Lecture 5: NEAT Planner Using Hybrid Temporal Situation Calculus

Mikhail Soutchanski
(Acknowledgement: joint work with Nikola Kadovic and Shaun Mathew)

July 10, 2025

Prerequisites: basics of Non-Linear Porgramming (NLP) at the level of an undergraduate course on multivariable calculus.

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+.

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.)

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

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. Any nonlinear change can be handled, in principle. Calls an external numerical solver for non-linear programming.

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

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 at https://github.com/KCL-Planning/DiNo
Descriptions: http://kcl-planning.github.io/DiNo/benchmarks
http://kcl-planning.github.io/SMTPlan/benchmarks

Constraints Manager (by Nikola Kadovic)

ConstraintsManager (abbreviated CM) is a library providing a software interface between a planner written in PROLOG and an external non-linear numerical constraint solver. CM is written in C++ and Prolog. It allows the user to encode and solve non-linear programming (NLP) problems of the following structure:

$$\min f(\bar{x}) \text{ over } \bar{x} \in R^n \text{ such that}$$
$$\bar{g_L} \leq g(\bar{x}) \leq \bar{g_U}$$
$$\bar{x_L} \leq \bar{x} \leq \bar{x_U},$$

$f(\bar{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ 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 $\bar{x_L}$, $\bar{x_U}$.

In CM, the objective function $f(\bar{x})$ will always be an expression, not a constraint. Expressions are intended to be symbolic (involving variables). They can be created from other expressions, and can be included in constraints. All un-instantiated variables inside expressions will be eventually assigned a final value once the NLP has been solved.

Constraints make up $\bar{g}$. When we "add" constraints, we are building 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.

Methodology: Planning in Logic + External NLP Solver

- Use a deductive regression-based planner that does heuristic planning over the situation tree (state space is implicit)
- Numerical constraints are **not** handled by our planner directly. Collect all the encountered numerical constraints in a data structure. Pass them + Objective to external Non-Linear Programming solver (NLP).
- 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 underlying continuous processes. To evaluate an action, 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 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 on durative actions/processes are verified after a plan has been computed.

CM communicates with AMPL

Once the problem has been built completely, it can be solved by using `'solve/3'`: *solve*(C, *min*(F), *MinOptFuncValue*), where C is a list of constraints, and F is an expression.

To evaluate 'solve', CM will invoke an external NLP solver with the problem. If an NLP is feasible, and a solution is found, then '*MinOptFuncValue*' will be unified with final value of F at the convergence point. If we are trying to minimize F, this will either be a local minimum or global minimum, depending on if the problem is convex or not.

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.

In our experimental results, we used the Euler method to approximate a temporal function, where the unit interval of time was divided into 8 sub-intervals.

All included results are preliminary: from a November 2024 version of the planner. They are subject to change. In particular, our heuristic was supplemented by a focal search based on minimization of the number of actions in a plan; in focal search, the meta-parameter was 0.3

Linear Generator domain

Run a generator for a specified amount of time refueling when necessary with auxiliary tanks.

- **Actions:** *generateBegin*(*gen*, *s*), *generateEnd*(*gen*, *s*), *refuelBegin*(*gen*, *tank*, *s*), *refuelEnd*(*gen*, *tank*, *s*)
- **Static Facts:** *capacity*(*gen*), *tank*(*tname*), *generator*(*gname*)
- **Processes:** *generate*(*gen*), *refuel*(*gen*, *tank*)
- **Fluents:** *generatorRan*(*gen*, *s*), *available*(*tank*, *s*)
- **Temporal Fluents:** *fuelLevel*(*gen*, *time*, *s*)

Instance with 2 refueling tanks: *initFuelLevel*(*gen*, 0, S_0) is 960, total capacity is 1000, there are tanks1 and tank2 with 20 units of fuel. While generating gen spends 1 fuel with rate 1, and refueling rate is 2

In the different planning instances, the number of refueling tanks vary from 1 to 50, the initial fuel level in the generator can vary across instances accordingly, but the flow rate is constant.

This is *not* a time minimization problem, since the task is to run a generator for a fixed (1000 units) time. The main challenge: symmetry between the refueling tanks.

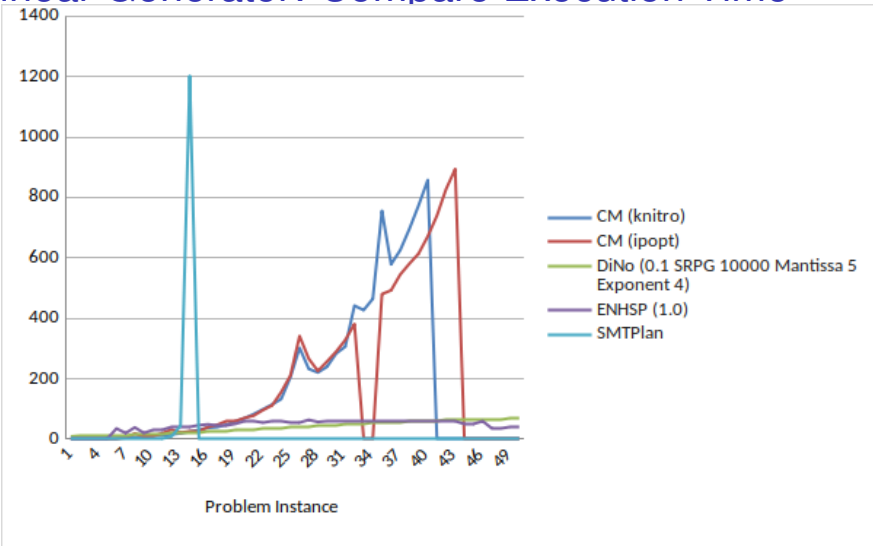
Linear Generator: OPTIC and OPTIC++

Linear Generator									
tanks	10	20	30	40	50	60	70	80	90
OPTIC	0.67	3.70	12.04	99.97	390.65	63.65	117.61	190.52	224.67
OPTIC++	1.87	10.19	30.80	152.00	420.27	63.77	119.11	192.85	235.50

Copy/Paste from E. Denenberg and A. Coles. “Mixed Discrete Continuous Non-Linear Planning through Piecewise Linear Approximation”. In: Proceedings of the 29th International Conference on Automated Planning and Scheduling, 2019, pp. 137–145. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3469 (cit. on p. 30).

The source code of OPTIC++ was not publicly available and for this reason we could not use it for comparison.

Linear Generator: Compare Execution Time



CM with Knitro solved only 40 instances within allocated time.
CM with IPopt solved 44 instances within allocated time.

SMTPlan solved only 15 instances.

DiNo and ENHSP solved all 50 instances fast thanks to their symmetry-breaking heuristics.

Car domain

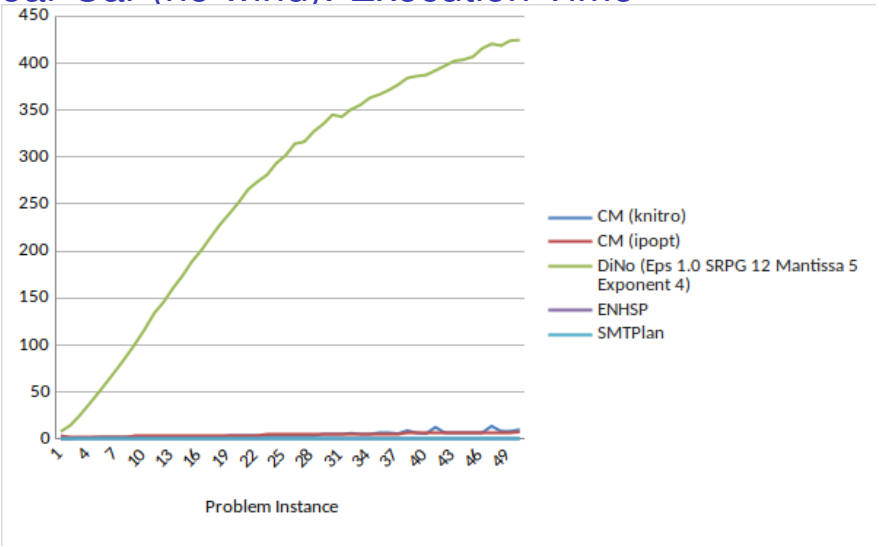
Drive a car from a standstill for a certain specified distance and arrive with 0 velocity as quickly as possible within the constraints.

- **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.

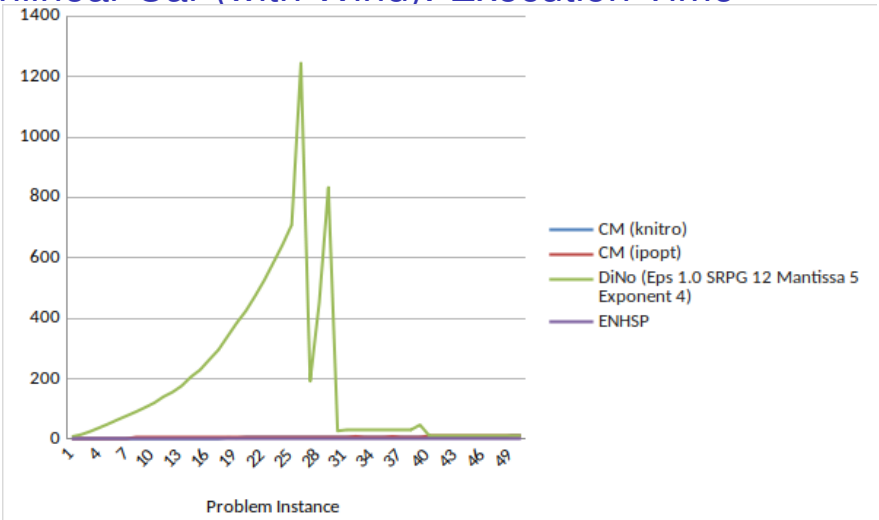
In the different planning instances, there are different upper and down limits on acceleration, and deceleration. For example, in the instance 7, the limits are +7 and -7, accordingly, but in the instance 25 the limits are +25 and -25. Everything else is the same.

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*)

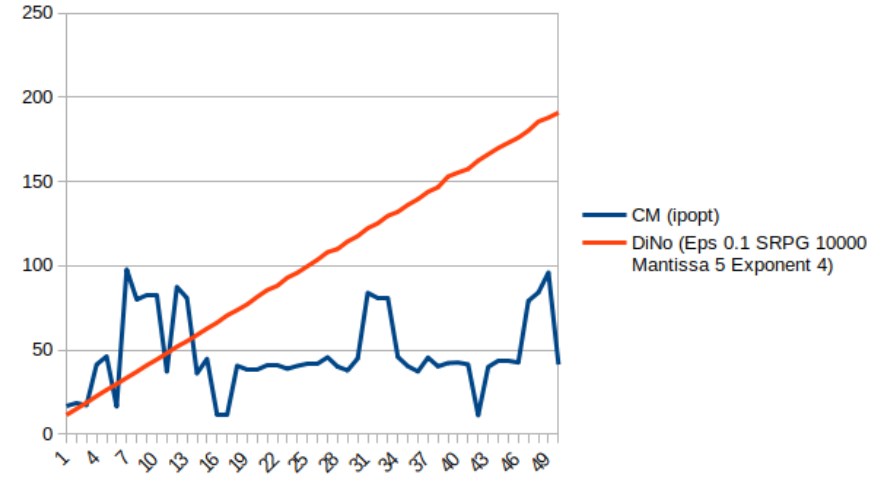
SoC= “state of charge" (for a battery). Initially, rover has 0 energy, there are 3 batteries with SoC 40, 80, 100, and a general battery with SoC=100. Time when *sunshine* event happens varies across the instances, e.g., in the 1st it is at 50, in the 40th instance it is at 2000, etc. Goal: get enough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

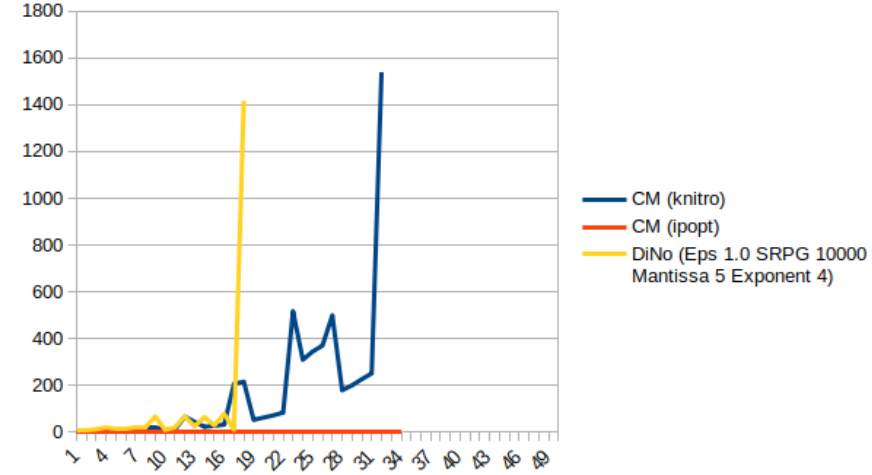
- **Actions:** *chargingBegin*(*Time*), *chargingEnd*(*Time*)
- **Process:** *charging*
- **Temporal Fluents:** *roverEnergy*(*time*, *s*)

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: 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

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.

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 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.

So far, reasoning in the HTSC is done using regression only. Is it possible to define progression for temporal BATs ? For atemporal fluents, this can be done similarly to progression in local effect BATs. For temporal fluents within situation this likely can be done using Picard's method for ODEs. The question is how to merge these methods to compute progression in HTSC.

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.

Erion 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.

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 huge 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

Marcello Balduccini, Daniele Magazzeni, Marco Maratea, Emily Leblanc: "CASP solutions for planning in hybrid domains". Theory Pract. Log. Program. 2017, 17(4): pages 591-633

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+)

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, Patrik Haslum, Daniele Magazzeni, Sylvie Thiébaux: "Landmarks for Numeric Planning Problems". IJCAI 2017: 4384-4390 (ENHSP)