Automated Planning in the Continuous World

October 24, 2025

Mikhail Soutchanski (Toronto Metropolitan University, Canada)
and Derek Long (Schlumberger Cambridge Research, UK)

Planning in Continuous Incompletely Known World

Problem: How can a computer with its discrete actions achieve goals in the continuous world where not all objects or facts are known?

Mixed discrete-continuous systems: instantaneous transitions between states due to discrete actions and events, and within each state there is a continuous change. States have *relational* structure: go beyond hybrid automata. Think of Data Bases with unknowns.

In general, the planning problem is undecidable already for simple hybrid automata. Our goal: Develop sound algorithms that can compute sometimes plans approximating optimization objectives.

Example of Hybrid Relational Systems: How to integrate Task and Motion Planning (TAMP) in Robotics.

(1) *Theorem proving* (deductive) approach to *lifted* planning, where search is done over *situation tree*, but not over the state space.

(2) Developed an efficient general planner NEAT that uses a new non-trivial domain independent heuristic.

(3) Our NEAT planner demonstrates competitive performance on popular benchmarks from King's College, London, UK (developed in Maria Fox and Derek Long's research group).

Mixed Discrete-Continuous Systems: SC + Timeline

- Each action is instantaneous - situations have unique start time.
- Reiter's book: $start(S_0) = 0$ and $start(do(A, S)) = time(A, T)$.
- Planner does search over sequences of actions (situations).
- The state space remains implicit, states are not saved in memory.

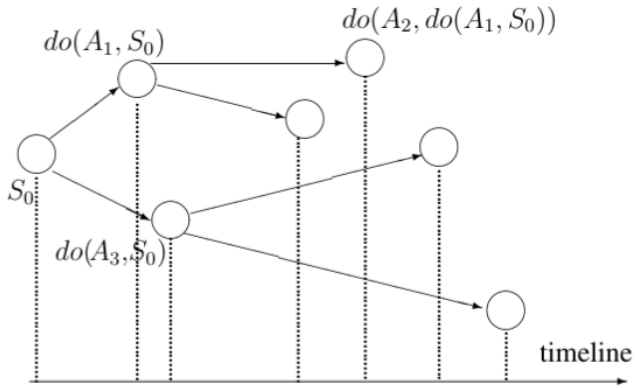


Figure: Situations are aligned with moments on a time line: next situation $do(A, S)$ starts at the moment T when an action A is executed in S.

Example: Bouncing Balls

Consider a finite number of balls that can be dropped and that can elastically bounce from the floor. Ignore friction.
Agent actions: *drop*$(b, time)$ and *catch*$(b, time)$.
Natural events: *bounce*$(b, time)$ - ball b hits the floor, and *atPeak*$(b, time)$ - ball is at the top point of its trajectory.

The atemporal fluent *falling*(b, s) means the ball b is falling down and accelerating under the Earth gravity.

The atemporal fluent *flying*(b, s) means the ball b bounced, it is flying up in situation s and decelerating due to gravity.

The vertical axis is oriented downwards, i.e., if a ball is falling down, then its speed is positive and increases. But when the ball bounces, its speed is negative and decreases.

Consider functional temporal fluent *distance*(b, t, s) that represents the ball b's height at the moment of time t within s, and functional temporal fluent *velocity*(b, t, s) that characterizes instantaneous velocity of b at the moment t within the time interval as long as situation s lasts.

These temporal fluents describe time dependent change within situation, in between two occurrences of the agent actions and/or the natural events.

This approach extends Reiter's Basic Action Theories: atemporal fluents only.

SSAs for Atemporal Fluents

Basic Action Theory (BAT) $\mathcal{D}$ includes preconditions $\mathcal{D}_{ap}$, successor state axioms (SSA) $\mathcal{D}_{ss}$, initial axioms $\mathcal{D}_{S_0}$, foundational axioms Σ, unique name axioms $\mathcal{D}_{una}$. First, consider SSAs. There are 3 contexts:

1. one where the ball is at rest,
2. one where it is falling down, and
3. one where the ball is flying up after it bounced.

$$(\forall a \forall s \forall b).\ falling(b, do(a, s)) \leftrightarrow \exists t(a = drop(b, t)) \vee \exists t(a = atPeak(b, t)) \vee falling(b, s)) \wedge (\neg\exists t(a = catch(b, t)) \wedge \neg\exists t(a = bounce(b, t))$$

$$(\forall a \forall s \forall b).\ flying(b, do(a, s)) \leftrightarrow \exists t(a = bounce(b, t)) \vee flying(b, s)) \wedge \neg\exists t(a = catch(b, t)) \wedge \neg\exists t(a = atPeak(b, t))$$

In a general case, there are finitely many (parameterized) context types which are pairwise mutually exclusive.

Important: even if situation does not change, temporal fluents change with time within situation that can last for an interval of time.

Actions only change the context, thereby switching between the continuous trajectories that the ball can follow. Each context determines its own continuous function of time how a physical quantity changes.

Preconditions: Logical and Numeric Constraints

$$\forall t \forall s \forall b.\ poss(drop(b, t), s) \leftrightarrow ball(b) \wedge \neg falling(b, s) \wedge \neg flying(b, s) \wedge \underline{t \geq start(s)}.$$

The agent action $drop(b, t)$ is possible in s at the moment of time t, if a ball b is neither falling, nor flying in s, and the moment of time $start(s)$ when s started is $\leq t$. (Due to $\forall t$ the branching factor is infinite for a planner.)

In an implementation, the temporal constraint $t \geq start(s)$ is added to a special data structure for a constraint store to be evaluated later when the planner checks whether the goal numerical conditions are satisfied.

$$\forall s \forall t \forall b.\ poss(catch(b, t), s) \leftrightarrow ball(b) \wedge (falling(b, s) \vee flying(b, s)) \wedge \underline{t \geq start(s)}.$$

$$\forall s \forall t \forall b.\ poss(bounce(b, t), s) \leftrightarrow ball(b) \wedge falling(b, s) \wedge \underline{distance(b, t, s) = 0} \wedge \underline{velocity(b, t, s) \geq \epsilon} \wedge \underline{t \geq start(s)}.$$

In an implementation, use external Non-Linear Programming (NLP) solver to deal with the numerical constraints. An action can be possible only if the numeric constraints are feasible.

$$\forall s \forall t \forall b.\ poss(atPeak(b, t), s) \leftrightarrow ball(b) \wedge \underline{distance(b, t, s) \geq 0} \wedge \underline{velocity(b, t, s) = 0} \wedge flying(b, s) \wedge \underline{t \geq start(s)}.$$

The last axiom is saying that a natural event $atPeak(b, t)$ can occur in s at the moment of time t if the ball b is flying up in s so that it reached its highest point at which its velocity is 0, but its hight is positive.

SSAs to Initialize Temporal Fluents

When actions occur, the temporal change can be either continuous, or there might be jumps or resets in the values of temporal fluents.

To describe transitions in temporal fluents due to actions when new situation starts, use auxiliary functional fluents $init_{dist}(b, d, s)$ and $init_{vel}(b, v, s)$.

Fluent $init_{dist}$
The height of the ball changes continuously, no matter what actions happen.

$$(\forall s \forall y \forall a \forall b).\ init_{dist}(b, do(a, s)) = y \leftrightarrow \underline{distance(time(a), s) = y}$$

Fluent $init_{vel}$

The velocity y of the ball b resets to 0, when the agent catches the ball. When the ball bounces, its velocity jumps to the quantity with the opposite sign. All other actions with any other balls have no effect on these physical quantities at the moment when new situation starts.

$$\forall s \forall y \forall a \forall b.\ init_{vel}(b, do(a, s)) = y \leftrightarrow \exists y_0.y_0 = velocity(time(a), s) \wedge \exists t(a = catch(b, t) \wedge y = 0) \vee \exists t(a = bounce(b, t) \wedge y = -y_0) \vee (\neg\exists t(a = bounce(b, t)) \wedge \neg\exists t(a = catch(b, t)) \wedge y = y_0).$$

State Evolution Axioms (SEA): Continuous Change

Each SEA characterizes how temporal fluent changes with time within a context determined by atemporal fluents. (The gravity acceleration is 9.81)

$$(\forall s \forall t \forall b).\ distance(b, t, s) = y \leftrightarrow \exists y_0.y_0 = init_{dist}(b, s) \wedge (\neg falling(b, s) \wedge \neg flying(b, s) \wedge y = y_0) \vee \underline{(falling(b, s) \wedge y = y_0 - \int_{start(s)}^{t} (9.81 \cdot x)\ dx)} \vee \underline{(flying(b, s) \wedge y = y_0 + \int_{start(s)}^{t} (9.81 \cdot x)\ dx)}.$$

$$(\forall s \forall t \forall b).velocity(b, t, s) = y \leftrightarrow \exists y_0.y_0 = init_{vel}(b, s) \wedge (\neg falling(b, s) \wedge \neg flying(b, s) \wedge y = y_0) \vee \underline{(falling(b, s) \wedge y = y_0 + \int_{start(s)}^{t} 9.81 dx)} \vee \underline{(flying(b, s) \wedge y = y_0 - \int_{start(s)}^{t} 9.81 dx)}.$$

In an implementation, collect all (underlined) numerical constraints in a data structure. Postpone evaluation until the planner checks if s is a goal state.

In a simplified implementation we assumed that the balls move along straight lines instead of physically correct quadratic trajectories. Then, equations for both height and velocity are linear wrt time.
=> can use the Linear Programming *eplex* library as an external solver .

If the planning objective is also a linear function of its arguments, then optimization reduces to solving the Linear Programming (LP) problem.

Foundational Axioms for Situations and Time

All the following axioms have straightforward implementation.

Foundational Axioms from Chapters 4 and 7 of Ray Reiter's book [2001].

$\forall a_1 \forall a_2 \forall s_1 \forall s_2 . do(a_1, s_1) = do(a_2, s_2) \rightarrow a_1 = a_2 \wedge s_1 = s_2$

$\forall s . \neg(s \sqsubset S_0)$

$\forall a \forall s \forall s' . s \sqsubset do(a, s') \leftrightarrow s \sqsubseteq s'$, where $s \sqsubseteq s'$ means $(s \sqsubset s' \vee s = s')$

$\forall P . (P(S_0) \wedge \forall a \forall s (P(s) \rightarrow P(do(a, s)))) \rightarrow \forall s P(s)$

$\forall a, s' . do(a, s') \sqsubseteq s \rightarrow (poss(a, s') \wedge start(s') \leq time(a)) \wedge$
$\forall a' (poss(a', s) \wedge natural(a') \wedge a \neq a' \rightarrow time(a') \leq time(a))$

$\forall a .\ start(do(a, s)) = time(a) \quad start(S_0) = 0$

Domain Specific Axioms for the Bouncing Ball Example

$\forall t, \forall b . time(drop(b, t)) = t$ $\quad$ $\forall t, \forall b . time(catch(b, t)) = t$

$\forall t, \forall b . time(bounce(b, t)) = t$ $\quad$ $\forall t, \forall b . time(atPeak(b, t)) = t$.

$\forall t \forall b . natural(atPeak(b, t))$ $\quad$ $\forall t \forall b . natural(bounce(b, t))$.

$\forall t \forall b . agent(drop(b, t))$ $\quad$ $\forall t \forall b . agent(catch(b, t))$.

Initial Theory

$ball(b1)$ $\quad$ $ball(b2)$

$velocity(b1, 0, S_0) = 0$ $\quad$ $velocity(b2, 0, S_0) = 0$

$distance(b1, 0, S_0) = 100$ $\quad$ $distance(b2, 0, S_0) = 150$

In a PROLOG program, functional fluent $distance(b, t, s)$ is implemented as predicate $dist(b, d, t, s)$, and functional fluent $velocity(b, t, s)$ is implemented as predicate $vel(b, v, t, s)$. Program is available at *https://www.cs.torontomu.ca/mes/publications/*

Temporal Change Axiom (TCA) in a General Case

Our starting point is a *temporal change axiom* (TCA) which describes the evolution of a particular temporal fluent due to the passage of time in a particular context of an arbitrary situation: Similar to $vel(b, t, s)$.

$$\gamma(\bar{x}, s) \wedge \delta(\bar{x}, y, t, s) \rightarrow f(\bar{x}, t, s) = y, \tag{1}$$

where $t, s, \bar{x}, y$ are variables and $\gamma(\bar{x}, s)$, $\delta(\bar{x}, y, t, s)$ are formulas uniform in s. We call $\gamma(\bar{x}, s)$ the *context* as it specifies the condition under which formula $\delta(\bar{x}, y, t, s)$ provides (may be implicitly) the value y to fluent f at time t.

$$\gamma(\bar{x}, s) \rightarrow \exists y\, \delta(\bar{x}, y, t, s). \tag{2}$$

For each TCA, we require that whatever the circumstance, the axiom supplies a value for the quantity modelled by f if its context is satisfied.

A finite set of k temporal change axioms for fluent f can be equivalently expressed as follows, where $\Phi(\bar{x}, y, t, s)$ is $\bigvee_{1 \leq i \leq k} (\gamma_i(\bar{x}, s) \wedge \delta_i(\bar{x}, y, t, s))$.

$$\Phi(\bar{x}, y, t, s) \rightarrow f(\bar{x}, t, s) = y \tag{3}$$

$$\Phi(\bar{x}, y, t, s) \wedge \Phi(\bar{x}, y', t, s) \rightarrow y = y'. \tag{4}$$

Condition (4) guarantees the consistency of the axiom (3) by preventing a continuous quantity from having more than one value at any moment of time. With condition (4), all contexts in the given set of TCA are pairwise mutually exclusive wrt a BAT $\mathcal{D}$. Note contexts $\gamma(\bar{x}, s)$ are *time-independent*.

Planning Problem for the Two Balls

Objective: find a plan that satisfies a goal in minimal time wrt constraints.

Find *the earliest moment of time* such that each ball reached its peak at least once, both balls are falling, the velocities of the two balls are equal, and their heights are also equal. Minimization wrt constraints collected so far.

Checking these goal conditions reduces to the linear programming problem that can be solved using an external *eplex* LP solver interfaced with our program. Solved this planning instance with an uninformed iterative deepening depth-first search (DFS) planner.

The program found a correct 8 step plan in 0.18 seconds: *[drop(b2, 0), bounce(b2, 30.581039755351682), drop(b1, 50.9683995922528), atPeak(b2, 61.162079510703364), catch(b2, 71.35575942915392), bounce(b1, 71.35575942915392), drop(b2, 91.743119266055047), atPeak(b1, 91.743119266055047)].*

Notice that this plan must be clever since the two balls had different initial heights: *dist(b1,100,0,[]). dist(b2,150,0,[]).*
But in the goal state their heights and velocities must be equal.

In an implementation, precondition axioms for nature's actions appear before preconditions for agent's actions. In a general case, need extra efforts to make sure nature's actions are executed as soon as they are possible.

Deriving State Evolution Axioms

Having combined all laws which govern the evolution of f with time into a single axiom (3), we can make a causal completeness assumption (Explanation Closure): *there are no other conditions under which the value of f can change in s from its initial value at $start(s)$ as a function of t*, i.e.,

$$f(\bar{x}, t, s) \neq f(\bar{x}, start(s), s) \rightarrow \exists y\, \Phi(\bar{x}, y, t, s). \tag{5}$$

Theorem

Let for each formula of the form (1) the background theory $\mathcal{D}$ entail $\forall(\gamma(\bar{x}, s) \rightarrow \exists y\, \delta(\bar{x}, y, t, s))$. Then the conjunction of axioms (1), (3), (4), (5) is logically equivalent to

$$\begin{aligned} f(\bar{x}, t, s) = y \leftrightarrow [&\Phi(\bar{x}, y, t, s) \vee \\ &y = f(\bar{x}, start(s), s) \wedge \neg\Psi(\bar{x}, y, t, s)], \end{aligned} \tag{6}$$

where $\Psi(\bar{x}, s)$ denotes $\bigvee_{1 \leq i \leq k} \gamma_i(\bar{x}, s)$.

We call the formula (6) a *state evolution axiom* (SEA) for the fluent f. Note what the SEA says: f evolves with time during s according to some law whose context is realized in s or stays constant if no context is realized.

See proof in the paper Vitaliy Batusov, Giuseppe De Giacomo, Mikhail Soutchanski, "Hybrid Temporal Situation Calculus", pages 11-13, https://arxiv.org/abs/1807.04861

Temporal Basic Action Theory (TBAT): HTSC

The SEA for a temporal fluent f does not completely specify the behaviour of f because it talks only about change within s. Need a SSA describing how the initial value of f changes (or does not change) when an action is performed.

How to relate $f(\bar{x}, time(a), do(a,s))$ with $f(\bar{x}, time(a), s)$? Enforce "=" or not? Transition is not always continuous, e.g., object's acceleration changes from 0 to $-9.8m/s^2$ when an object is dropped. Need ability to model action-induced discontinuous jumps in the values of the continuously varying quantities.

For each temporal functional fluent $f(\bar{x}, t, s)$, we introduce an *auxiliary atemporal* functional fluent $f_{init}(\bar{x}, s)$ whose value in s represents the value of the temporal fluent f in s at the time instant $start(s)$. Add new SSA for f_{init}:

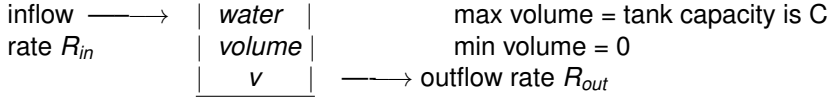
$$f_{init}(\bar{x}, do(a,s)) = y \leftrightarrow \exists y'. f(\bar{x}, time(a), s) = y' \wedge Init(\bar{x}, y', y, a, s), \quad (7)$$

where $Init(\bar{x}, y', y, a, s)$ is a formula uniform in s whose purpose is to describe how the initial value y of f_{init} in $do(a,s)$ relates to the temporal fluent f value y' at the same time instant in s (i.e., prior to execution of action a).

To establish the relationship between temporal fluents and their atemporal *init*-counterparts, we require $\mathcal{D}_{ss} \wedge \mathcal{D}_{se} \models f(\bar{x}, start(s), s) = f_{init}(\bar{x}, s)$.

A *temporal basic action theory* is $\mathcal{D}_T = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{se}$ such that $\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ constitutes a BAT, and $\mathcal{D}_{se}$ is a set of state evolution axioms. This is new **Hybrid Temporal Situation Calculus (HTSC)**

Example 2: a Water Reservoir



Consider a water reservoir with an adjustable inflow and an adjustable outflow. Let the temporal functional fluent $vol(t,s)$ represent the volume of water in the tank at time t. The maximum capacity of the tank is C. Let R_{in} and R_{out} be inflow and outflow rates (volume per unit time), which are for simplicity are constant, i.e., rates are time-invariant.

Let actions $startIn(t)$, $endIn(t)$ represent opening/closing of the inflow valve. These actions initiate/terminate the process represented as the fluent $inflow(s)$. Let actions $startOut(t)$, $endOut(t)$ represent opening/closing of the outflow valve. These actions initiate/terminate the process $outflow(s)$.

$Poss(startIn(t), s) \leftrightarrow \neg inflow(s) \wedge t \geq start(s) \wedge (vol(t,s) < C)$.
$Poss(endIn(t), s) \leftrightarrow inflow(s) \wedge t \geq start(s)$.
$Poss(startOut(t), s) \leftrightarrow \neg outflow(s) \wedge t \geq start(s) \wedge (vol(t,s) > 0)$.
$Poss(endOut(t), s) \leftrightarrow outflow(s) \wedge t \geq start(s)$.

$inflow(do(a,s)) \leftrightarrow \exists t(a = startIn(t)) \vee inflow(s) \wedge \neg\exists t(a = endIn(t))$
$outflow(do(a,s)) \leftrightarrow \exists t(a = startOut(t)) \vee outflow(s) \wedge \neg\exists t(a = endOut(t))$

Reservoir: a State Evolution Axiom (SEA) for Volume

An obvious initial value SSA asserts the continuity of volume (no leaks):

$$vol_{init}(do(a,s)) = v \leftrightarrow vol(time(a), s) = v.$$

To write a State Evolution Axiom for the fluent $vol(t,s)$ consider all possible combinations of inflow and outflow: either the inflow valve is open while outflow is present or not, or the inflow valve is closed while the outflow valve can be open or closed. Each combination is a separate context. For each context, $vol(t,s)$ evolves according to a different function of flow rates and time. Inflow cannot exceed capacity C, and outflow cannot yield $vol(t,s) < 0$.

$$\begin{aligned} vol(t,s) = v \leftrightarrow \exists v_0 \exists t_0 \big(& vol_{init}(s) = v_0 \wedge start(s) = t_0 \wedge \\ & \big(inflow(s) \wedge \neg outflow(s) \wedge v = \min\{v_0 + R_{in} \cdot (t - t_0), C\} \vee \\ & inflow(s) \wedge outflow(s) \wedge v = \max\{\min\{v_0 + R_{in} \cdot (t - t_0) - R_{out} \cdot (t - t_0), C\}, 0\} \vee \\ & \neg inflow(s) \wedge outflow(s) \wedge v = \max\{v_0 - R_{out} \cdot (t - t_0), 0\} \vee \\ & \neg inflow(s) \wedge \neg outflow(s) \wedge v = v_0\big)\big). \end{aligned}$$

The expression $\neg inflow(s) \wedge \neg outflow(s)$ on the last line of the SEA is logically equivalent to the negated disjunction of the three contexts on preceding lines. Notice that in this axiom there are four different pairwise exclusive contexts, and for each context there is its own function describing how the fluent evolves with time.

PDDL 2.1 (2003) and PDDL+ (2006)

PDDL= the Planning Domain Definition Language: to standardize input

Maria Fox and Derek Long, "PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains" 2003, JAIR, v.20, p.61–124

PDDL+: Maria Fox and Derek Long, "Modelling Mixed Discrete Continuous Domains for Planning", JAIR, 2006, vol 27, p.235–297. (Semantics: hybrid automata; fluents and actions are instantiated.)

- PDDL 2.1 and PDDL+ are languages developed to describe temporal planning problems in a domain agnostic way.
- Consists of the following constructs:
  - Objects e.g. (gen1 - solarGenerator)
  - (Agent) Actions e.g. enableGenerator, disableGenerator
  - Durative Actions e.g. rampUp, rampDown, generatePower
  - Predicates e.g. IsGenerating, IsEnabled
  - Functions e.g. PowerGenerated, RunningCost
  - (Natural) Events e.g. powerFailure, overheat
  - Processes e.g. running

Vitaliy Batusov and Mikhail Soutchanski, "A Logical Semantics for PDDL+", ICAPS, 2019, pages 40-48. (It is based on the HTSC.)

PDDL+ Planners

Planner	Discretize	Heuristic	Heur. Type	Non-Linear?
CASP(2016)	Yes	No	-	Yes
COLIN(2012)	No	Yes	Independent	No*
DiNo(2016)	Yes	Yes	Independent	Yes
ENHSP(2017)	Yes	Yes	Independent	Yes
OPTIC++(2019)	No	Yes	Independent	Yes**
SMTPlan+(2016)	No	No	-	Yes***
UPMurphi(2010)	Yes	No	-	Yes
TM-LPSAT(2005)	No	No	-	No

* COLIN can only solve PDDL 2.1 problems (no contin. processes)
** OPTIC++ can only handle linear equations in preconditions
*** SMTPlan+ can only handle polynomial change

Daniel Bryce, Sicun Gao et al, “SMT-Based Nonlinear PDDL+ Planning", 29th AAAI, 2015, p.3247-53. Reduce PDDL+ planning to reachability problems of hybrid automata, which are encoded and solved as FOL formulas over the reals. Use the δ-complete decision procedure over the reals (dReal solver).

M.Balduccini et al “C*ASP* solutions for planning in hybrid domains", Theory and Practice of Logic Programming, 2017, Vol 17, N4, p.591-633 *(ASP)*

Compare with the **best** state-of-the-art planners: DiNO, ENHSP, SMTPlan+.

Our NEAT (Non-linEAr Temporal) Planner

All PDDL+ planners are based on grounding. They instantiate all actions schemas with constants from the planning problem instance. Most build a grounded transition system *before* search starts.

Our NEAT (Non-linEAr Temporal) Planner is a *lifted* forward search planner. It works directly with action schemas, i.e., no grounding in advance.

No discretization of time. The moments of physical time when actions are executed remain symbolic until the planner computes a schedule for the actions in a plan. Sometimes, non-linear change can be handled too. Calls an external numerical solver for non-linear programming problems.

The NEAT planner does lifted greedy best-first search (GBFS). Heuristic search is guided by a *domain-independent heuristic* (from optimal control).

The closely related zero-crossing (the continuous Skolem) problem is decidable for a function defined by a linear differential equation, if the order of the equation is up to 8 (OK for some realistic cases), conditional on Schanuel’s Conjecture [V.Chonev, J.Ouaknine, J.Worrell, J. of ACM, 2023]

Input: automatically translated from PDDL+ benchmarks, and then manually edited to produce a temporal BAT in the HTSC (can be fully automated).

Output: a time-stamped sequence of agent acts; may include natural actions.

PDDL+ benchmarks are available: `https://github.com/KCL-Planning/DiNo`
Descriptions: `http://kcl-planning.github.io/DiNo/benchmarks`
`http://kcl-planning.github.io/SMTPlan/benchmarks`

Experimental Evaluation

We evaluated our planner, *NEAT* and the existing state-of-the-art non-linear temporal numeric planners for which source code is readily available: *DiNo* , *ENHSP* (2020 version) and *SMTPlan+*.

Both *DiNo* and *ENHSP* are model-based planners that discretize time and reduce the temporal numeric planning problem to a numeric planning problem (without time). Both use different heuristics.

SMTPlan+ reduces the temporal numeric planning problem to a Satisfiability Modulo Theories (SMT) problem (Z3 solver), and it does not discretize time.

There are a few other PDDL+ planners such as *dReach*, *CASP*, *UPMurphi*, *OPTIC+*, *ScottyActivity*, but either they were previously discussed and compared, or their source code was not available.

Each planner is given 30min per instance and 1 GB of memory.

Metrics for comparison:
(1) execution time (in seconds): how long the planner takes to find a plan,
(2) coverage: how many instances were successfully solved
(3) plan duration (in time units) which is a measure of plan optimality

Methodology: Planning in Logic + External NLP Solver

- Heuristic search over the situation tree (state space is implicit)
- Numerical constraints are **not** handled by our planner directly. Collect all the numerical constraints encountered in a data structure. Pass them + Objective to an external non-linear programming (NLP) solver.
- Use state-of-the-art NLP solvers *KNITRO (Artelys)* and *IPOPT*.
- Our domain-independent heuristic finds the most promising action by relaxing and approximating the continuous processes. To evaluate an action, the heuristic has two stages.
- (a) Let action manifest its effects ("beneficial" or "harmful") through the processes initiated/terminated by the action.
  (b) Imagine a convex combination of all processes runs until numerical goal conditions are satisfied. Determine how much time it takes. This is a relaxation of what can happen in reality.

Limitations of our current implementation:

- translation from PDDL+ to HTSC is manual
- $\#t$ is not implemented: we encode a solution to ODE manually in the state evolution axioms.
- *overall* global constraints are verified after a plan has been computed.

Constraints Manager (by Nikola Kadovic)

ConstraintsManager (abbreviated CM) is a library providing a software interface between a planner and an external non-linear numerical solver. It allows the user to encode and solve non-linear programming (NLP) problems:

$\min f(\bar{x})$ over $\bar{x} \in R^n$ such that

$$\bar{g_L} \leq g(\bar{x}) \leq \bar{g_U}$$
$$\bar{x_L} \leq \bar{x} \leq \bar{x_U},$$

$f(\bar{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ is an objective function, $g(\bar{x}) : \mathbb{R}^n \mapsto \mathbb{R}^m$ is constrained to be between the vectors $\bar{g_L}$ and $\bar{g_U}$, and $\bar{x}$ must be between the vectors $\bar{x_L}$, $\bar{x_U}$.

Constraints make up $\bar{g}$. When we "add" constraints, we build upon a previous list of generated constraints, i.e., increasing the vector output of g. The set of constraints is feasible, if there is at least one point $\bar{x} \in \mathbb{R}^n$ that satisfies all constraints.

CM communicates with external solvers through an intermediate software package called *AMPL* ("A Mathematical Programming Language"). CM produces a string that is passed to AMPL that does pre-processing and then sends its output to a specified NLP solver.

All included results are preliminary: from a November 2024 version of the planner. They are subject to change.

Car domain

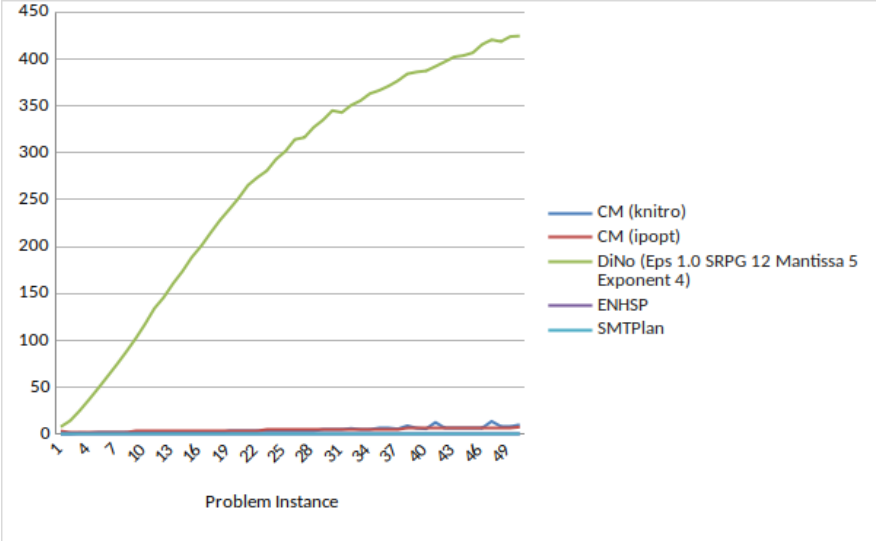
Drive a car from a standstill for a certain specified distance and arrive with 0 velocity as quickly as possible within the constraints.

- **Actions:** *accelerate*(*time*), *decelerate*(*time*), *stop*(*time*)
- **Natural Actions:** *movingBegin*(*time*), *movingEnd*(*time*), *engineExplode*(*t*), *windResistBegin*(*t*), *windResistEnd*(*t*)
- **Numerical Fluent:** *acceleration*(*x*, *s*) subject to an upper and lower bounds: its value *x* determines a context.
- **Fluents:** *engineBlown*(*s*), *stopped*(*s*), *running*(*s*)
- **Temporal Fluents:** *velocity*(*time*, *s*), *distance*(*time*, *s*)

In all planning instances, the goal is to travel as soon as possible at least 30 units distance and stop with 0 velocity within 50 units of time allotted to the car. Wind resistance happens at velocity 50. Engine explodes at velocity 100.

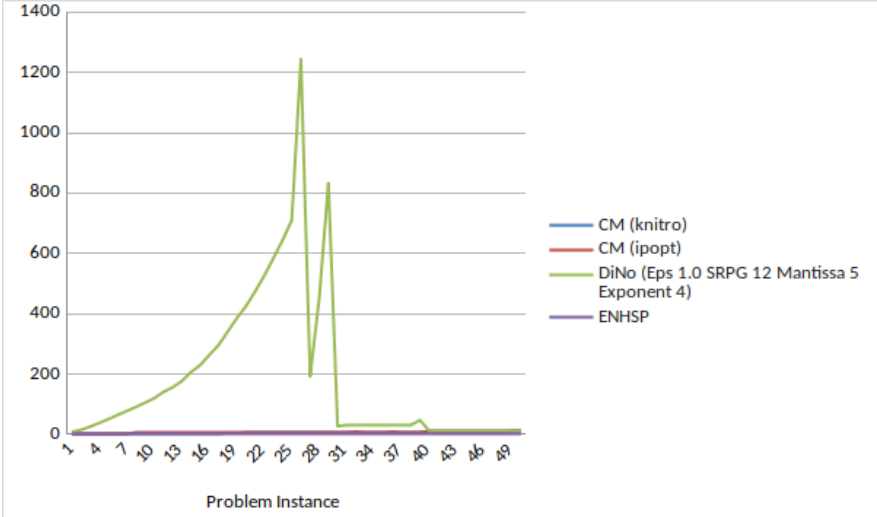
In the different planning instances, there are different upper and down limits on acceleration, and deceleration. For example, in the instance 7, the limits are +7 and -7, accordingly, but in the instance 25 the limits are +25 and -25. Everything else is the same.

Linear Car (no wind): Execution Time



All planners solved all 50 instances: SMTPlan takes less than 0.07sec, ENHSP takes around 0.3sec on all instances. NEAT time varies from 1 to 10sec (CM with Knitro), and from about 3 to 7sec for CM with IPopt. The computed plans are non-optimal: 2 accelerate actions followed by 2 decelerate actions.

Nonlinear Car (with Wind): Execution Time



SMTPlan could not solve any instances, since it works only with polynomial functions of time; in this problem velocity changes $\log()$. Other planners solved all 50 instances. ENHSP takes about 0.3sec. NEAT: CM with Knitro time varies from about 1 to 10sec, CM with IPopt time varies from 3sec to about 10sec. Execution time of DiNo varies wildly as shown.

Solar Rover domain

A rover needs to charge its internal batteries in order to transmit some data

- **Actions:** *switchGenBatteryOn*(*genBattery*, *t*), *useBatteryBegin*(*b*, *t*), *useBatteryEnd*(*b*, *t*), *sendData*(*t*) – needs 500 units of energy
- **Natural Actions:** *sunshine*(*t*) - provides 400 units
- **Static Fact:** *sunexposure*(*sunriseTime*), *powerlimit*(1000)
- **Temporal Fluents:** *roverEnergy*(*s*)
- **Fluents:** *on*(*battery*, *s*), *off*(*battery*, *s*), *gbon*(*genBattery*, *s*), *gboff*(*genBattery*, *s*), *roverSafe*(*s*), *dataSent*(*s*), *night*(*s*), *usingBattery*(*b*, *s*)
- **Temporal Fluents:** *SoC*(*battery*)

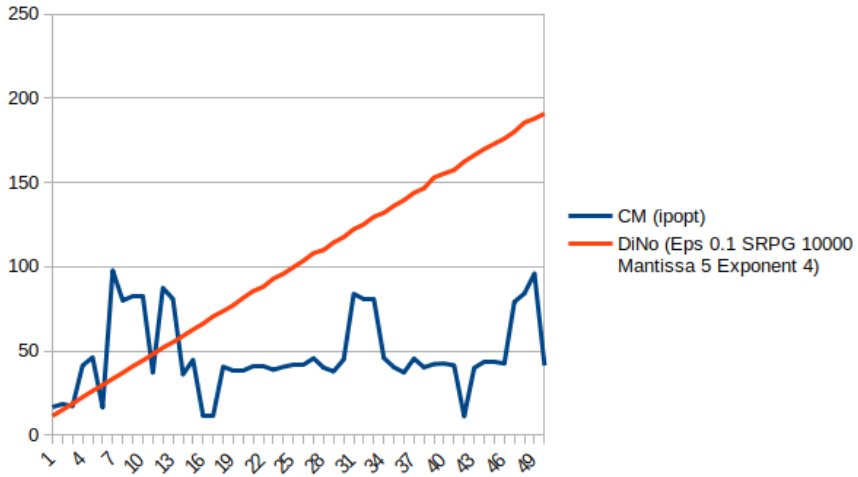
SoC= “state of charge" (for a battery). Initially, rover has 0 energy, there are 3 batteries with SoC 40, 80, 100, and a general battery with SoC=100. Time when *sunshine* event happens varies across the instances, e.g., in the 1st it is at 50, in the 40th instance it is at 2000, etc. Goal: get enough power to send data as soon as possible.

Non-linear Solar Rover

Identical to the linear version, but now there is a charging process that can begin or end. Its purpose is to increase the value of the *roverEnergy* temporal fluent by some non-linear function. Below, we list only the new additions to the domain.

- **Actions:** *chargingBegin*(*Time*), *chargingEnd*(*Time*)
- **Process:** *charging*
- **Temporal Fluents:** *roverEnergy*(*time*, *s*)

Non-Linear Solar Rover: Execution Time



ENHSP and SMTPlan could not compute any plans.
DiNo: time increased linearly from about 11sec to 190sec.
NEAT: CM with IPopt time varied a lot from 16sec to about 97sec. CM with Knitro did not work. AMPLEX with Knitro: time varies between 4 and 11 sec. AMPLEX is described in S.Mathew, M.Soutchanski “Heuristic Planning for Hybrid Dynamical Systems with Constraint Logic Programming", Italian Wsh on Planning, IPS-2023, `https://ceur-ws.org/Vol-3585/`

1D Powered Descent

How to land softly on the surface of a planet? The spacecraft falls down and gains velocity due to the force of gravity. It can begin thrusting process by firing its engines against gravity to decrease velocity. The duration of thrust process is flexible.The change of distance due to thrust is calculated as

$$-Isp \cdot G \cdot (t-t_0) - Isp \cdot G \cdot (1/q) \cdot (m(t_0) - q \cdot (t-t_0)) \cdot \log\left((m(t_0) - q \cdot (t-t_0))/m(t_0)\right)$$

where *Isp* is the specific impulse of the thruster, q is a constant, G is the acceleration due to gravity, $m(t_0)$ is the initial mass of the spacecraft before firing thrusters, t_0 - time when thrust (fall) begins, t - current time.

- **Actions:** *fallBegin*(t), *fallEnd*(t) *thrustBegin*(t), *thrustEnd*(t), *land*(t).
- **Natural Action**: *crash*(t)
- **Temporal Fluents:** mass, velocity, distance
- **Fluents:** *crashed*(s), *inProgress*(s), *landed*(s)

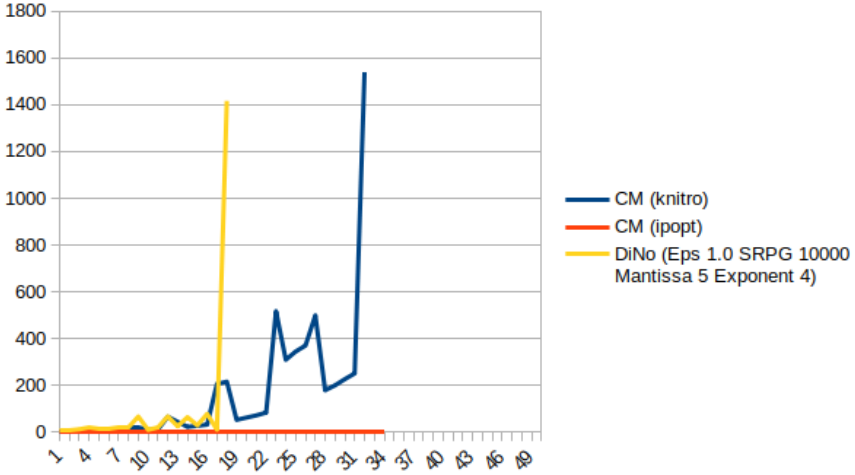
land is possible if the final velocity and distance from the surface are within the safe bounds, otherwise *crash* happens.

For a falling body, distance changes according to Newton’s equation $d(t_0) + v(t_0) \cdot (t-t_0) + 0.5 \cdot G \cdot (t-t_0)^2$.

Velocity of a falling body with active thrust changes as $v(t_0) + G \cdot (t-t_0) - Isp \cdot G \cdot \log\left((m(t_0) - q \cdot (t-t_0))/m(t_0)\right)$

In the instances, only the values of final distance vary from 100 to 2000.

1D-Powered Descent: Execution Time



ENHSP and SMTPlan could not compute any plans.
DiNo solved only 19 instances: time varies between 12 and 99sec, but Instance 19 takes about 1770 sec.
NEAT: CM with Knitro solved 33 instances: from 0.3sec to 250sec, the last instance 32 takes 1500 sec. CM with Ipopt solved 35: time varies between 0.9 and 2.2 sec. AMPLEX with Knitro solved 30 instances (could not solve instances 16-18,32): time was around 0.4s

A Few Topics for Future Research: Part 1

In the HTSC, the mutually exclusive contexts include only truth-valued (parameterized) fluents. But it would be interesting to consider an extension of the HTSC where the contexts may include numerical fluents as well. This would require significant revision of the State Evolution Axioms.

Integration of Task and Motion Planning (TAMP) is a large and practically important research area with focus on probabilistic algorithms. However, there is a lack of conceptual understanding of TAMP. The question is how the TAMP problems can be formulated within the HTSC in a general form. The following papers are good starting points.

*E*rion Plaku, Gregory D. Hager: “Sampling-Based Motion and Symbolic Action Planning with Geometric and Differential Constraints". ICRA 2010: pages 5002-5008.
Marc Toussaint: “Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning". IJCAI 2015: p.1930-1936.

Conclusion and Future Work

Our NEAT planner demonstrates performance that is comparable with the state-of-the-art planners DiNo, SMTPlan+, ENHSP across several realistic benchmarks for hybrid systems.

Future Work.

- Consider a broader class of hybrid systems where actions can change logical fluents only, but have no effect on processes.
- Optimize iteratively constructed NLP: consecutive NLP problems are closely related
- More informative heuristic for domain-independent planning in hybrid systems
- Prove that under certain conditions our planner is sound.
- Find when NEAT planner can work correctly with natural actions
- Experimental evaluation on the realistic domains where the objects can be created/destroyed at run time.
- Explore if NEAT can be useful to solve practical optimization problems for hybrid systems.

A few Topics for Future Research: Part 2

The most popular approach to solving planning and control problems in the hybrid systems is Mixed Integer Non-Linear Programming (MI-NLP). There is a large library MINLPlib that accumulates the realistic benchmarks solved using traditional techniques from Operation Research. See details at

`http://minlplib.org/applications.html`
`https://www.minlp.org/index.php`

It would be interesting to demonstrate that some of those planning benchmarks can be formulated in the HTSC. This research may encounter new features and extensions that have to be added to HTSC. Caution: not all benchmarks in MINLPlib are planning related, and among those that are related, not all of them model a hybrid system.

The important research question is whether the flexibility and generality of modelling mixed discrete-continuous domains in HTSC can also provide the benefits in terms of solving the related optimization problems more efficiently.

References

Vitaliy Batusov and Mikhail Soutchanski: “A Logical Semantics for PDDL+", ICAPS, 2019, pages 40-48.

A.Ben-Tal and A. Nemirovski: “Optimization III: Convex Analysis, NLP Theory and Algorithms", www2.isye.gatech.edu/~nemirovs/OPTIIILN2024Spring.pdf

Michael Cashmore, Daniele Magazzeni, Parisa Zehtabi: “Planning for Hybrid Systems via Satisfiability Modulo Theories". JAIR, 2020, v.67: 235-283 (SMTPlan+)

Ventsislav Chonev, Joel Ouaknine, and James Worrell: “On the Zeros of Exponential Polynomials", J. ACM, vol 70, N4, 2023. https://doi.org/10.1145/3603543

E. Fernández-González, Brian C. Williams, Erez Karpas: “ScottyActivity: Mixed Discrete-Continuous Planning with Convex Optimization". JAIR, v.62: 579-664 (2018)

Maria Fox and Derek Long, “PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains" 2003, JAIR, v.20, p.61–124

Maria Fox and Derek Long: “Modelling Mixed Discrete Continuous Domains for Planning", JAIR, 2006, vol 27, p.235–297. (Introduction to PDDL+)

Shaun Mathew and M.Soutchanski: “Heuristic Planning for Hybrid Dynamical Systems with Constraint Logic Programming", https://ceur-ws.org/Vol-3585/

Yurii Nesterov: “Lectures on Convex Optimization", 2nd edition, Springer, 2018. (Chapter 1 provides intro to general optimization problems and NLP).

Wiktor Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, Fabio Mercorio: “Heuristic Planning for PDDL+ Domains". IJCAI 2016: 3213-3219 (DiNo)

Enrico Scala, P. Haslum, S. Thiébaux, M. Ramírez: “Subgoaling Techniques for Satisficing and Optimal Numeric Planning". JAIR, 2020, v68: 691-752. (ENHSP)

Appendix: Why theorem proving approach to plan?

Our deductive approach to planning is formulated in the Situation Calculus (SC) where the initial theory $\mathcal{D}_{S_0}$ is a *finite* set of fluent literals and a goal can be a $\exists$-quantifed conjunction of fluents.

This allows for representation of a broad class of planning problems.

- Can plan without the Domain Closure Assumption (DCA) that restricts to finitely many $C_1, \ldots, C_n$ s.t. $\forall x(x = C_1 \vee \cdots \vee x = C_n)$. Objects can be initially unknown, or created/destroyed while planning. Need an infinite model over constants naming objects.
- Can plan without the Closed World Assumption (CWA), i.e., knowledge can be incomplete, e.g. numeric conformant plan.
- Can plan when actions have parameters that vary over infinite domains, since search for a plan is done over the situation tree: situations serve as concise symbolic proxies for infinite models.
- Can do lifted planning using action schemas: instantiate actions at search time without building explicit state space in advance.

In our current implementation, we make simplifying assumptions to guarantee that our planner is efficient.