defines a geometrical conflict detection rule about *continuous-time variables* that must hold all over the plan. Implementation of the validation rule depends on the domain definition.

To illustrate an example of a planning domain with *continuous-change effects*, we present the path sharing problem. Figure 1 shows the multi package - multi depot storage domain which consists of several depots connected with a shared narrow corridor which is a one-lane shared path. There are multiple packages to be moved to their destinations in this multi-agent environment. Each agent is capable of picking up/dropping off a single package and moving from one depot to another using the shared path either with or without a package. We assume that agents move on the path with a constant speed that is predefined for each agent in the domain definition. Positions (one dimensional) of the open doors of the depots on the path are also defined in the domain definition.

This example domain includes both *discrete* and *continuous-effect actions*. The shared path (corridor) is the critical resource that should be shared in time among agents in order to find better quality plans. An agent can enter the one-lane shared path and move on the path unless it collides with any other agent concurrently moving on the shared path. *Validation rule* for that scenario is the avoidance of collisions among agents. This rule only applies to *continuous-effect actions*.

The expected solution for this problem is a sequence of precisely scheduled actions that leads the initial state of the world to the goal state, and the optimization metric is the minimization of makespan duration of the constructed plan. Existing temporal planners do not take into consideration of *continuous-effect actions*, and cannot construct optimal plans in such domains.

TLPlan

Forward chaining is an efficient reasoning approach which performs progressive search in the search space. This reasoning mechanism is also used as an efficient search strategy for temporal planning. Although forward chaining

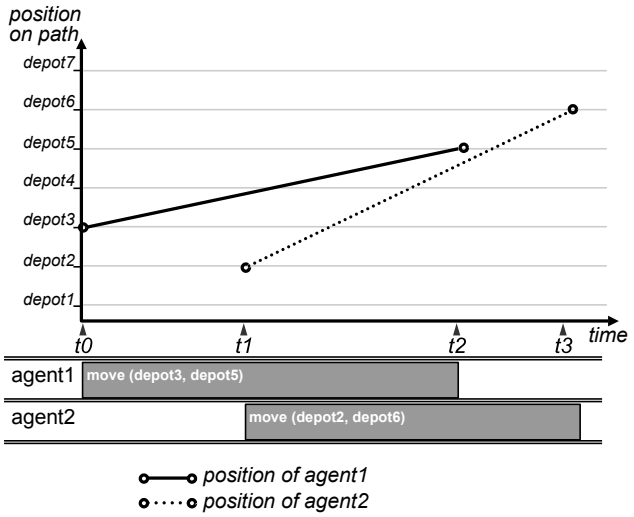


Figure 2: Optimal solution for a sample problem with two agents

approach does not provide a "goal directed" search as in backward chaining, it presents some key advantages. For example, even if a backward-chaining planner starts with a completely described initial world and actions that preserve the completeness of this description, it will still have only incomplete knowledge of the world state at the various points of its search space. Partial order planners also suffer from this problem. The points of their search space are incomplete partially ordered plans, and at the various stages of an incomplete plan there is only a limited knowledge of the state of the world (Bacchus& Kabanza 2000). On the other hand, forward chaining temporal planners have complete information about world states during search, and that information can provide powerful guidance for the search (Bacchus &Ady 2001). This feature of forward chaining is required in problems including actions that highly depend on the state information to match action preconditions and to estimate duration or cost of the actions. For instance, in the storage domain, in order to fire the collision-avoidance rule during the expansion of a new move action in a state, the precise start times of the previously scheduled move actions must be known in advance. Additionally, as a result of complete state information, such planners can support richer planning representations that can model more complex resources (Bacchus & Teh 1998).

To get advantages of forward chaining, our approach is based on TLPLAN, a forward chaining temporal planner originally proposed by (Bacchus &Ady 2001). A search node in TLPlan includes the world state and the action applied along with the *world clock* that defines the start time of that action. In other words applying an action at a state means that this action is scheduled for the *world clock* of that state. The search proceeds by applying new actions or advancing the *world clock* towards finding a complete temporal plan. Note that, the application of an action does not advance the *world clock* but only instant effects of the

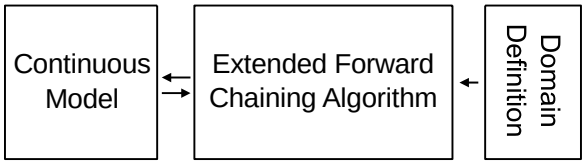


Figure 3: TLPlan-C Architecture

action are applied. The *at-end* effects of that action are set to be applied when the *world clock* is advanced to the ending time of that action. The advance of the *world clock* is provided by a special action which is applicable at any state, and the only effect of that action is setting the *world clock* to the next time point. To avoid infinite branching factor, forward chaining planners restrict available time points to a small set of special time points called *decision epochs* (Cushing et al. 2007) where a previously scheduled action terminates. This idea underlies the fact that a world state does not change at any time point except these *decision epochs*. The suitable time points to schedule a new action are selected only from the set of these *decision epochs* since no change occurs between these points. It has been shown that (Little & Aberdeen 2005) this method is complete and optimal under the assumption that TGP-style actions are used in which two actions do not overlap in any way if an effect or precondition of one is the negation of an effect or precondition of the other. However, as the set of *decision epochs* does not cover continuous timeline but only discrete time points, the planners employing this strategy (e.g, TLPlan), are incomplete if the problem involves *continuous effect actions*. A sample problem instance with two agents is illustrated in figure 2. While the vertical axis shows the positions of the agents on the shared path, the horizontal axis shows the time line. The positions of 7 depot entrances are marked on the vertical axis. The graph illustrates the makespan-optimal plan for executing two move actions by the two agents. The critical factor in getting the optimal plan is scheduling the second move action for time point *t1* (instead of *t2*), which is not included in the standard set of the *decision epochs*. Existing temporal planners cannot construct an optimal plan in such cases. This is due to their lack of *continuous effect action* representations and incapability of scheduling actions for time points outside the decision epoch set.

TLPlan-C

As *decision epoch* temporal planners are not optimal for planning domains including *continuous effect actions*, we propose TLPlan-C, an extended temporal planner to address *continuous changes*. Figure 3 shows the planer architecture. There are two interrelated key features we concentrate on. Firstly, we introduce a continuous model which can represent *continuous effect actions* by *continuous-time variables* and some validation functions to handle interaction between these variables. Secondly, we propose an extended forward chaining temporal planning

algorithm which utilizes this model in order to construct complete and makespan optimal plans.

Continuous Model

As defined earlier, *continuous change* does not represent an instantaneous change but a *continuous change* in the value of a specific *continuous-time variable* during the application of a *continuous effect action*. For instance, the move action in the storage domain introduced earlier is a *continuous effect action* because its effect can be represented by a *continuous-time variable* which denotes one dimensional position of the related agent on the path during its movement.

A domain may include several *continuous-time variables* and each *continuous effect action* affects at least one of these variables. Since *continuous change* represents linear change in the value of a *continuous-time variable*, it can be demonstrated by a linear equation. Let D be a planning domain including n *continuous-time variables*. Let a be a *continuous effect action* of D that is scheduled to start at t_0^i and y_i be a *continuous-time variable* ($i \in \{1,2, \dots ,n\}$) affected by a, m_i be the rate of change, and c_i be the initial value of the variable. Further let Δ_i be the duration of a where t represents time elapsed since the start of the *plan*. Accordingly the change in the value of y_i can be formulated as above.

$$y_i(t) = m_i\left(t - t_0^i\right) + c_i$$
$$t \in [t_0^i\,, t_0^i + \Delta_i]$$

As there may be multiple *continuous effect actions* that affect a *continuous-time variable*, y_i formulation is extended for h actions as the following equation.

$$y_i(t) = \begin{cases} m_{i,0}\left(t - t_0^{i,0}\right) + c_{i,0}\,; & t \in [t_0^{i,0}\,, t_0^{i,0} + \Delta_{i,0}] \\ m_{i,1}\left(t - t_0^{i,1}\right) + c_{i,1}\,; & t \in [t_0^{i,1}\,, t_0^{i,1} + \Delta_{i,1}] \\ \vdots & \\ m_{i,h}\left(t - t_0^{i,k}\right) + c_{i,k}\,; & t \in [t_0^{i,k}\,, t_0^{i,k} + \Delta_{i,h}] \end{cases}$$

The continuous-time variable y_i could be represented as a set of line segments where a line segment is modeled by a tuple $f_{i,j} = \langle\, c_{i,j}\,,\, m_{i,j}\,,\, \Delta_{i,j}\,,\, t_0^{i,j}\,\rangle$, and $f_{i,j}$ defines y_i in the interval of $[\, t_0^{i,j},\, (t_0^{i,j} + \Delta_{i,j})\,]$. In the planning domain, $f_{i,j}$ corresponds to a scheduled *continuous-effect action* $a_{i,j}$. The terms $c_{i,j}$, $m_{i,j}$ *and* $\Delta_{i,j}$ are derived from the action definition and its parameters and $t_0^{i,j}$ is assigned by the planner. Based on this notation, before it is scheduled by the planner, a *continuous effect action* $a_{i,j}$, is represented by $k_{i,j}$, where $k_{i,j}$ is a tuple $\langle\, c_{i,j}\,,\, m_{i,j}\,,\, \Delta_{i,j}\,\rangle$. Since its value is determined during the scheduling of that action, $t_0^{i,}$ is not included in $k_{i,j}$.

A validation step is performed to eliminate the conflicts of different *continuous-effect actions* which affect a *continuous-time variable.* The rule for the validity depends on the context of the *continuous-time variables*. For the path sharing scenario, the corresponding *validation rule* checks whether a pair of line segments intersects. Let validation rule V be a first order formula where;

$$\mathrm{V}_{i,j,l,k} = \{true\ if\ f_{i,j}\ and\ f_{k,l}\ do\ not\ intersect\}$$

Further let F ($f_{i,j} \in F$) be the set of all scheduled *continuous changes* of all *continuous time variables*. Validation of the set F is expressed by function *valid* where;

$$valid(F) \;\Leftrightarrow \bigwedge_{\substack{i,j,k,l \\ i \neq k}} \mathrm{V}_{i,j,k,l}$$

Finally let S_{min} be a special function to find the earliest possible start time $t_0^{i,j}$, for the action $k_{i,j}$ preserving validity of the set of current variables F.

$$S_{min}\left(k_{i,j}\right) = arg\,min\{\, \mathrm{t_o^i} \mid valid\left(\{\langle\, c_i\,, m_i\,, \Delta_i\,, t_o^i\,\rangle\} \cup F\right)\}$$

A sample illustration of a *continuous-effect action* in the storage domain can be given as follows. Positions of each agents are represented by individual *continuous-time variables*. Move action definition is given as:

$$\mathrm{action\_move}(\,?\mathrm{agent}, ?\,\mathrm{from}, ?\,\mathrm{to})$$

where *?agent* , *?from* and ?to denotes an agent, the initial depot of the agent and the destination depot respectively. The position of $agent_i$ is represented by a *continuous-time variable* y_i, *loc* is a predefined function that returns the position of the related entrance from depot to the path, and *speed* is another predefined function that returns the speed of an agent. Before it is scheduled by the planner, a move action for $agent_i$ is represented with $k_{i,j}$ which denotes the related *continuous change*. According to the given action definition, $k_{i,j}$, is given below.

$$k_{i,j} = \langle\, c_{i,j}\,, m_{i,j}\,, \Delta_{i,j}\,\rangle$$

$$\mathrm{m_{i,j}} = \begin{cases} \mathrm{speed(?agent)}\,; & loc(?\,from) < loc(?\,to) \\ \mathrm{-speed(?agent)}\,; & loc(?\,from) > loc(?\,to) \end{cases}$$

$$\Delta_{i,j} = |loc(?\,\mathrm{from}) - loc(?\,\mathrm{to})|\,/speed(?\,\mathrm{agent})$$

$$c_{i,j} = loc(?\,from)$$

Note that, $\mathrm{m_{i,j}}$ takes a positive or negative value depending on the direction of the movement. In order to generate a valid plan, when a new *continuous effect action* is being scheduled, its validity with F (the set of previously scheduled move actions) must be checked. In this domain, *valid(F)* satisfies that no two pair of the scheduled actions for different agents intersect with each other resulting in a collision-free plan. In other words, as the variables $y_i(t)$ and $y_j(t)$ defines positions of any two agents, no t exists where $y_i(t) = y_j(t)$. Whenever *valid(F)* is satisfied, the planner may select *action_move* to be scheduled for $t_0^{i,j}$, if the related $\langle\, c_{i,j}\,, m_{i,j}\,, \Delta_{i,j}\,, t_0^{i,j}\,\rangle$ preserves the validity of F.

Extended Forward Chaining Algorithm

TLPlan-C is an extended forward chaining temporal planner which uses the new continuous model to handle *continuous change* in planning problems. The base temporal

planning algorithm is adapted from TLPlan which runs forward chaining search on the action space. Initially, L, the list of successor states includes only the initial state and F, the set of scheduled *continuous changes* is empty. At each step, the state *s* with the lowest cost is selected within the successor list *L*. Then, for each action of which preconditions are met at state *s*, a new successor state s^+ is created and inserted into *L*. Each state *s* is associated with a timestamp which denotes the actual time the state will occur during the execution of the plan. Additionally, each state maintains an event queue to hold *at-end* effects of previously scheduled actions, to be applied in future planning steps.

Algorithm1 presents the main flow of TLPlan-C. A_d is set of *discrete effect actions* where A_c is the set of *continuous effect actions*. The first iteration in the algorithm handles *discrete effect actions* (A_d). This part of the algorithm is almost identical to TLPlan. A *discrete effect action a* is applied to state *s* to create a successor state if its preconditions are met at state *s*. To apply an action, planner generates a new successor state *s+* which inherits the state information, timestamp and event queue of *s*. Standard actions do not advance the world clock, but only instant effects of action *a* are applied to state s^+ and *at-end* effects of *action a* are inserted into the event queue of s^+. In addition to standard actions, there is one special action: the unqueue-event action, which advances the world clock. This action moves time forward to the next scheduled point in time and applies all *at-end* effects that are scheduled for that time.

The second iteration in the algorithm handles *continuous effect actions*. The critical factor for planning with *continuous effect actions* is continuous treatment of time. If only a limited set of discrete time steps (ie. *decision epochs*) are considered to schedule actions, it is not guaranteed to find optimal plans.

TLPlan-C employs an extended branching scheme to cover continuous time by relaxing the constraint that each action must start at only *decision epochs*. The function S_{min} calculates the earliest possible start time for a *continuous effect action*, considering a tuple *k* and the set *F*. During planning, while expanding a state *s* with applying *continuous effect action a*, successor state s^+ with timestamp t_0 and action *a* is generated if preconditions of *a* are met at t_0, where t_0 is the earliest possible start time calculated by S_{min} function. Note that t_0 is a point in continuous time, independent from decision epochs. However, calculation of t_0 may depend on other *continuous effect action* choices that appear after t_0. Therefore a backward branching mechanism is adopted in order to handle states with timestamp t_0 that are released later than t_0. Before branching to an earlier state, it must be checked that whether the discrete preconditions of action *a* are also satisfied at t_0. The special function *evalAt* checks the preconditions of *action a* at the first predecessor state with the timestamp earlier than t_0. This predecessor state has the same state information with the potential new state at t_0 due to the fact that no *discrete change* appears between any two sequential timestamps. If

```
Plan (<L,F> ,Ad, Ac, Goal)
if (s⊨ Goal and s.Q={})
  return(s)
else
  s:=minCost(L)
  for all a | a ∈ Ad and s⊨pre(a)
    s+:=s
    s+.prev:=s
    if a != unqueue-event
      s+:=ApplyInstantEffects(s,a)
      s+.Q:=AddDelayedEvents(s,a)
    else
      newTime:=eventTime(front(s.Q))
      s+.time:=newTime
      while eventTime(front(s+.Q))== newTime
        e:=removeFront(s+.Q)
        s+:= ApplyEffect(s+,e)
       L.insert(s+)

  for all a | a ∈ Ac
    t0:= Smin(a.k,F)
    if evalAt(t0,pre(a)) and t0 < s.time
      s+:= predecessor(t0,s)
      s+.time:=t0
      s+:=ApplyInstantEffects(s,a)
      s+.Q:=AddDelayedEvents(s,a)
      F.insert(a.k, s+.time)
      L.insert(s+)
    else if s⊨pre(a) and valid(F, a.k, s.time)
      s+:=s
      s+.prev:=s
      s+:=ApplyInstantEffects(s,a)
      s+.Queue:=AddDelayedEvents(s,a)
      F.insert(a.k, s+.time)
      L.insert(s+)
  Plan (<L,F> ,Ad, Ac, Goal)
```

Algorithm1: TLPlan-C algorithm

the preconditions are met, this predecessor state of *s* is duplicated as s^+ using the predecessor function, and timestamp of s^+ is set to t_0. After generation of s^+, the instant effects of action *a* are applied to s^+ and at-end effects are inserted to the event queue of s^+. Since the applied action is a *continuous-effect action*, the *continuous change* related with that action (represented by the tuple *f*) is inserted into *F*. Note that, the backward branching does not prune the state *s* but only generates s^+ as a successor of earlier states.

C*ontinuous effect actions* have both discrete preconditions and *continuous-change* constraints. Although *continuous change* constraints are satisfied at timestamp t_0, it might be the case that the discrete preconditions are not met at that time. In that case, the next *decision epoch*, in which discrete preconditions are satisfied, is selected. In general, a *continuous effect action a*, that cannot be applied at t_0 is scheduled for a discrete timestamp where its pre-

```
(define (initial-state)
   (largepackage pack1)
   (smallpackage pack2)
   (smallpackage pack3)

   (large-carrier agent1)
   (small-carrier agent2)
   (small-carrier agent3)

   (at agent1 depot4)
   (at agent2 depot2)
   (at agent3 depot1)

   (in pack1 depot6)
   (in pack2 depot7)
   (in pack3 depot3)

   (= (speed agent1) 0.25)
   (= (speed agent2) 0.90)
   (= (speed agent3) 0.36)
)

(define (goal)
   (in pack1 depot5)
   (in pack2 depot4)
   (in pack3 depot1)
)
```

Figure 4: Problem definition

conditions are met and the validity of *continuous-time variables* are preserved.

Optimality

Makespan optimal plan for a problem involves the actions scheduled for their earliest possible times to minimize the total duration of the plan. TLPlan-C as an extended forward chaining temporal planner, searches for every possible permutation of actions and considers continuous time so that each action starts at the earliest possible start time considering both discrete and continuous constraints of that action. That is, the start times of actions are not limited to a small set of decision epochs. Therefore, the optimality of the constructed plan is guaranteed.

An Example Scenario

The introduced storage domain involves packages to be transported from their initial depots to destination depots. Clearly, the agents should move on the shared path to go from one depot to another. During the movement of an agent on the shared path, its position on the path changes continuously in time and the rate of that change is defined as the speed of that agent.

To exemplify how TLPlan-C works, we use a sample problem instance and analyze the constructed final plan for this problem. In the example problem, there are three agents from two different agent types. *Agent1* is a large carrier, and can pick both large and small packages. *Agent2* and *Agent3* are small carriers and they can only pick small packages. There are two small and a large package in this domain to be transported from one depot to another. The sample problem formulation including the initial and goal state definitions is shown in figure 4. A move action should be executed for each object to reach the goal state. Duration of pickup and release actions is assumed to be constant. However, duration of a move action depends on the distance between initial and final locations and the speed of the agent executing the action.

Figure 5 presents the constructed plan for this problem by TLPlan as a decision epoch planner. As the results illustrate, although there is a better plan, TLPlan constructs a suboptimal one. The decision epoch planning approach has two interrelated drawbacks in solving such a problem. Firstly, it does not provide continuous treatment of time. Secondly, it is not capable of modeling *continuous change* in a planning domain. These properties of decision epoch planners cause their final plans to be longer than the optimal plan. This is due to their simple mutex rules to avoid mutually conflicting situations (e.g., collisions in the example domain). According to this inadequate rule, in the example domain, a path can be used by only a single agent during its movement. The other agents are not allowed in the path (i.e., the move actions of these agents are not scheduled) during this time which causes delays in the total plan. Since TLPlan is restricted to *discrete changes* in the planning domain, it does not ensure simultaneous use of the shared path. Although all possible permutations of actions are explored, the action start times are selected from the set of decision epochs. As a result, the optimal solution cannot be found.

On the other hand, TLPlan-C uses the advantage of both its continuous model representation and the additional branching scheme performed during the search. Therefore, the resulting plan allows multiple agents to share the path concurrently whenever a collision-free path-sharing is possible.

Figure 6 depicts the constructed plan by TLPlan-C. The graph in the figure plots the positions of agents on the path during their move. The horizontal axis indicates the time elapsed since the start of the plan. The vertical axis indicates the positions of the agents on the shared path. The positions of 7 depot entrances are marked on the vertical axis. The continuous model representation of TLPlan-C and the extended branching scheme using this model enables concurrent move actions to be scheduled in a collision-free setting. Not only the decision epochs but also the intermediate points in time (e.g., *t,1 t2* and *t5*) are considered for scheduling actions to reach the optimal solution. These points are derived by the continuous model, and inserted into the search tree by backward branching method. Through this extension, TLPlan-C can schedule actions for the earliest possible points in the timeline. Consequently, the makespan-optimal plan is found by the best first search mechanism. The numerical analysis of both approaches for different performance measures is given in table 1. The results show that TLPlan-C finds the optimal plan and solves the problem more efficiently than TLPlan. TLPlan evaluates all permutations of actions in a longer

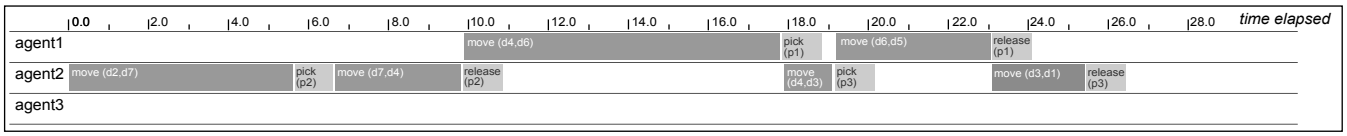


Figure 5: Plan constructed by TLPlan

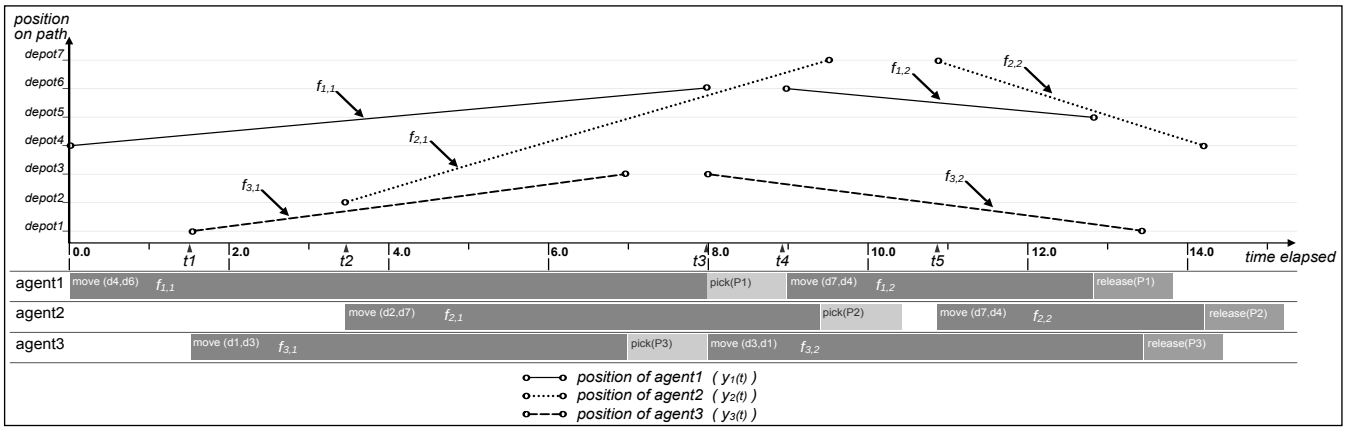


Figure 6: Optimal plan constructed by TLPlan-C

time than that of TLPlan-C. This is due to the deeper search tree expanded in TLPlan, whereas, TLPlan-C finds the optimal solution at a shallower depth, before expanding deeper nodes with makespan cost higher than the optimal cost.

	TLPlan	TLPlan-C
number of nodes generated	5341	704
number of nodes searched	3268	557
CPU time (sec)	0.665	0.140
duration of constructed plan	26.3	15.2
number of actions	12	12

Table 1: Results for the Storage Domain

Conclusion

In this paper, we introduce realistic planning problems including both *discrete/continuous effect actions*. We show that existing temporal planning algorithms are not complete for these problems. Then, we present TLPlan-C as an extension to an existing forward chaining temporal planner: TLPlan. We propose a continuous model to represent *continuous change* and to be used with the extended temporal planner. A backward branching mechanism is integrated into TLPlan-C to ensure completeness on the continuous model. This new extended algorithm finds makespan-optimal solutions. Due to the complete state information maintained by the algorithm, this new approach offers the same advantage of the easy implementation of the heuristic guidance as standard forward chaining planners. The future work includes investigation of special pruning techniques for temporally identical states and use of heuristic guidance for makespan estimation to reduce the complexity.

References

Bacchus, F.; and Teh, Y. W. 1998. Making forward chaining relevant. In *Proc. AIPS,* volume 4, 54–61. AAAI Press.

Bacchus, F. ; and Kabanza. F. 1998 Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22(1-2):5–27.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.

Bacchus, F.; Ady, M. 2001. Planning with resources and concurrency: A forward chaining approach. *Int. Joint Conf. on AI.*

Coles, A. I.; Fox, M.; Long D.; and Smith, A. J.; 2008 Planning with problems requiring temporal coordination. In *Proc. 23rd Nat. Conf. on AI* (AAAI).

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009. Temporal planning in domains with linear processes. In *Proc. of IJCAI 09.*

Cushing,W., Kambhampati, S.; Mausam M.; and Weld, D. 2007. When is temporal planning really temporal?. In *Proc. of IJCAI 07*

Do, M. B.; and Kambhampati, S. 2003 SAPA: A multi-objective metric temporal planner. *JAIR*, 20:155–194

Ghallab, M.; and Laruelle,H. 1994 Representation and control in lxTeT: a temporal planner. In *Proceedings of second International Conference on Artificial Intelligence Planning Systems* 61-67

Little, I.; Aberdeen, D.; and Thiebaux, S. 2005. Prottle: A probabilistic temporal planner. In *AAAI*'05.

Muscetolla, N.; 1993 HSTS Integrating Planning and Scheduling. In *Intelligent Scheduling* Morgan Kaufmann.

Smith, D.E.; Frank, J.; and Jonsson A.K. 2000 Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1),

Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial Intelligence* 166(1-2):194–253.