28th International Conference on
Automated Planning and Scheduling

June 24-29, 2018, Delft, the Netherlands

PlanSOpt 2018

Proceedings of the 3rd Workshop on

Planning, Search and Optimization

Edited by:

Michael Cashmore, Andre A. Cire, Chiara Piacentini

Organization

Michael Cashmore
Department of Informatics, King's College London, UK

Andre A. Cire
Department of Management & Rotman School of Management, University of Toronto, Canada

Chiara Piacentini
Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Program Committee

Kyle E. C. Booth, University of Toronto
Margarita Castro, University of Toronto,
Michele Lombardi, DISI, University of Bologna
Andrea Micheli, Fondazione Bruno Kessler
Michael Morin, Laval University
Mark Roberts, Naval Research Laboratory
Domenico Salvagnin, University of Padova
Enrico Scala, Fondazione Bruno Kessler

Foreword

AI planning problems are traditionally formulated using state transition systems and solved using heuristic search. In optimization, problems consist of finding the values for variables that maximize an objective, while subject to logical and mathematical constraints. Many optimization algorithms, e.g. branch and bound for Mixed Integer Linear Programming, rely on search over the solution space. While similarities between AI planning and optimization are numerous, the two fields advanced almost independently and their interconnection remains largely unexplored. Renewed interest in the integration of optimization techniques into AI planning has emerged recently with the use of integer and linear programming to automatically derive heuristics, but several other auxiliary methods can be exploited to speed up search from one area to the other.

The aim of this workshop is to foster communication and collaboration between researchers in the fields of AI planning/scheduling, search, and optimization.

Michael Cashmore, Andre A. Cire, Chiara Piacentini
June 2018

Contents

Using Squeaky Wheel Optimization to Derive Problem Specific Control Information for a One Shot Scheduler for a Planetary Rover
Wayne Chi, Jagriti Agrawal and Steve Chien **1**

Off-line/on-line Optimization under Uncertainty
Allegra De Filippo, Michele Lombardi and Michela Milano **10**

Metric Nonlinear Hybrid Planning with Constraint Generation
Buser Say and Scott Sanner **19**

Using Squeaky Wheel Optimization to Derive Problem Specific Control Information for a One Shot Scheduler for a Planetary Rover

Wayne Chi, Steve Chien, Jagriti Agrawal
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

We describe the application of using Monte Carlo simulation to optimize a schedule for execution and rescheduling robustness and activity score in the face of execution uncertainties. We apply these techniques to the problem of optimizing a schedule for a planetary rover with very limited onboard computation. We search in the schedule activity priority space - where the onboard scheduler is (a) a one shot non-backtracking scheduler in which (b) the activity priority determines the order in which activities are considered for placement in the schedule and (c) once an activity is placed it is never moved or deleted. We show that simulation driven search outperforms a number of alternative proposed heuristic static priority assignment schemes. Our approach can be viewed using simulation feedback to determine problem specific heuristics much like squeaky wheel optimization.

Introduction

Embedded schedulers must often perform within very limited computational resources. We describe an approach to automatically deriving problem specific control knowledge for a one-shot (non-backtracking) scheduler intended for a planetary rover with very limited computing. In this application, the onboard scheduler is intended to make the rover more robust to run-time variations (e.g., execution durations) by rescheduling. Because the general structure of the schedule is known a priori on the ground before uplink, we use both analysis of the schedule dependencies and simulation feedback to derive problem specific control knowledge to improve the onboard scheduler performance.

The target onboard scheduler is a one-shot limited search scheduler. Because the scheduler does not backtrack across activity placements, the order in which it considers activities heavily influences generated schedule quality. In our approach, we search the space of activity priorities which determine the order in which the scheduler considers activity placement. At each step in the priority search, a Monte Carlo simulation is conducted to assess the likelihood of an activity being executed. Using an approach analogous to squeaky wheel optimization, these runs are automatically analyzed and used to feed back into adjustments to the activity priorities (and hence the order in which they are considered for inclusion in the schedule for both initial schedule generation and rescheduling). This search in the activity priority space continues until all requested activities are included or a resource bound is exceeded. We call this method *Priority Search* and we present empirical results that show that *Priority Search* outperforms several static priority assignment methods (those that do not use Monte Carlo feedback) *including manual expert derived priority setting*.

We study this problem in the context of setting activity priorities as part of the ground operations process for a one-shot, non-backtracking scheduler (Rabideau and Benowitz 2017) designed to run onboard NASA's next planetary rover, the Mars 2020 (M2020) rover (Jet Propulsion Laboratory 2017a). For our problem, the onboard scheduler is treated as a predetermined "black box".

The remainder of the paper is organized as follows. First we describe our formulation of the scheduling problem, metrics for schedule goodness, and the onboard scheduling algorithm. Second, we describe several static approaches to priority assignment as well as our *priority search* approach that leverages Monte Carlo simulation feedback. Third, we describe empirical results demonstrating the efficacy of *priority search* over static methods, evaluating on *sol types*, the best available anticipated operations plans for the M2020 planetary rover mission. Finally, we describe related and future work and conclusions.

Problem Definition

For our defined scheduling problem (Rabideau and Benowitz 2017), the scheduler is given

- a list of activities
  $A_i\langle p, R, e, dv, \Gamma, T, D\rangle \ldots A_n\langle p, R, e, dv, \Gamma, T, D\rangle$
- where p is the scheduling priority of the activity, and
- R is the set of unit resources $R_1 \ldots R_m$ that the activity will use (up to project limitations - 128 for M2020), and
- e and dv are the rate at which the consumable resources energy and data volume respectively are consumed by the activity, and
- Γ are non-depletable resources used such as sequence engines available or peak power, and

- T is a set of the activity's optional a) start time windows $T_{i\_start} \dots T_{i\_end}$ and b) preferred schedule time $T_{i\_preferred}$, and
- D is a set of the activity's dependency constraints from $A_j \rightarrow A_k$ [1]

All activities are *Mandatory Activities*. These are activities, $m_1 \dots m_j \subseteq A$, that must be scheduled as long as the given set of inputs are *valid*. In order for a set of inputs to be considered *valid*, there must exist a valid (e.g. constraint satisfying) schedule - in the context of the scheduler - that includes all of the mandatory activities. Note that the M2020 Onboard Scheduler is an incomplete algorithm. As a result, there could be a set of inputs where valid schedule exists and a complete scheduler would place all mandatory activities, but the Onboard scheduler would not. Since not all input sets will be *valid*, it is important for us to modify the input sets (e.g. changing priorities) to allow all mandatory activities to be scheduled.

In addition, activities can be grouped into *Switch Groups*. A *Switch Group* is a set of activities where exactly one of the activities in the set must be scheduled. The activities within a switch group are called *switch cases* and vary only by how many resources (time, energy, and data volume) they consume. Switch groups allow us to schedule a more resource-consuming activity if it will fit in the schedule. For example, one of the M2020 instruments takes images to fill mosaics which can vary in size; for instance we might consider $1x5$, $3x5$, or $5x5$ mosaics. Taking larger mosaics might be preferable, but taking a larger mosaic takes more time, takes more energy, and produces more data volume. These alternatives would be modeled by a switch group that might be as follows:

$$SwitchGroup = \begin{cases} Mosaic_{1x5} & \text{Duration=100 sec} \\ Mosaic_{3x5} & \text{Duration=200 sec} \\ Mosaic_{5x5} & \text{Duration=400 sec} \end{cases} \quad (1)$$

In the above example, the scheduling priority order would be $Mosaic_{1x5}$ the lowest of the three, then $Mosaic_{3x5}$, and $Mosaic_{5x5}$ the highest. The desire is for the scheduler to schedule the activity $Mosaic_{5x5}$ but if it does not fit then try scheduling $Mosaic_{3x5}$, and eventually try $Mosaic_{1x5}$ if the other two fail to schedule. The challenge for the scheduler is that getting a preferred switch case is not deemed worth forcing out another mandatory activity from the schedule. Because the normal approach to handling such interactions is to search, this introduces complications into the scheduling algorithms but these are the subject of a different paper.

The charter of the scheduler is to produce a grounded time schedule that satisfies all of the above constraints.

We also make the following assumptions:

1. There exists a set of activity scheduling priorities that would allow all mandatory activities to be scheduled by the scheduler [2].
2. The prior schedule is executed while the scheduler is running (Chi et al. 2018).
3. Activities do not fail.
4. No preemption (activities are only preempted as a major failure case for M2020).
5. The onboard scheduler is a "black box" - the onboard scheduler algorithm (Algorithm 1) is fixed.

The goal of the scheduler is to schedule all mandatory activities and better switch cases [3] while respecting individual and plan-wide constraints.

The goal of the priority setting algorithm is to derive a set of priorities that will best allow the scheduler to achieve that goal. Not only that, but we must derive that set of priorities in the shortest amount of time possible in order to satisfy daily mission time constraints.

Scheduler Design

Algorithm 1 Onboard Scheduler

Input:
$A\langle p, R, e, dv, \Gamma, T, D\rangle$: List of activities with their individual constraints
C: Constraints for the whole plan (e.g. available cumulative resources)
S: Current state of the spacecraft (state of charge, data volume, activity status)
Output:
U: Resulting schedule

Sort(A) ▷ Sorted by highest to lowest priority.
for each $a \in A$ **do**
  $P \leftarrow \emptyset$ ▷ Some activities may require automatically generated preheats
  $M \leftarrow \emptyset$ ▷ Some activities may require automatically generated maintenances
  $I \leftarrow [a.earliest\_start\_time, a.latest\_start\_time] \bigcap find\_valid\_intervals(a.unit\_resources) \bigcap find\_valid\_intervals(a.activity\_status) \bigcap find\_valid\_intervals(a.data\_volume)$
  if $requires\_preheat(a)$ **then**
    $P \leftarrow generate\_preheat\_activities(a)$
    $M \leftarrow generate\_maintenance\_activities(a)$
  end if
  $I \leftarrow I \bigcap find\_valid\_intervals(a.energy, P, M) \bigcap find\_valid\_intervals(a.peak\_power, P, M)$
  $awake \leftarrow generate\_awake\_activity(a, I)$
  if $I \neq \emptyset$ **then**
    $schedule\_activity(a, I)$
    $schedule\_activity(awake, I)$
    for each $p \in P$ **do**
      $schedule\_activity(p, I)$
    end for
    for each $m \in M$ **do**
      $schedule\_activity(m, I)$
    end for
  end if
end for

The Mars 2020 onboard scheduler (Algorithm 1) is a single shot, non-backtracking scheduler that schedules (consid-

[1] $A_j \rightarrow A_k$ means the scheduled end time of A_k must be before the scheduled start time of A_j.

[2] Since our algorithm includes an incomplete scheduler, our assumption of a valid set of inputs can only hold true for our particular scheduler

[3] See Evaluating a Schedule for more information

ers activities) priority first order and never removes or moves an activity after it is placed during a single scheduler run. It does not search except when considering valid intervals for a single activity placement and when scheduling sleep and preheats [4] (Rabideau and Benowitz 2017).

Due to the greedy, non-backtracking nature of the onboard scheduler, the order in which activities are scheduled can greatly impact the quality of the schedule.

Evaluating a Schedule

In order to evaluate the goodness of a particular priority assignment, we have developed a scoring method based on how many and what type of mandatory and switch group activities are able to be scheduled successfully by the scheduler. The score is such that the value of any single mandatory activity being scheduled is much greater than that of any combination of switch cases (at most one activity from each switch group can be scheduled). This ensures the following strict ordering:

$$V(m \in M) \gg \sum_{i=1}^{n_S} V(s \in S_i) \tag{2}$$

where $V(x)$ is the value of activity x being scheduled, M is the set of all mandatory activities, n_S is the number of switch groups, S_i is switch group i, and s is a switch case in switch group S_i.

Static Algorithms for Activity Priority Assignment

We have developed several static algorithms which set the priorities of activities based on various activity ordering criteria. These algorithms do not consider Monte Carlo simulations of plan execution where activities may end early or late while determining priorities, unlike our Priority Search approach. We will later compare these to our Priority Search approach to gain a better understanding of how well it performs. Activities which must begin at a particular time (e.g. data downlink) are always given the highest priority and thus are not affected by the static algorithms described.

The following four methods are used to initialize activity priorities:

- *Equal Priorities.* All activities have equal priorities.
- *Random Assignment.* Each activity is given a random priority.
- *Latest Start Time.* The activity priorities are ordered by the latest time they are allowed to start. The activity with the earliest such time has the highest priority.
- *Human Expert.* Each activity is assigned a priority based on the start time of the activity in a schedule constructed by a human expert. The activity with the earliest start time in this schedule has the highest priority.

The following two methods are applied to the priorities after they have been initialized in one of the four ways described above:

[4] Sleep and preheats are activities automatically generated and scheduled by the scheduler.

- *Dependencies.* $A \rightarrow B$ means that B must execute successfully before A can start. To generate a schedule that respects this,

$$A \rightarrow B \Rightarrow priority_A < priority_B \tag{3}$$

where higher priority means an activity is considered for scheduling earlier.

- *Tie Breaker.* If activities have the same priority assignment the activity with earliest latest allowed start time is of higher priority. If they also have the same latest allowed start time then the longer activity has the higher priority. If all of these attributes are equal then the higher priority activity is chosen lexicographically based on each activity's unique identifier.

Priority Search

In order to determine a set of priorities which will allow the scheduler to generate a schedule better than our static heuristics, we attempt to search the priority space in an approach similar to Squeaky Wheel Optimization (SWO) as described in Joslin and Clements 1999 (Joslin and Clements 1999). Squeaky Wheel Optimization usually involves a constructor, an analyzer, and a prioritizer. The constructor generates a schedule, the analyzer determines problem areas and assigns "blame" to certain elements in the schedule, and the prioritizer modifies the order in which the elements are considered. This process repeats until a satisfactory result is reached or allotted time runs out. However, our scheduling problem is intrinsically tied to execution and analyzing the initial schedule generated by itself is not satisfactory. Our approach (Figure 1) builds upon the usual SWO approach by incorporating a simulation of execution and Monte Carlo to build an execution sensitive result. We call our approach Priority Search as it searches the priority space using Monte Carlo simulation feedback to find a good set of priorities, unlike the static algorithms.

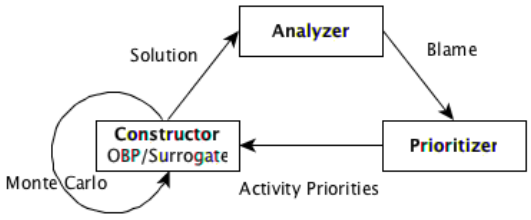


Figure 1: Squeaky Wheel accounting for Execution

Constructor

Typically, the constructor generates a schedule as the solution, which is then fed into the analyzer. However, our scheduling problem must be taken in context with execution. Activities may finish early or late which affect how many and which activities can be scheduled. In order to take this into account, we generate the final schedule of a

(lightweight) simulation of the entire plan execution. This is simulated by letting activities finish early or late by a variable amount based on a probabilistic model of plan execution [5]. However, the probabilistic model may promote misleading results if only sampled once. As a result, our constructor (Algorithm 2) runs a Monte Carlo and simulates multiple plan executions, passing on all of the executed plans as the solution to the analyzer.

Algorithm 2 Monte Carlo Constructor

Input:
$A\langle p, R, e, dv, \Gamma, T, D\rangle$: List of activities with their individual constraints
C: Constraints for the whole plan (e.g. available cumulative resources)
N: Number of runs in the Monte Carlo
Output:
S: List of all final schedules after simulating execution
1: $i \leftarrow 0$
2: **while** $i < N$ **do**
3: $schedule \leftarrow$ simulation(A, C)[6]
4: $S_i \leftarrow schedule$
5: $i \leftarrow i + 1$
6: **end while**

Priority Analyzer

The analyzer (Algorithm 3) takes the solution and assigns blame to problem areas. Since our objective is to schedule all mandatory activities and better switch cases, we blame all activities that are not scheduled. Since the solution is multiple schedules, there may be some Monte Carlo runs where the activities do not succeed or fail to be scheduled. For simplicity, we chose to blame any activity that was unscheduled in any of the schedules, but other approaches may assign blame according to how many times an activity was not scheduled.

Algorithm 3 Monte Carlo Analyzer

Input:
$A\langle p\rangle$: List of activities with priorities
S: List of all final schedules after simulating execution
Output:
U: List of all unscheduled activities
$score$: Score (objective function)
1: **for each** $S_i \in S$ **do**
2: $U \leftarrow U \bigcup \{\forall a \in A | a \notin S_i\}$
3: $score \leftarrow score +$ get_score(S_i)
4: **end for**

Constant Step Prioritizer

A simple way to re-prioritize is to increase the blamed (unscheduled) activities' priorities by a constant step size s.

Typically, activities have varying degrees of flexibility due to their constraints (resources, dependencies, time, etc.).

[5] See Empirical Results for how that probabilistic model was generated.

[6] The final schedule after simulating execution.

Algorithm 4 Constant Step Reprioritization

Input:
$A\langle p\rangle$: List of activities with priorities
U: List of all unscheduled activities (from analyzer)
$step$: Constant step size
Output:
A: Best relative ordering of activities found
1: **for each** $a \in U$ **do**
2: incrementRelativePriority(a, $step$, A)
3: **for each** $d \in a.dependents$ **do**
4: incrementRelativePriority(d, $step$, A)
5: **end for**
6: **for each** $sg \in a.switchGroup$ **do**
7: incrementRelativePriority(sg, $step$, A)
8: **end for**
9: **end for**

Higher priority activities can consume resources (unit resources, energy, and data volume) or change state in a way that prevents lower priority activities from scheduling such that their constraints are satisfied. Increasing the blamed activities' priorities allows them to schedule earlier (scheduling order) which means they have more "slack" to satisfy their constraints. The goal is that the algorithm will slowly promote less flexible activities to the top so that their constraints can be satisfied, and demoted activities are flexible enough to be scheduled in a more constrained plan.

When increasing the relative priorities of blamed activities, the existing relative priorities between certain groups of activities must remain enforced.

First, each switch group must maintain the relative priorities between each activity in the grouping. For each switch group, the activities $(s_1, \ldots, s_n)$ must be ordered such that those with higher resource consumption (time, energy, and data volume) have higher priorities as well.

Second, dependency relationships must be enforced such that (3) is held true.

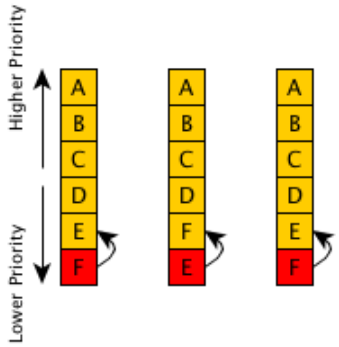


Figure 2: Cycle in the Constant Step approach. Red activities were unable to be scheduled and assigned blamed.

There is one main issue with the Constant Step approach

- it is extremely susceptible to cycles. One common cause for cycles is that a set of activities needs to be promoted beyond a particular activity together, but the constant step size prevents this from ever occurring. For example, in Figure 2 activity F is unschedulable and assigned blame. Its priority is increased, but this causes activity E to fail to schedule. Activity E is then promoted in the next iteration, causing F to fail to schedule and the process repeats. In reality, both E and F have to be promoted above D, but because the step size is constant, they will never achieve that and form a cycle. The situation where activities are unable to be promoted above an activity blocking it can be extended to any constant step size less than the maximum step size [7].

Stochastic Step Reprioritization

Algorithm 5 Stochastic Step Reprioritization

Input:
$A\langle p\rangle$: List of activities with priorities
U: List of all unscheduled activities (from analyzer)
Output:
A: Best relative ordering of activities found

```
step ← random(1, A.length)
for each a ∈ U do
    incrementRelativePriority(a, step, A)
    for each d ∈ a.dependents do
        incrementRelativePriority(d, step, A)
    end for
    for each sg ∈ a.switchGroup do
        incrementRelativePriority(sg, step, A)
    end for
end for
```

Injecting randomness to the step size allows the algorithm to become robust to cycles. In each iteration of the priority setting algorithm, a random step distance between 1 and N, where N is the number of activities in the plan, is assigned to all of the blamed activities. This lets the scheduler always have the possibility of being promoted above a resource constraining activity, while still allowing smaller step size priority permutations.

The main issue that lies with a random approach is that empirically [8] it finds the global maximum score slower than desired. This is further exacerbated by the fact that each iteration of our SWO cycle takes a non-negligible amount of time (a few seconds) due to the need to run a lightweight simulation and Monte Carlo.

Max Step Reprioritization

Stochastic Step Reprioritization (empirically) produced results slower than desired. Max Step Reprioritization seeks to solve both of those issues by always promoting blamed activities to have the highest scheduling priorities. The earlier an activity is considered for scheduling, the more flexibility that activity has to be scheduled. Therefore, if an activity is first to be considered for scheduling, but still cannot be successfully scheduled, there is no other scheduling priority that would allow the activity to be scheduled. Knowing this, by promoting blamed activities to have the highest scheduling priorities we can attempt to avoid iterations that fail to schedule the same blamed activities, thereby speeding up the overall algorithm.

[7] See section Max Step Reprioritization

[8] More information can be found in Empirical Evaluation.

Algorithm 6 Max Step Reprioritization

Input:
$A\langle p\rangle$: List of activities with priorities
U: List of all unscheduled activities (from analyzer)
Output:
A: Best relative ordering of activities found

```
for each a ∈ U do
    step ← A.length
    incrementRelativePriority(a, step, A)
    for each d ∈ a.dependents do
        incrementRelativePriority(d, step, A)
    end for
    for each sg ∈ a.switchGroup do
        incrementRelativePriority(sg, step, A)
    end for
end for
```

Since the blamed activities will have the highest scheduling priorities, cycles such as those seen in Figure 2 can be avoided. However, Max Step Reprioritization doesn't prevent cycles entirely and they still pose an issue when encountered.

Empirical Evaluation

In order to evaluate how well our Priority Search algorithm is able to generate a priority assignment which results in an optimal schedule, we have applied the algorithm to various sets of inputs comprised of activities with their constraints and priorities and compared against various static algorithms. The inputs are derived from *sol types*. *Sol types* are currently the best available data on expected Mars 2020 rover operations (Jet Propulsion Laboratory 2017a). In order to construct a schedule and simulate plan execution, we use the *M2020 surrogate scheduler* - an implementation of the same algorithm as the M2020 onboard scheduler (Rabideau and Benowitz 2017), but implemented for a Linux workstation environment. As such, it is expected to produce the same schedules as the operational scheduler but runs much faster in a workstation environment. The surrogate scheduler is expected to assist in validating the flight scheduler implementation and also in ground operations for the mission (Chi et al. 2018).

Each input file contains approximately 40 activities. We use a probabilistic execution model based on operations data from the Mars Science Laboratory Mission (Jet Propulsion Laboratory 2017b; Gaines et al. 2016a; 2016b) in order to simulate activities completing early by a reasonable amount. In our model to determine activity execution durations, each of the actual execution durations provided in MSL data is first divided by the corresponding predicted execution dura-

tion. Then, we use a linear regression on the scaled values to obtain a mean and standard deviation presuming the ratio of predicted to actual execution times is normally distributed. The value representing the actual execution duration on the regression line for the given conservative duration is used as the mean. A scaled prediction of the actual duration is generated from a a normal distribution using the derived mean and standard deviation. Finally, this value is scaled back by multiplying by the given conservative duration. Note that we do not explicitly change other activity resources such as energy and data volume since they are generally modeled as rates and changing activity durations implicitly changes energy and data volume as well.

Using each of the sol types, we create variants by adding two switch groups to a set of inputs. Each switch group contains three switch cases where the switch cases differ in duration in a manner similar to the one described in (1). Each of the two switch groups are as follows:

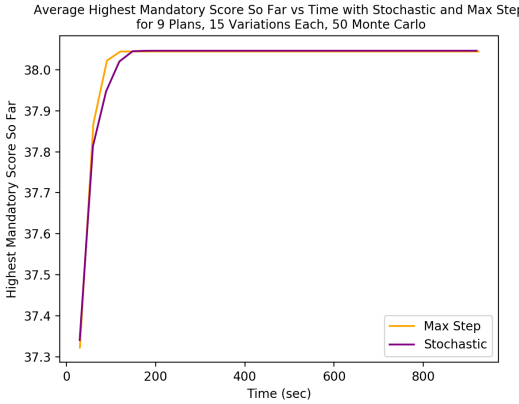
$$SwitchGroup = \begin{cases} Activity_{original} & \text{Duration=}x \text{ sec} \\ Activity_{2x} & \text{Duration=}2x \text{ sec} \\ Activity_{4x} & \text{Duration=}4x \text{ sec} \end{cases} \tag{4}$$

Due to the inequality in (2), a successfully scheduled mandatory activity is of much higher value than a successfully scheduled longer switch case. Therefore, the mandatory activity score is weighted at a much larger value then the switch group score. Each mandatory activity that is successfully scheduled is given one point which contributes to the mandatory score. If a switch case with a duration that is 2 times that of the original activity is able to be scheduled, then it contributes $1/5$ to the switch group score. If a switch case that is 4 times the original duration is able to be scheduled, then it contributes $2/5$ to the switch group. Since there are two switch groups in each variant, the maximum switch group score for a variant is $2 * (2/5) = 4/5$. In the following empirical results, we average the mandatory and switch groups scores over all Monte Carlo runs of execution.

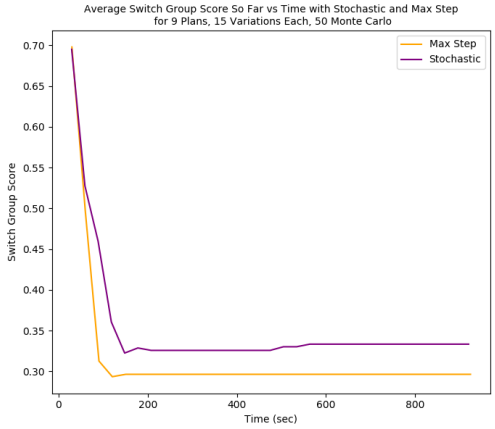
Also, in each of our variants we set the preferred schedule time of each activity to the earliest time the activity is allowed to start.

We first compare the different approaches to implementing Priority Search to understand which performs better.

The highest score so far is a combination of the mandatory score and the switch group score where the mandatory score is weighted at a much higher value than the switch group score. In Figure 3 we plot how the mandatory and switch case components of the highest score achieved up to the current time change over time using both the Stochastic method and the Max Step method. We do not consider the Constant Step method since it is so highly susceptible to cycles. For both methods, as the score for mandatory activities increases, the score for switch groups largely decreases until the highest mandatory score is reached. This is a reasonable outcome because as more mandatory activities are scheduled, the schedule likely becomes more constricted, thus making it more difficult to schedule longer switch cases. Since the mandatory score contributes much more to the total score than the switch group score and the mandatory sore



(a) Mandatory score component of highest score so far vs Time averaged across sol type variants using both priority search methods.



(b) Switch group score component of highest score so far vs Time averaged across sol type variants using both priority search methods.

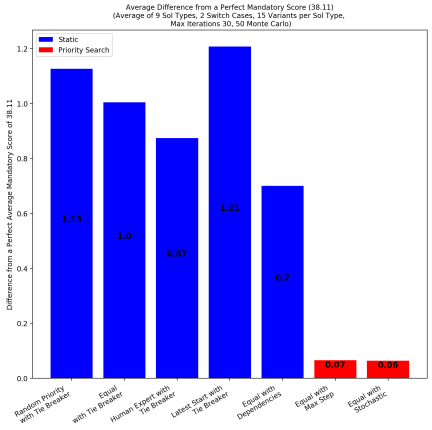
.

Figure 3: Plot of the highest score so far separated by mandatory score (3a) and switch group score (3b) over time using the Stochastic Step method and the Max Step method averaged over 9 sol types, each with 10 variants each containing 2 switch groups. Each iteration of Priority Search was run with 10 Monte Carlo runs and with 30 iterations of Priority Search alloted for each run of the algorithm.

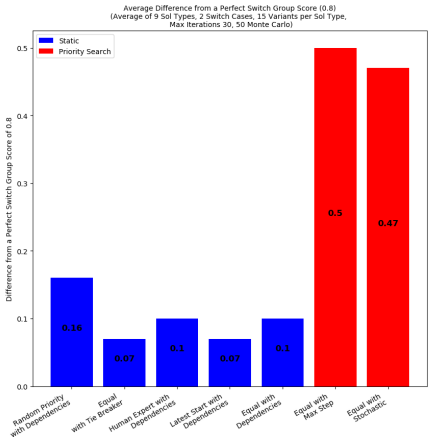
is increasing in both figures, the total highest score so far is always increasing over time, as it should be.

Figure 3a shows that Stochastic Step reaches its highest mandatory score that is ever achieved over the time span of approximately 920 seconds (30 iterations of the priority search algorithm) in 207.58 seconds. The highest mandatory score achieved at this time and onwards is 38.047. The high-

est mandatory score using the Max Step method is reached at 120.59 seconds and has a value of 38.044. Figure 3b shows that the highest switch group score after the point at which the highest mandatory score is reached is 1.67 at 568.16 seconds using the Stochastic method and 1.48 at 150.87 seconds using the Max Step method. Therefore, we conclude that using the stochastic method results in a marginally higher total highest score but it takes less time to reach the highest score using the Max Step method.



(a) Difference from perfect mandatory score averaged across sol type variants for various scheduling methods.



(b) Difference from perfect switch group score averaged across sol type variants for various scheduling methods.

Figure 4: The difference from a perfect mandatory score of 38.11 and perfect switch group score of 1.0 using various scheduling methods is averaged over 9 sol types where 15 variants are derived from each sol type and each variant contains 2 switch cases. Each iteration of the Priority Search algorithm is run with 50 Monte Carlo runs of execution

Figure 4 shows the results of comparisons between Priority Search and other static priority setting algorithms. Since the scheduling of mandatory activities and switch groups are not weighted equally, we have constructed two separate plots to show the results for each. Both methods of Priority Search, in red, result in fewer unscheduled mandatory activities and consequently a lower difference from the perfect mandatory score. This implies they set the priorities such that more mandatory activities are able to be scheduled over multiple Monte Carlo runs compared to how the static algorithms set the activity priorities. As shown in 4b, they result in a higher number of unscheduled switch cases, likely because if more mandatory activities were scheduled it becomes more difficult to schedule longer switch cases. Due to the strict inequality described in (2), even though fewer longer switch cases are scheduled, the total scheduling score is still higher when using Priority Search. Thus, we conclude that both Priority Search methods outperform the static algorithms. Among the static algorithms, running the Dependencies algorithm with Tie Breaker on equal priorities performs the best as it results in the highest mandatory score while running Tie Breaker after setting the priorities based on the latest start time performs the worst.

Related Work

Our Priority Search approach is inspired by Squeaky Wheel Optimization (SWO). Typically, SWO uses a constructor and analyzer, and prioritizer for the next iteration of schedule generation (Joslin and Clements 1999). Priority Search differs in that the intent is not to generate a good schedule but rather to set priorities that perform well in execution and rescheduling. Therefore the Priority Search constructor must use the scheduler through multiple runs of execution (where each run of execution incurs multiple scheduler invocations) to assess priority assignment performance.

Generating schedules that are robust to execution run time variations (Leon, Wu, and Storer 1994) is a mature area of work. However, the topic usually revolves around developing a scheduler that can generate robust schedules. In our case, the scheduler is a) a fixed ”black box” that we have no control over and b) robust to execution run time variations mainly through rescheduling (Chi et al. 2018). As a result, rather than developing a scheduler itself, we’re developing a methodology that is able to generate a set of priorities for a fixed scheduler that enables it to be robust to rescheduling due to runtime variations.

Other approaches (Drummond, Bresina, and Swanson 1994; Washington, Golden, and Bresina 2000) use branching to increase robustness - these differ from our work that adjusts priorities and allows rescheduling.

A number of other spacecraft (Muscettola et al. 1998; Pell et al. 1997; Chien et al. 2005; 2016) and rover (Woods et al. 2009; Gregory et al. 2002) autonomy systems have included planning, but these differ in that we are deriving control information specific to scheduling for a limited context - e.g. one rover sol. temporal schedule.

Discussion and Future Work

While we have focused on the impact of activity priority on the scheduler (and hence rescheduling during execution), there is often an execution system that may also have some flexibility to add robustness to the overall system (Chi et al. 2018). For the empirical evaluation described above, we ran without such an execution system. In the future, we could consider the execution system in the schedule and Monte Carlo analysis and potentially derive information usable by the execution system (e.g. allow an activity to run late but only until time T). This paper describes initial work to determine priorities for scheduler activity consideration ordering to optimize scheduler execution results for an embedded scheduler. However, this work is still preliminary with many other ideas to be explored as described below.

First, more sophisticated critique/blame assignment methods should be explored. Currently, priorities of activities that are not executed are modified, but more sophisticated analysis of scheduler runs could provide greater insight into how the priorities should be modified. Prior work in Process Chronologies (Biefeld and Cooper 1991) has been used to focus scheduling tactics by finding regions where time constraints or high demand for some resource results in conflict. By evaluating which periods of time or what resources are over-subscribed using Capacity/Over-Subscription Analysis, we can pinpoint which activities are more tightly constrained and increase their priorities. Prior work in Oversubscribed Scheduling Problems (Kramer and Smith 2006) show that scheduling according to maximum-availability (least subscribed) allows a suitable schedule to be generated. Similar analysis could be used to determine which activities to assign blame to and by how much to promote the blamed activities. We can also consider precedence constraints when deciding by how much to promote activity priorities. For every blamed activity, there is likely a scheduled activity that is using resources needed by the blamed activity. Precedence constraints could help discern which activity is using those resources. The blamed activity could then be promoted only as much as is necessary in order to be scheduled before the offending activity.

We can also implement several methods to help us explore different search spaces. Priority Search only adjusts priorities to improve execution and rescheduling performance. We could also add new activity precedence constraints (e.g. A must end before B starts) or enforce partitions in the schedule (e.g. all of these activities must be scheduled to end prior to 11 am). These types of constraints could drive the scheduler towards subsets of the schedule search space. Randomized restart can allow our priority search algorithm to better explore the global space rather than searching locally. Another alternative would be to keep a list of promising schedule priority assignments and backtrack to those randomly, allowing us to better explore the search space.

We can also make improvements to our Monte Carlo method and use the resulting simulations for further analysis of the scheduler. In order to build a model of run time variations that is not overly skewed, we use Monte Carlo to repeatedly sample a variety of execution run time results. Standard Monte Carlo simulations tend to focus most runs on the nominal cases, but a more effective methodology samples edge cases but weighs the cases by their likelihood to increase coverage of the variability in the space (in this case variable activity execution times). The Monte Carlo of execution run time variations can provide valuable information for why activities fail to schedule, what input plans are best suited for the current scheduler design, and how the current input could end up executing. We are working on visualizing this information to better inform those working with the scheduler.

Currently, we only test with mandatory activities. In the future, we will extend our approach to include optional activities, which will add further complexity to the algorithm and analysis. Optional activities are lower priority activities what are nice to have scheduled, but not necessary. They are generally only able to be scheduled if mandatory activities end early or consume less resources than expected. We also plan to use an activity's actual scheduled preferred time while testing.

Cycles pose an issue to both Constant Step Reprioritization and Max Step Reprioritization. Better cycle detection would allow us to not only overcome the issues presented, but also provide additional information on how to permute the priority set for the next iteration. For example, cycle detection could allow us to not only detect the cycle in Figure 2, but know that both E and F should be incremented together.

While we have established a few methods to improve the prioritizer and decide on the next permutation of activity priorities, we have utilized the same objective function to determine the success of our algorithm. However, our objective function is simple and coarse; oftentimes, the same score will appear repeatedly in multiple consecutive. As a result, the algorithm often travels swaths of plateaus before sharply improving. This choppiness is suboptimal for Squeaky Wheel Optimization and gradient descent problems in general. Some potential additions to the objective function could be how much energy is leftover in the plan or how close an activity is to their preferred scheduling time. Evaluating a more precise objective function can reduce choppiness and better steer the algorithm towards a more optimal solution.

Conclusion

We have presented a study of methods to assign activity priorities to control a limited, embedded scheduler to optimize rescheduling for a specific problem. We first define a set of static methods that assign activity priorities based on heuristics and schedule dependencies. We then describe how these priorities can be further adjusted based on feedback from simulated execution and rescheduling using Monte Carlo methods to perform Priority Search. We present an empirical evaluation of several static and priority search methods using best available planetary rover operations data. This empirical evaluation shows that Priority Search outperforms static methods including human expert derived priorities. Finally we describe a number of promising areas for future improvements to our algorithms.

Acknowledgments

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

Biefeld, E., and Cooper, L. 1991. Bottleneck identification using process chronologies. In *IJCAI*, 218–224.

Chi, W.; Chien, S.; Agrawal, J.; Rabideau, G.; Benowitz, E.; Gaines, D.; Fosse, E.; Kuhn, S.; and Biehl, J. 2018. Embedding a scheduler in execution for a planetary rover. In *ICAPS*.

Chien, S. A.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Mandl, D.; Trout, B.; Shulman, S.; et al. 2005. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing Information and Communication* 2(4):196–216.

Chien, S.; Doubleday, J.; Thompson, D. R.; Wagstaff, K. L.; Bellardo, J.; Francis, C.; Baumgarten, E.; Williams, A.; Yee, E.; Stanton, E.; et al. 2016. Onboard autonomy on the intelligent payload experiment cubesat mission. *Journal of Aerospace Information Systems*.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *AAAI*, volume 94, 1098–1104.

Gaines, D.; Anderson, R.; Doran, G.; Huffman, W.; Justice, H.; Mackey, R.; Rabideau, G.; Vasavada, A.; Verma, V.; Estlin, T.; et al. 2016a. Productivity challenges for mars rover operations. In *Proceedings of 4th Workshop on Planning and Robotics (PlanRob)*, 115–125. London, UK.

Gaines, D.; Doran, G.; Justice, H.; Rabideau, G.; Schaffer, S.; Verma, V.; Wagstaff, K.; Vasavada, A.; Huffman, W.; Anderson, R.; et al. 2016b. Productivity challenges for mars rover operations: A case study of mars science laboratory operations. Technical report, Technical Report D-97908, Jet Propulsion Laboratory.

Gregory, N. M.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International Workshop on Planning and Scheduling for Space*. Citeseer.

Jet Propulsion Laboratory. 2017a. Mars 2020 rover mission https://mars.nasa.gov/mars2020/ retrieved 2017-11-13.

Jet Propulsion Laboratory. 2017b. Mars science laboratory mission https://mars.nasa.gov/msl/ 2017-11-13.

Joslin, D. E., and Clements, D. P. 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 10:353–373.

Kramer, L. A., and Smith, S. F. 2006. Resource contention metrics for oversubscribed scheduling problems. In *ICAPS*, 406–409.

Leon, V. J.; Wu, S. D.; and Storer, R. H. 1994. Robustness measures and robust scheduling for job shops. *IIE transactions* 26(5):32–43.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103(1-2):5–47.

Pell, B.; Gat, E.; Keesing, R.; Muscettola, N.; and Smith, B. 1997. Robust periodic planning and execution for autonomous spacecraft. In *International Joint Conference on Artificial Intelligence*, 1234–1239.

Rabideau, G., and Benowitz, E. 2017. Prototyping an onboard scheduler for the mars 2020 rover. In *International Workshop on Planning and Scheduling for Space*.

Washington, R.; Golden, K.; and Bresina, J. 2000. Plan execution, monitoring, and adaptation for planetary rovers. *Electron. Trans. Artif. Intell.*

Woods, M.; Shaw, A.; Barnes, D.; Price, D.; Long, D.; and Pullan, D. 2009. Autonomous science for an exomars rover–like mission. *Journal of Field Robotics* 26(4):358–390.

Off-line/On-line Optimization under Uncertainty

Allegra De Filippo, Michele Lombardi and **Michela Milano**
DISI, University of Bologna
allegra.defilippo@unibo.it, michele.lombardi2@unibo.it, michela.milano@unibo.it

Abstract

In this work we present two general techniques to deal with multi-stage optimization problems under uncertainty, featuring off-line and on-line decisions. The methods are applicable when: 1) the uncertainty is exogenous; 2) there exists a heuristic for the on-line phase that can be modeled as a parametric convex optimization problem. The first technique replaces the on-line heuristics with an anticipatory solver, obtained through a systematic procedure. The second technique consists in making the off-line solver aware of the on-line heuristic, and capable of controlling its parameters so as to steer its behavior. We implement our approaches on two case studies: an energy management system with uncertain renewable generation, and load demand and a vehicle routing problem with uncertain travel times. We show how both techniques achieve high solution quality w.r.t. an oracle operating under perfect information, while striking different trade-offs in terms of computation time.

1 Introduction

Dealing with uncertainty in optimization problems is challenging, but also increasingly recognized as necessary to obtain practically relevant results (Powell 2016). In many cases, this class of problems features both an off-line "strategic" phase that can be tackled with relative leisure, and an on-line "operational" phase where decisions need to be taken under stringent time constraints. Since optimization under uncertainty is tough, anticipatory methods such as stochastic optimization (see (Shapiro and Philpott 2007; Birge and Louveaux 1997; Kall, Wallace, and Kall 1994)) have been historically employed for the off-line phase, and simple non-anticipativity heuristics for the on-line phase.

However, on-line algorithms have the ability to exploit additional information as the uncertainty is slowly revealed. With the aim of tapping into this potential, a growing number of works has been addressing on-line problems via techniques originally introduced for stochastic programming, e.g. sampling and the Sample Average Approximation (Shapiro 2013). Sampling refers to obtaining realizations (scenarios) of the random variables used to model the uncertainty; by solving deterministic optimization problems over multiple scenarios and by computing averages, it is possible to enrich an on-line algorithm with some degree of anticipation. These developments lead to the EXPECTATION (Chang, Givan, and Chong 2000), CONSENSUS (Bent and Van Hentenryck 2004b) and REGRET (Bent and Van Hentenryck 2004a) algorithms, and to more advanced methods such as AMSAA (Hentenryck and Bent 2009; Mercier and Van Hentenryck 2008). All such methods are well discussed in (Hentenryck and Bent 2009).

There is a delicate trade-off between speculating vs. waiting for the uncertainty to be resolved and this leads to an informal distinction between off-line and on-line problems. In particular, on-line algorithms require to make decisions over time and delaying decisions can either increase the costs or be impossible due to constrained resources. Off-line problems are often solved via exact solution methods on approximate models with limited look-ahead, e.g. via decomposition based methods (Laporte and Louveaux 1993). The next logical step seems to find methods to integrate off-line and on-line decision making. This paper proposes two methods that are applicable when: 1) the uncertainty is exogenous; 2) there exists a heuristic for the on-line phase with certain basic properties. Each method alters either the off-line or the on-line part of the solution process, so that the two play better together. We believe our techniques represent a significant step toward integrated off-line/on-line optimization.

We implement our approaches on two case studies: 1) an energy system management problem, where load shifts are planned off-line and power flows must be controlled on-line; and 2) a Vehicle Routing Problem where customers are assigned off-line, but the routes can be chosen on-line. We show how the two methods strike radically different trade-offs in terms of off-line and on-line complexity, but they achieve solutions of high quality.

The rest of the paper is organized as follows: Section 2 formalizes our two solution methods. Section 3 shows how our methods can be implemented. Section 4 reports experimental results. Concluding remarks are in Section 5.

2 Formalization

We consider multi-stage optimization problems under uncertainty, where the first stage (indexed with 0) involves off-line decisions, and all subsequent n stages involve on-line decisions. We will start by describing a baseline solution approach, and then improve it by: 1) adding anticipation to the on-line solver through a systematic procedure; and 2) making the off-line solver aware of the on-line one, and capable

of controlling its parameters so as to steer its behavior. The first method is related to existing on-line anticipative algorithms; the second relies on the mixed nature of the problem.

Formally, let y represent the off-line decisions; let x^k represent the on-line decisions for stage k; let s^k (resp. ξ^k) represent the system state (resp. the uncertainty) revealed at the beginning (resp. the end) of stage k. All variables are assumed to be vectors, they can be numeric or discrete, and have finite or infinite domain.

Baseline on-line heuristic (PH) : We assume the availability of an on-line heuristic that can be modeled as a parametric convex (and therefore efficient) optimization problem:

$$\min f(y, x^k, s^k; \alpha^k) \qquad (\mathbf{PH})$$
$$\text{s.t. } e(y, x^k, s^k) = 0 \qquad (1)$$
$$g(y, x^k, s^k) \leq 0 \qquad (2)$$

where f is the cost function with parameter vector α^k, while e and g are the constraint functions (with vector output). We assume the optimization problem to be convex, which means that f and g must be convex and e to be linear. Typically, the convexity requirement will prevent x^k from being integer, but there are important exceptions (e.g. the one in Sec. 3.2).

We will refer as $\mathcal{F}(y, x^k, s^k)$ to the actual cost incurred at stage k for taking decisions x^k. Note this is distinct from the objective function f of the heuristic, although in practice the two are likely to be based on similar formulas. The transition from the state in stage k to the state in stage $k+1$ is defined by means of a transition function T, i.e.:

$$s^{k+1} = T(y, x^k, s^k, \xi^k)$$

where it can be seen that the effect of the uncertainty (i.e. the random variable) is encoded in the state.

Flattened Problem (PF) : Let Ω be a set of scenarios ω from the sample space of $\xi = (\xi^0, \dots \xi^{n-1})$. Given a single scenario ω, it is possible to collapse the instantiations of **PH** for each stage to obtain a flattened (on-line) problem:

$$\min \sum_{k=1}^{n} \mathcal{F}(y, x^k_\omega, s^k_\omega) \qquad (\mathbf{PF})$$
$$\text{s.t. } e(y, x^k_\omega, s^k_\omega) = 0 \qquad \forall k = 1..n \qquad (3)$$
$$g(y, x^k_\omega, s^k_\omega) \leq 0 \qquad \forall k = 1..n \qquad (4)$$
$$s^{k+1}_\omega = T(y, x^k_\omega, s^k_\omega, \xi^k_\omega) \qquad \forall k = 1..n-1 \qquad (5)$$

where $x^k_\omega / s^k_\omega / \xi^k_\omega$ are the on-line decisions/state/realizations for stage k in scenario ω. The only actual adjustment w.r.t. instantiating **PH** multiple times is that the true cost $\mathcal{F}$ is used rather than the heuristic cost function. Since **PF** assumes the availability of all ξ^k_ω values, it is effectively a clairvoyant approach. In the on-line optimization literature the flattened problem is known as the off-line problem (see (Hentenryck and Bent 2009)), but we adopt a different name to avoid ambiguity with the actual off-line phase.

Since the on-line problem can be solved with relative ease, the complexity depends heavily on the properties of the state transition function. If T is linear (e.g. in Sec. 3.1), then the flattened problem will be convex and relatively easy to solve. Non-linear transition functions (e.g. Sec. 3.2) are conversely much harder to handle.

Baseline off-line problem (PO) : As a baseline to deal with the off-line decisions we consider a two-stage stochastic optimization problem obtained by instantiating **PF** once per scenario:

$$\min f_o(y) + \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{k=1}^{n} \mathcal{F}(y, x^k_\omega, s^k_\omega) \qquad (\mathbf{PO})$$
$$\text{s.t. } Eq.(3) - (5) \qquad \forall \omega \in \Omega$$
$$s^1_\omega = T_o(y, \xi^0_\omega) \qquad \forall \omega \in \Omega \qquad (6)$$
$$y \in \mathcal{Y} \qquad (7)$$

where the function $f_o(y)$ represents the cost that depends directly on the off-line decisions. The remainder of the cost function is given by the Sample Average Approximation of the expected cost of the subsequent stages. The function $T_o(y, \xi^0_\omega)$ determines the initial state for the on-line stages, based on the value of y and on the uncertainty revealed at the end of the off-line stage (i.e. ξ^0_ω). Finally, $\mathcal{Y}$ is the feasible space for the off-line decision variables y. We make no special assumption on $\mathcal{Y}$, $f_o(y)$, and $T_o(y, \xi^0_\omega)$, meaning that even when the flattened problem is convex the off-line problem may be NP-hard (or worse). Still, the fact that the problem is solved off-line makes its complexity less critical.

The biggest drawback of this approach is that using the flattened problem to estimate the effect of the off-line decision on the future stages is equivalent to assuming the availability of an oracle. In practice, however, an on-line approach can behave much worse than an oracle-powered solver.

Anticipatory On-line Phase (PB) : In this context, off-line on-line integration can be obtained by providing the on-line algorithm with something that resembles an oracle, i.e. by making it anticipative. A simple approach to achieve this is the one employed for the baseline off-line problem, i.e. instantiating **PF** for the remaining stages. Formally, let h be the index of the current stage, then we consider:

$$\min \mathcal{F}(y, x^h, s^h) + \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{k=h+1}^{n} \mathcal{F}(y, x^k_\omega, s^k_\omega) \qquad (\mathbf{PB})$$
$$\text{s.t. } Eq.(1), (2) \qquad \text{for stage } h$$
$$Eq.(3), (4) \qquad \text{for } k > h$$
$$Eq.(5) \qquad \text{for } k \geq h, \text{ with } s^h_\omega = s^h \text{ and } x^h_\omega = x^h$$

The off-line decisions are taken using **PO**. This first approach, named Boosted On-line OptimizatoN (BOON), improves the accuracy of the on-line component at the expense of its solution time.

PB has the same semantic of the EXPECTATION algorithm, which obtains the same results by enumerating the feasible decisions for the current stage, and evaluating the expected cost by solving the flattened problem on each scenario. Since each scenario is considered in isolation, EXPECTATION is arguably much more efficient than BOON whenever the current stage decisions can be enumerated reason-

ably fast. The same argument applies to the REGRET algorithm, an efficient approximation of EXPECTATION.

However, when the decision space is not enumerable (e.g. for continuous x^k variables, as in Sec. 3.1), EXPECTATION, REGRET (and even CONSENSUS and AMSAA) cannot be applied directly, while our method is still viable. Moreover, when each on-line stage requires to take multiple decisions, enumeration may be expensive and associating an expected cost to each decision in isolation leads to underestimations if the costs are not additive ((Awasthi and Sandholm 2009)).

On-line Aware Off-line Phase (PM) : An alternative integration approach consists in making the off-line solver aware of the on-line heuristic. Moreover, we can allow the off-line phase to adjust the heuristic parameters so as to steer its behavior. We start by observing that, since the on-line heuristic problem **PH** is convex, any local minimum must be a global minimum. Local minima can be characterized in terms of the Karush-Kuhn-Tucker optimality conditions (Winston 2004), which for **PH** in a given scenario ω are given by:

$$-\nabla_{x^k_\omega} f = \sum_{i=1}^{|e|} \lambda^k_{\omega,i} \nabla_{x^k_\omega} e_i + \sum_{i=1}^{|g|} \mu^k_{\omega,i} \nabla_{x^k_\omega} g_i \qquad (8)$$

$$\mu^k_{\omega,i} g_i = 0 \qquad \forall i = 1..|g| \qquad (9)$$

$$\mu^k_{\omega,i} \geq 0 \qquad \forall i = 1..|g| \qquad (10)$$

$$Eq.(1), (2)$$

where, for sake of readability, $f(y, x^k_\omega, s^k_\omega; \alpha_k)$ has been shortened to f, the i-th component (out of $|e|$) of $e(y, x^k_\omega, s^k_\omega)$ to e_i, and the i-th component (out of $|g|$) of $g(y, x^k_\omega, s^k_\omega)$ to g_i. The $\lambda^k_{\omega,i}$ and $\mu^k_{\omega,i}$ variables represent dual multipliers. Eq. (8) corresponds to the gradient cancellation condition, Eq. (9) to complementary slackness, Eq. (10) to dual feasibility ($\lambda^k_{\omega,i}$ is free), and Eq. (1), (2) to primal feasibility. Note that here we use the actual heuristic cost, parameterized in α^k.

Now, the KKT conditions can be injected as constraints in **PO**. This will force all x^k_ω variables in the off-line problem to take the values that would be actually assigned by the heuristic. This leads to the following problem:

$$\min f_o(y) + \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{k=1}^{n} \mathcal{F}(y, x^k_\omega, s^k_\omega) \qquad (\mathbf{PM})$$

$$\text{s.t. } Eq.(3) - (5) \qquad \forall \omega \in \Omega$$

$$Eq.(6), (7)$$

$$Eq.(8) - (10) \qquad \forall \omega \in \Omega, \forall k = 1 \dots n$$

The decision variables are in this case y, x^k_ω, s^k_ω, $\lambda^k_{\omega,i}$, $\mu^k_{\omega,i}$, and crucially α^k. The on-line decisions are then taken using the original heuristics, but its behavior will be affected by the "parameter schedule" $\alpha^1, \dots \alpha^n$ produced by solving **PM**.

We named this second approach Master Off-line OptimizatioN (MOON): it achieves integration at the cost of off-line solution time, because of the additional variables in **PM** and the presence of non-linearities in Eq. (9). The approach has some similarities with Logic Based Benders Decomposition (LBBD): in our case, however, the KKT conditions provide an exact model of the subproblem rather than a relaxation to be iteratively defined. As an interesting future development, it should be possible to use LBBD to solve our **PM**.

3 Case Studies

In this section we present our case studies. The first one (an energy management system) was considered in (De Filippo et al. 2017): since it features continuous on-line decision variables, it is not amenable to existing approaches such as EXPECTATION or REGRET. The second use case (a Vehicle Routing Problem variant) is meant to provide a realistic, different, example of how the methods can be implemented: it features discrete on-line decisions, and allows a quality comparison with classical algorithms because in such cases BOON leads to the same results as EXPECTATION (with no solution time restrictions). For simplicity, we will make heavy use of monolithic models: our methods work however with any solution approach, provided that the correct problems are solved.

3.1 Energy Management System

We consider a Virtual Power Plant (VPP) management system (see (Morales et al. 2013)) with partially shiftable loads, renewable energy generators, storage systems, and grid-connection. The VPP concept is based on the idea of aggregating the capacity of many Distributed Energy Resources (DER) to create a single operating profile to increase flexibility to manage the uncertainty. The load shifts must be planned off-line and the energy balance should be maintained on-line. The goal is to decide the minimum-cost energy flows at each on-line stage (see (Clavier et al. 2015)), i.e.: 1) how much energy should be bought; 2) which generators should be used; 3) whether the surplus energy (if any) should be stored or sold to the market.

The uncertainty stems from uncontrollable deviations from the planned shifts and from the presence of Renewable Energy Sources (RES) (see (Palma-Behnke et al. 2011; Bai et al. 2015)). We assume that the RES production forecast is good enough that its error in each stage can be considered an independent random variable.

Baseline Heuristic and Transition Function: Based on the shifts produced by the off-line step, and adjusted to take into account the uncertainty, the on-line heuristic minimizes the operational cost and covers the energy demand by manipulating flows between nodes in $g \in G$. We assume the index 0 refers to the storage system and index 1 to the RES generators. The stages represent periods long enough to treat the corresponding flow decisions as independent. The heuristic can be formulated as an LP model:

$$\min \sum_{k=1}^{n} \sum_{g \in G} c^k_g x^k_g \qquad (\mathbf{P1.1})$$

$$\text{s.t. } \tilde{L}^k = \sum_{g \in G} x^k_g \qquad (11)$$

$$0 \leq \gamma_k + \eta x^k_0 \leq \Gamma \qquad (12)$$

$$\underline{x}_g \leq x^k_g \leq \overline{x}_g \qquad (13)$$

where n is the number of on-line stages, and x_g^k represents the flow from g to the VPP (if positive) or in the reverse direction (if negative). All flows must respect the physical bounds $\underline{x}_g$ and $\overline{x}_g$. The flow costs c_g^k correspond to the problem parameters α^k in **PH**. The state variables are the RES energy flow x_1^k, the load to be satisfied $\tilde{L}^k$, and the battery charge γ^k. The battery upper limit is Γ and η is the charging efficiency. The off-line decisions do not appear directly in the heuristic model, but they affect instead the state transition function:

$$\gamma^{k+1} = \gamma^k + \eta x_0^k \tag{14}$$
$$x_1^{k+1} = \hat{R}^k + \xi_R^k \tag{15}$$
$$\tilde{L}^{k+1} = \hat{L}^k + y^k + \xi_L^k \tag{16}$$

where $\hat{R}_k$ and $\hat{L}^k$ are the estimated RES production and load, and ξ_R^k and ξ_L^k are the corresponding errors (random variables). We assume that the errors follow roughly a Normal distribution $N(0, \sigma^2)$, and that the variance σ^2 is such that the 95% confidence interval corresponds to $\pm 20\%$ of the estimated value (Gamou, Yokoyama, and Ito 2002). The y^k variable represents the (off-line planned) shift from the estimated load.

Baseline Off-line Problem: The off-line problem is modeled via Mixed Integer Programming (MILP):

$$\min \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{g \in G} \sum_{k=1}^{n} c_g^k x_{g,\omega}^k \qquad (\mathbf{P1.2})$$
$$\text{s.t. } \tilde{L}_\omega^k = \sum_{g \in G} x_{g,\omega}^k \qquad \forall \omega \in \Omega, \forall k = 1, \dots n \tag{17}$$
$$\underline{x}_g \leq x_{g,\omega}^k \leq \overline{x}_g \qquad \forall \omega \in \Omega, \forall k = 1, \dots n \tag{18}$$
$$0 \leq \gamma_\omega^k \leq \Gamma \qquad \forall k = 1, \dots n \tag{19}$$
$$\gamma_\omega^{k+1} = \gamma_\omega^k + \eta x_{0,\omega}^k \quad \forall \omega \in \Omega, \forall k = 1, \dots n-1 \tag{20}$$
$$x_{1,\omega}^{k+1} = \hat{R}_k + \xi_{R,\omega}^k \qquad \forall \omega \in \Omega, \forall k = 1, \dots n \tag{21}$$
$$\tilde{L}_\omega^{k+1} = \hat{L}_k + y_k + \xi_{L,\omega}^k \quad \forall \omega \in \Omega, \forall k = 1, \dots n \tag{22}$$
$$\sum_{k=t}^{t+m} y_k = 0 \qquad \forall t = 1, \dots n-m \tag{23}$$
$$\underline{y}^k \leq y_k \leq \overline{y}^k \qquad \forall k = 1, \dots n \tag{24}$$

where Eq. (17) – (22) define the flattened problem, and Eq. (23) – (24) the feasible space for the off-line variables y. Eq. (23) ensures that the shifts respect a local balance. The initial battery charge γ_ω^0 is identical for all scenarios.

Implementing BOON for the VPP: A model for the BOON approach can be obtained by applying in an almost straightforward fashion the definitions from Sec. 2:

$$\min \sum_{g \in G} c_g^h x_g^h + \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{k=h+1}^{n} \sum_{g \in G} c_g^k x_{g,\omega}^k \qquad (\mathbf{P1.3})$$
$$\text{s.t. } Eq.\ (11) - (13)$$
$$Eq.\ (17) - (19) \qquad \forall k > h$$
$$Eq.\ (20) - (22) \qquad \forall k \geq h, \text{ with } s_\omega^h = s^h \text{ and } x_\omega^h = x^h$$

Note that **P1.3**, although potentially large, is a Linear Program and can be solved in polynomial time.

Implementing MOON for the VPP: We start by formulating the KKT conditions for the on-line heuristic in a single scenario, thus obtaining:

$$-c_g^k = \lambda_\omega^k + \mu_{g,\omega}^k - \nu_{g,\omega}^k \qquad \forall g \in G \tag{25}$$
$$\mu_{g,\omega}^k (x_{g,\omega}^k + \overline{x}_g) = 0 \qquad \forall g \in G \tag{26}$$
$$\nu_{i,\omega}^k (\underline{x}_g - x_{g,\omega}^t) = 0 \qquad \forall g \in G \tag{27}$$
$$\hat{\mu}_\omega^k (\eta x_{0,\omega}^k + \gamma^k - \Gamma) = 0 \tag{28}$$
$$\hat{\nu}_\omega^k (\eta x_{0,\omega}^k + \gamma^k) = 0 \tag{29}$$
$$\mu_{g,\omega}^k, \nu_{g,\omega}^k \geq 0 \qquad \forall g \in G \tag{30}$$
$$\hat{\mu}_\omega^k, \hat{\nu}_\omega^k \geq 0 \tag{31}$$

where $\mu_{g,\omega}^k$ and $\nu_{g,\omega}^k$ are the multipliers associated to the physical flow bounds, while $\hat{\mu}_\omega^k$ and $\hat{\nu}_\omega^k$ are associated to the battery capacity bounds. The multiplier λ_ω^k is associated to the balancing constraint, i.e. Eq. (11), and can be eliminated with a few algebraic transformations. Injecting the conditions in the off-line model yields:

$$\min \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{g \in G} \sum_{k=1}^{n} c_g^k x_{g,\omega}^k \qquad (\mathbf{P1.4})$$
$$\text{s.t. } Eq.\ (17) - (24)$$
$$Eq.\ (25) - (31) \qquad \forall \omega \in \Omega, \forall k = 1, \dots n$$

where the decision variables are $y^k, x_{g,\omega}^k, \mu_{g,\omega}^k, \nu_{g,\omega}^k, \hat{\mu}_\omega^k, \hat{\nu}_\omega^k$. To those, we add the cost c_0^k associated to the flow between the VPP and the storage system (the only parameter that we allow the solver to adjust). Normally, there are neither economic penalties nor incentives for such flow, while there is a profit associated to flows from the VPP to the grid. As a side effect, the naive **P1.1** heuristic will always choose to sell the surplus energy. MOON allows the off-line solver to associate a "virtual profit" to storing energy, which enables addressing the original limitation at no on-line computational cost.

3.2 Vehicle Routing Problem

We consider a variant of the Capacitated VRP with uncertain travel times (Toth and Vigo 2002; Bertsimas and Simchi-Levi 1996; Lee, Lee, and Park 2012; Taş et al. 2013). The problem consists in establishing the paths of a set of vehicles to serve a set of customers. All vehicles have a finite capacity, and customers have a known demand and can be visited by a single vehicle. There are n fully connected customers/nodes, with node 0 being the (single) depot. Customer assignments must be done off-line, while the vehicle

routes are chosen on-line. We assume that, whenever a node is reached, its binary "state" becomes known, and with that the (uniform) distributions followed by the travel times of all its outgoing arcs. Formally, this results in bimodally distributed, statistically dependent, travel times. The objective is to minimize the total travel time.

Baseline Heuristic and Transition Function: The on-line heuristic consists in simply picking the outgoing arc with the shortest travel time. This can be modeled also as a simple Integer Program. Let h be the current node, then we have:

$$\min \sum_{j \in V_h} c_{hj} x_{hj} \qquad (\mathbf{P2.1})$$

$$\text{s.t.} \sum_{j \in V_h} x_{h,j} = 1 \tag{32}$$

$$x_{h,j} \in \{0,1\} \qquad \forall j \in V \tag{33}$$

where $x_{hj} = 1$ iff we choose to move from h to j, V_h is the set of nodes that still needs to be visited (and it always include the depot), and the travel times c_{hj} are the heuristic parameters. **P2.1** does not apparently satisfy our assumptions, due to the integer variables. However, its LP relaxation has always an integer solution, banning degenerate cases (i.e. arcs with the same cost). We can therefore relax the integrality requirement without loss of generality. The transition function is given by:

$$V_{h^*} = V_h \setminus \{h^*\} \tag{34}$$

$$c_{h^*,j} = \xi_{h^*,j} \tag{35}$$

where h^* is index of the next node selected by the heuristic and $\xi_{h^*,j}$ is the travel time from h^* to j (a random variable). Note also that in this case the index of the on-line stage is implicitly given by h. We take advantage of this and reduce the notation clutter by moving the ω index to apex position.

Baseline Off-line Problem: We tackle the off-line problem via Mixed Integer Linear Programming, which forbids us to directly embed the non-linear Eq. (34) in the model. In practice, however, the equation states that 1) each vehicle should serve only its assigned customers, and 2) the visit should form a single loop. Both are well known VRP constraints and can be linearized. In particular, we use the model:

$$\min \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{k \in K} \sum_{i,j \in V} \xi^{\omega}_{i,j} x^{\omega}_{k,i,j} \qquad (\mathbf{P2.2})$$

$$\text{s.t.} \sum_{j \in V} x^{\omega}_{k,i,j} = y_{k,i} \qquad \forall k \in K, \forall i \in V \tag{36}$$

$$\sum_{i \in V} x^{\omega}_{k,i,j} = y_{k,j} \qquad \forall k \in K, \forall j \in V \tag{37}$$

$$y_{k,0} = 1 \qquad \forall k \in K \tag{38}$$

$$t^{\omega}_{k,j} \geq t^{\omega}_{k,i} - M + (M+1) x^{\omega}_{k,i,j} \quad \forall k \in K, \forall i,j \in V, V^+ \tag{39}$$

$$t^{\omega}_{k,0} = 0 \qquad \forall k \in K \tag{40}$$

$$\sum_{i \in V} q_i y_{k,i} \leq C_k \qquad \forall k \in K \tag{41}$$

$$\sum_{k \in K} y_{k,i} = 1 \qquad \forall i \in V^+ \tag{42}$$

where all constraints where an ω apex appears should be posted $\forall \omega \in \Omega$. All x and y variables are binary, and $y_{ki} = 1$ iff customer i should be visited by vehicle k. We have $M = |V|$, and $V^+ = V \setminus \{0\}$. Eq.(36) – (40) define the flattened problem, and Eq. (41) – (42) define the feasible space of the off-line decision variables. For sake of simplicity, we eliminate subloops by keeping track of the visiting order t^{ω}_{ki} of each node for each vehicle: this is a simple, but not particularly effective method, because it relies on big-Ms and reduces the quality of the LP bound.

Implementing BOON for the VRP: The BOON method can be implemented for each vehicle k separately, by first restricting the focus to the set of nodes V_h, and then by applying the definition from Sec. 2 and linearizing Eq. (34) in the baseline off-line problem, we get:

$$\min \sum_{j \in V_h} c_{h,j} x_{k,i,j} + \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{i,j \in V_h} \xi^{\omega}_{i,j} x^{\omega}_{k,i,j} \qquad (\mathbf{P2.3})$$

$$\text{s.t. } Eq.\ (32)$$

$$Eq.\ (36) \text{ restricted to } V_h \setminus \{0\} \tag{43}$$

$$Eq.\ (37) - (39) \text{ restricted to } V_h$$

$$t^{\omega}_{k,h} = 0 \tag{44}$$

where Eq. (44) means that the vehicle path should start from the current node h (and end as usual in the depot).

Implementing MOON on the VRP: As usual, the first step in the MOON is formulating the KKT conditions for **P2.1**. In this case after some algebraic transformations, for a given vehicle k, node h, and scenario ω we obtain:

$$(c_{hj} + \lambda^{\omega}_{k,h}) x^{\omega}_{k,h,j} = 0 \qquad \forall j \in V_h \tag{45}$$

$$(c_{hj} + \lambda^{\omega}_{k,h}) \geq 0 \qquad \forall j \in V_h \tag{46}$$

where $\lambda^{\omega}_{k,h}$ is the multiplier for Eq. (32), and all other multipliers have been eliminated. The main difficulty is again dealing with the set V_h, which is part of the state and should be constructed dynamically in the off-line problem.

Here, we handle V_h by introducing new variables r^{ω}_{kji} such that $r^{\omega}_{kji} = 1$ iff node i has been visited when node i is reached. The semantic is enforced via additional non-linear constraints in the off-line model. The latter is given by:

$$\min \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \sum_{k \in K} \sum_{i,j \in V} \xi^{\omega}_{i,j} x^{\omega}_{k,i,j} \qquad (\mathbf{P2.4})$$

$$\text{s.t. } Eq.(36) - (42)$$

$$(c_{ij} + \lambda^{\omega}_{k,i}) x^{\omega}_{k,i,j} (1 - r_{kji}) = 0 \quad \forall k \in K, \forall i, j \in V$$

$$(c_{ij} + \lambda^{\omega}_{k,i})(1 - r_{kji}) \geq 0 \quad \forall k \in K, \forall i, j \in V$$

$$r^{\omega}_{k,i,i} = y_{k,i} \quad \forall i \in V$$

$$r^{\omega}_{k,j,i} = r^{\omega}_{k,h,i} x^{\omega}_{k,h,j} \quad \forall i \in V, \forall h \in V, \forall j \in V$$

$$\underline{c}_{ij} \leq c_{ij} \leq \overline{c}_{ij} \quad \forall i, j \in V$$

The decision variables are y_{ki}, x^{ω}_{kij}, λ^{ω}_{ki}, r^{ω}_{kji}, plus the "virtual travel times" c_{ij}, i.e. the parameters for the on-line heuristic. Where the constraints on the r^{ω}_{kji} variables enforce the transitive property on the set of visited nodes. Bounding the virtual travel times is necessary to prevent the solver from building degenerate parameterizations for **P2.1** on purpose, which would trivially satisfy all KKT constraints and make the approach boil down to the baseline off-line solver.

4 Experiments

We performed an experimentation to compare the solution quality and run times of our methods. As references for comparison we use the baseline approaches, plus an optimal solver operating under perfect information. Additionally, on the VRP, BOON obtains the same solution quality as EXPECTATION, which gives a third term of comparison.

Experimental Setup: Our methods are evaluated over different uncertainty realizations, obtained by sampling the random variables for the loads and RES generation in the VPP model, and for the travel times in the VRP model. We consider a sample of 100 realizations for six different instances of each problem. We then run each approach on each realization and measure the cost and run time. The scenarios in our models, conversely, are not sampled, but programmatically chosen: for the VPP we consider four "extreme" scenarios where (resp.) the load and the RES generation are at low/high values. For the VRP, each scenario corresponds to the mean travel times in one mode of the distribution. The VPP problem has 24 on-line stages, while in the VRP the number depends on how many customers are assigned to each vehicle.

We solve our LPs and MILPs using Gurobi, while for the non-linear problems we use BARON via the GAMS modeling system on the Neos server for optimization. The time limit is 100 seconds for the VPP, and 500 seconds for the VRP. For the VPP, we use data from two public datasets to define problem instances for a residential (Espinosa and Ochoa 2015) and industrial plant[1]. For the VRP we use modified version of classical instances[2], by including problems from 10 to 30 customers with one depot and different numbers of vehicles.

[1]https://data.lab.fiware.org/dataset/

[2]http://myweb.uiowa.edu/bthoa/TSPTWBenchmarkDataSets.htm

Instance	Oracle (k€)	Baseline (k€)	BOON (k€)	MOON (k€)
I1	**331.36**	404.62 (+22.1%)	**342.06 (+3.2%)**	344.60 (+4.1%)
I2	**247.21**	311.14 (+25.9%)	265.32 (+7.3%)	**263.80 (+6.7%)**
I3	**393.81**	462.57 (+17.5%)	**404.32 (+2.7%)**	408.72 (+3.8%)
I4	**798.38**	923.24 (+15.6%)	819.24 (+2.6%)	**811.11 (+1.6%)**
I5	**565.60**	684.19 (+21.1%)	580.17 (+2.7%)	**573.93 (+1.5%)**
I6	**856.95**	984.90 (+14.9%)	874.58 (+2.1%)	**868.76 (+1.5%)**

Table 1: Cost values for the different VPP models

	Off-line part (sec)		On-line part (sec)	
Instance	Baseline	MOON	Heuristic	BOON
I1	0.184	27.884	0.778	5.011
I2	0.190	31.992	0.772	5.017
I3	0.185	30.772	0.775	5.009
I4	0.346	58.913	0.839	5.430
I5	0.341	59.184	0.832	5.423
I6	0.348	57.777	0.835	5.420

Table 2: Computation time for the different VPP model parts

Discussion for VPP: In Tables 1 and 2 we show the average costs and run time over the 100 input realizations for each approach for the VPP use case. On-line times refer to the sum of the stages. The baseline model (being an LP) appears to be rather efficient in terms of computation time, but yields solutions of limited quality. The BOON method comes much closer to the oracle solver, at the cost of a higher, but still reasonable, on-line run time. The MOON method incurs substantially larger off-line solution times, but it manages to beat or match the BOON solution quality by making use of the original, straightforward, on-line heuristic.

We show, for the VPP, the average values of each hourly optimized flow over the 100 realizations for each proposed model (i.e. Oracle, Baseline, BOON, MOON) in instance I2. We can see, in Fig. 2, the limits of using a non anticipatory algorithm, compared for example to Fig. 1 showing the Oracle optimization, since it is not possible to acquire energy from the grid in advance (i.e. when the cost is lower) and/or to sell energy to the grid in periods of highest price on the market or when more energy is available from renewable sources. Moreover the exchange of energy with the storage system is almost never used, i.e. to store RES energy. In Fig. 4, it is possible to see that, near the peak of renewable energy production, the MOON model accumulates energy in the storage and uses in a more balanced way the energy present in the storage system compared to the baseline model represented in Fig. 2. Furthermore, still looking at Fig. 3, it can be seen that BOON has peaks of energy sold on the network near the increase in electricity prices on the market. In Fig. 3 it is possible to notice the more consistent

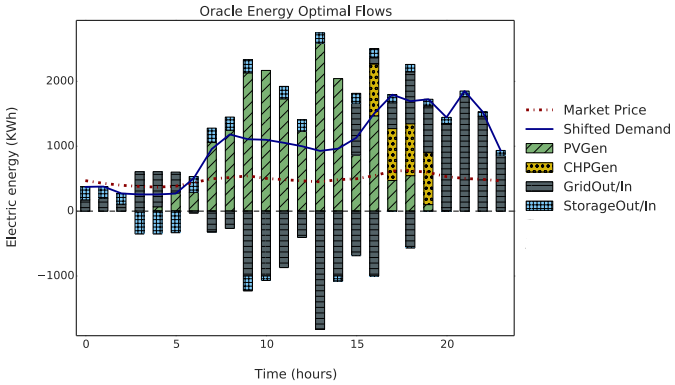


Figure 1: Oracle optimized energy flows

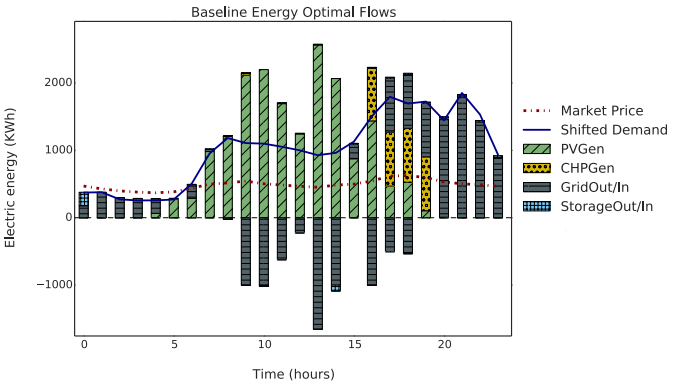


Figure 2: Baseline optimized energy flows

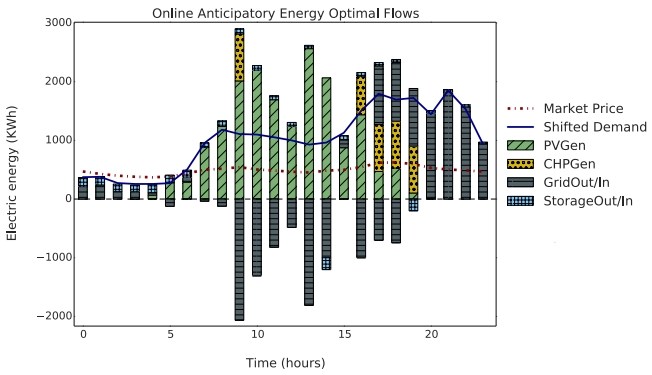


Figure 3: BOON optimized energy flows

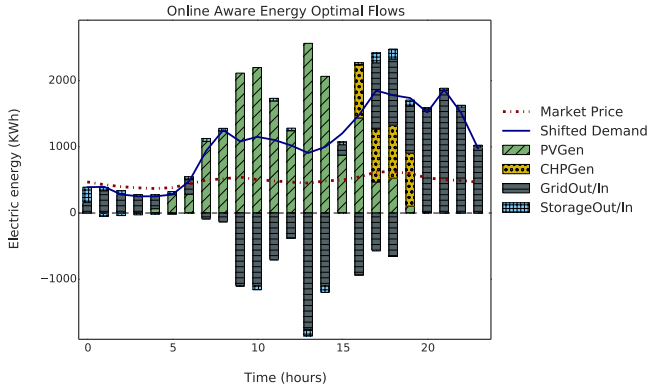


Figure 4: MOON optimized energy flows

use of the storage system. We can see that, by optimizing the virtual storage cost in the off-line stage, we can improve solution quality in term of cost (see Table 1) by using the storage system. Since the on-line solver has the ability to sell energy on the market, and storing energy has no profit, it ends up in always selling unless the virtual cost is employed.

Discussion for VRP: Tables 3 and 4 report the same results in terms of costs and computation time for the VRP. Here the on-line times are summed over all the vehicles. The original on-line heuristic is very efficient, but coupled with the baseline off-line model it does not come close to the oracle quality. The off-line model (a Mixed Integer Linear Program) takes also considerably more time to be solved. BOON, which in this case yields the same results as EXPECTATION with no time limit, yields substantially better solutions, but, being also MILP-based, it takes non-negligible time during the on-line phase. The MOON results follow the same trend as the VPP: the solution quality matches or beats that of BOON, at the cost of a higher off-line computation time, though the gap wrt the baseline is now much smaller.

From Fig. 5 to Fig. 8 we show the average on-line routing decisions over the 100 realizations for the same instance I2 (10 customers, one depot and two vehicles). The heatmaps shown below represent different colors for each vehicle and different color intensity for the number of times that each route has been chosen over the 100 realizations. We remember that our models make first off-line decisions (i.e. assignment of clients for each vehicle) and then make routing (on-line) decisions. We remember also that in MOON model we can have different off-line decisions (compared to those made from the other three models) since we inject KKT conditions in the off-line part. We therefore propose an instance as example where the off-line decisions are the same for all the models with the aim to observe the different on-line routing decisions. Indeed, we have different trends with the same off-line decisions. In particular, the Baseline model makes different routing decisions compared to the Oracle decisions: routes 2 -> 10, 3 -> 2, 6 -> 5 are never considered in the Baseline decisions while they are (with a certain probability) considered in the BOON decisions. We can also notice that, the Baseline and BOON models assume a (low) probability also for different routing decisions of vehicle 0 and this is not present in MOON routing decisions. The MOON routing decisions are equals to the Oracle ones for vehicle 0 in terms of probability and, for the other vehicle, they present routes (with the relative probability) never used by BOON (e.g. 0 -> 6, 5 -> 7). Moreover, we can notice that BOON presents on-line decisions with a higher probability but, in general, different from the most frequent decisions of the Oracle. Instead, MOON makes more decisions with lower probability than BOON, but considering more often decisions similar to the Oracle.

Instance	Oracle (t)	Baseline (t)	BOON (t)	MOON (t)
I1	**146.10**	165.83 (+13.5%)	151.23 (+3.5%)	**148.84 (+1.9%)**
I2	**278.37**	347.28 (+24.8%)	298.67 (+7.3%)	**290.43 (+4.3%)**
I3	**372.82**	561.66 (+50.7%)	**477.16 (+28.1%)**	507.80 (+36.2%)
I4	**321.57**	381.45 (+18.6%)	342.94 (+6.6%)	**340.85 (+6.1%)**
I5	**503.65**	670.86 (+33.2%)	559.22 (+11.1%)	**543.92 (+8.2%)**
I6	**448.53**	971.87 (+116.7%)	**470.99 (+5.1%)**	504.82 (+12.5%)

Table 3: Travel time (cost) values for the different VRP models

	Off-line part (sec)		On-line part (sec)	
Instance	Baseline	MOON	Heuristic	BOON
I1	1.699	6.255	0.255	7.134
I2	2.477	17.445	0.169	15.222
I3	2.532	25.938	0.554	18.024
I4	186.798	338.998	3.444	255.932
I5	243.330	357.543	5.248	313.656
I6	361.537	490.856	5.342	416.645

Table 4: Computation time for the different VRP model parts

5 Conclusion

This paper makes a first step toward generic integrated off-line/on-line optimization. We propose two alternative approaches, based on the idea of making the off-line and on-line solvers operate synergistically.

In the BOON method this is done by providing the on-line solver with the approximation of an oracle. In the MOON method, we instead make the off-line solver aware of the limitations of the on-line one, and capable of controlling its behavior by adjusting parameters. In general, our approaches work best for problems with numeric on-line decisions, but important classes of on-line heuristics are also covered (e.g. $\arg\min$ of parametric scores).

Both techniques yield substantially improved solutions: BOON matches the quality level of EXPECTATION, but it is applicable under more general assumptions. Unfortunately, the method is less efficient. MOON often manages to beat BOON (and therefore EXPECTATION) in terms of solution quality. While this comes at the price of a substantially increased off-line computation time, the method achieves these results by using naive and efficient on-line heuristics.

We believe there is room for improving the efficiency of our methods, for example by applying a method such as Logic Based Benders Decomposition to tackle each scenario in a separate subproblem (similarly to what done in the L-shaped method). As an alternative, we may attempt to generalize the technique used by REGRET to improve the efficiency of EXPECTATION. We also plan to test the sensitivity of our methods w.r.t. different choices for the on-line heuristic, and to apply our approaches to different problems, such as resource allocation and scheduling with Simple Temporal Networks under Uncertainty.

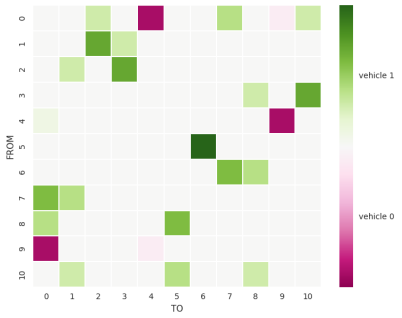


Figure 6: Baseline routing decisions

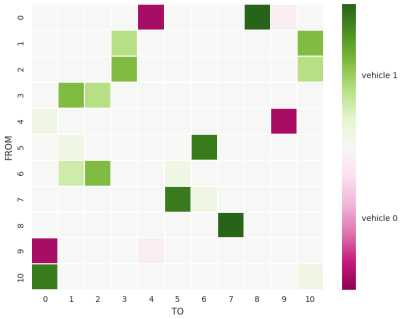


Figure 7: BOON routing decisions

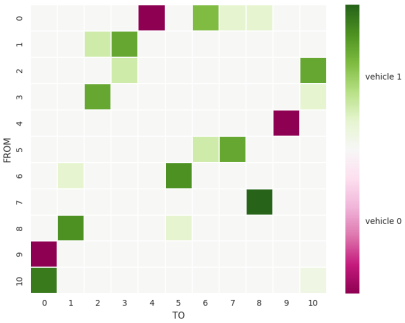


Figure 5: Oracle routing decisions

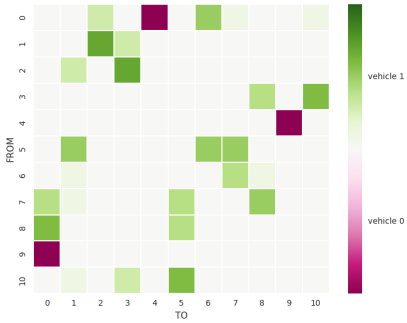


Figure 8: MOON routing decisions

References

Awasthi, P., and Sandholm, T. 2009. Online stochastic optimization in the large: Application to kidney exchange. In *IJCAI*, volume 9, 405–411.

Bai, H.; Miao, S.; Ran, X.; and Ye, C. 2015. Optimal dispatch strategy of a virtual power plant containing battery switch stations in a unified electricity market. *Energies* 8(3):2268–2289.

Bent, R., and Van Hentenryck, P. 2004a. Regrets only! online stochastic optimization under time constraints. In *AAAI*, volume 4, 501–506.

Bent, R. W., and Van Hentenryck, P. 2004b. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research* 52(6):977–987.

Bertsimas, D. J., and Simchi-Levi, D. 1996. A new generation of vehicle routing research: robust algorithms, addressing uncertainty. *Operations Research* 44(2):286–304.

Birge, J. R., and Louveaux, F. 1997. Introduction to stochastic programming. series in operations research and financial engineering.

Chang, H. S.; Givan, R.; and Chong, E. K. 2000. On-line scheduling via sampling. In *AIPS*, 62–71.

Clavier, J.; Bouffard, F.; Rimorov, D.; and Jos, G. 2015. Generation dispatch techniques for remote communities with flexible demand. *IEEE Transactions on Sustainable Energy* 6(3):720–728.

De Filippo, A.; Lombardi, M.; Milano, M.; and Borghetti, A. 2017. Robust optimization for virtual power plants. In *Conference of the Italian Association for Artificial Intelligence*, 17–30. Springer.

Espinosa, A. N., and Ochoa, L. N. 2015. Dissemination document "low voltage networks models and low carbon technology profiles". Technical report, University of Manchester.

Gamou, S.; Yokoyama, R.; and Ito, K. 2002. Optimal unit sizing of cogeneration systems in consideration of uncertain energy demands as continuous random variables. *Energy Conversion and Management* 43(9):1349 – 1361.

Hentenryck, P. V., and Bent, R. 2009. *Online stochastic combinatorial optimization*. The MIT Press.

Kall, P.; Wallace, S. W.; and Kall, P. 1994. *Stochastic programming*. Springer.

Laporte, G., and Louveaux, F. V. 1993. The integer l-shaped method for stochastic integer programs with complete recourse. *Operations research letters* 13(3):133–142.

Lee, C.; Lee, K.; and Park, S. 2012. Robust vehicle routing problem with deadlines and travel time/demand uncertainty. *Journal of the Operational Research Society* 63(9):1294–1306.

Mercier, L., and Van Hentenryck, P. 2008. Amsaa: A multistep anticipatory algorithm for online stochastic combinatorial optimization. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* 173–187.

Morales, J. M.; Conejo, A. J.; Madsen, H.; Pinson, P.; and Zugno, M. 2013. *Integrating renewables in electricity markets: operational problems*, volume 205. Springer Science & Business Media.

Palma-Behnke, R.; Benavides, C.; Aranda, E.; Llanos, J.; and Saez, D. 2011. Energy management system for a renewable based microgrid with a demand side management mechanism. In *2011 IEEE Symposium on Computational Intelligence Applications In Smart Grid (CIASG)*, 1–8.

Powell, W. B. 2016. *A Unified Framework for Optimization Under Uncertainty*. chapter 3, 45–83.

Shapiro, A., and Philpott, A. 2007. A tutorial on stochastic programming. *Manuscript. Available at www2. isye. gatech. edu/ashapiro/publications. html* 17.

Shapiro, A. 2013. Sample average approximation. In *Encyclopedia of Operations Research and Management Science*. Springer. 1350–1355.

Taş, D.; Dellaert, N.; Van Woensel, T.; and De Kok, T. 2013. Vehicle routing problem with stochastic travel times including soft time windows and service costs. *Computers & Operations Research* 40(1):214–224.

Toth, P., and Vigo, D. 2002. *The vehicle routing problem*. SIAM.

Winston, W. 2004. *Operations research: applications and algorithms*, volume 3.

Metric Nonlinear Hybrid Planning with Constraint Generation

Buser Say, Scott Sanner
Department of Mechanical & Industrial Engineering, University of Toronto, Canada
{bsay,ssanner}@mie.utoronto.ca

Abstract

We introduce a novel planner SCIPPlan for metric nonlinear hybrid planning in domains with general metric objectives, transcendental functions such as exponentials, and instantaneous continuous actions. Our key contribution is to leverage the spatial branch-and-bound solver of SCIP inside a nonlinear constraint generation framework where we iteratively check relaxed plans for temporal feasibility using a domain simulator, and repair the source of the infeasibility through a novel nonlinear constraint generation methodology. We experimentally show that SCIPPlan can plan effectively on a variety of domains, and outperforms ENHSP in terms of plan quality and runtime performance. We further show that SCIPPlan is competitive with a Tensorflow-based planner in highly nonlinear domains with exponential transition functions and a variety of metric objectives.

Introduction

Metric optimization is at the core of many real-world nonlinear hybrid planning domains where the *quality* of the plan matters. Most nonlinear hybrid planners in the literature either ignore metric specifications (Penna *et al.* 2009; Bryce *et al.* 2015; Cashmore *et al.* 2016), or leverage heuristics to guide their search for finding a plan quickly (Löhr *et al.* 2012; Piotrowski *et al.* 2016) with the notable exceptions COLIN (Coles *et al.* 2012) and ENHSP (Scala *et al.* 2016), which can handle metric optimization for a subset of PDDL+ (Fox and Long 2006) domains.

In this paper, we leverage the SCIP nonlinear constrained optimization solver SCIP (Maher *et al.* 2017) to present SCIPPlan for solving metric nonlinear hybrid planning problems by decomposing the original problem into a master problem and subproblem. In the master problem, we relax the original problem to a system of sequential function updates, which allows us to handle arbitrary nonlinear functions (such as polynomial, transcendental, logarithmic, trigonometric etc.) with metric objectives and instantaneous continuous action inputs that are beyond the expressivty of existing hybrid planners.

In the nonlinear hybrid planning literature, the time at which a conditional expression (e.g., a mode switch condition) is satisfied is known as a *zero-crossing* (Shin and Davis 2005), and the candidate solution found by solving the master problem can include zero-crossings between two consecutive decisions which can violate the global constraints of the original problem. When the dynamics of the planning problem are piecewise linear, one can use the TM-LPSAT compilation to find valid plans. Similarly, when the continuous change of the planning problem can be described as polynomials, one can use the SMTPlan (Cashmore *et al.* 2016) compilation of the hybrid planning problem to avoid zero-crossings between two consecutive decision points (i.e. happenings). However in general, problem dynamics can include arbitrary nonlinear change which can violate these assumptions. To identify and repair global constraint violations that are due to zero-crossings, we use simulate-and-validate approach (Fox *et al.* 2014) in the subproblem where domain simulators are used to simulate the candidate plan, and if the candidate plan is found to be infeasible, temporal constraints associated with zero-crossings are added back to the master problem. SCIPPlan iteratively solves the master problem and the subproblem until a valid plan is found.

Experimentally, we show that SCIPPlan outperforms the state-of-the-art metric nonlinear hybrid planner ENHSP in almost all problem instances with respect to solution quality and run-time performance. We further experiment with the capabilities of SCIPPlan beyond the expressiveness limitations of ENHSP in the optimization of general metrics on a subset of modified domains and test its scalability against a Tensorflow-based planner (Wu *et al.* 2017) on a nonlinear RDDL domain with exponential functions from the literature.

Problem Definition

A factored metric nonlinear hybrid planning problem is a tuple $\Pi = \langle F, A, C, T, M, I, G \rangle$ where $F = \{F^d, F^r\}$ is a mixed set of fluent variables (fluents) with discrete $F^d \in \mathbb{Z}$ and real $F^r \in \mathbb{R}$ domains, $A = \{A^d, A^r\}$ is a mixed set of action variables (actions) with discrete $A^d \in \mathbb{Z}$ and real $A^r \in \mathbb{R}$ domains, global constraint $C(f, a, s) \rightarrow \{true, false\}$ is a function that returns true if value assignments to actions $a \in A$ and fluents $f \in F$ at time $s \in \mathbb{R}_{\geq 0}$ satisfies C, T denotes the transition of fluent $f \in F$ to $f' \in F$ after duration $\Delta s \in \mathbb{R}_{\geq 0}$ from time s such that $T(f, a, s, \Delta s) = f'$ if there does not exist $\Delta s', \quad 0 \leq \Delta s' \leq \Delta s$ and $C(T(f, a, s, \Delta s'), a, s + \Delta s') = false$ where $\Delta s \in \mathbb{R}_{\geq 0}$ denotes the duration of the transition, $I \subseteq C$ and $G \subseteq C$

are the initial and goal constraints that specify the values of all fluents and subset of fluents at times $s = 0$ and $s = \sum_{t \in \{1,\ldots,H\}} \Delta s_t$ respectively where horizon H is the bound on the number of sequential actions allowed to be executed (decisions), and M denotes the metric of interest that must be optimized. For a given horizon H, a solution (plan) to Π is a value assignment to all action variables $a \in A$ from their domains such that $T(\ldots T(T(f_0, a_0, s_0, \Delta s_0), a_1, s_1, \Delta s_1) \ldots, a_H, s_H, \Delta s_H)$ satisfies all constraints $c \in C$ and optimizes metric M.

Note that our definition Π extends RDDL formalism (Sanner 2010) with the addition of goal constraints G and the temporal validity condition that checks whether a transition $T(f, a, s, \Delta s)$ between time interval $[s, s + \Delta s]$ violates a global constraint $c \in C$. Unlike the PDDL+ (Fox and Long 2006) formalism, we do not assume that the effects of instantaneous actions are realized ϵ time after their execution. The formalism of RDDL allows us to model the instantaneous effects of actions by the use of layers and intermediate fluents, and allow instantaneous continuous actions $A^r \subseteq A$ that are not functions of time.

Spatial Branch-and-Bound

Spatial Branch-and-Bound (SBB) (Mitten 1970) is an algorithm based on the *divide-and-conquer* strategy for solving Mixed-Integer Nonlinear Programming (MINLP) that is in the form of $\min f(\mathbf{x})$ subject to $\mathbf{g(x)} \leq 0$ where function $f(x)$ and function vector **g(x)** contain nonlinear functions, and the decision variable vector **x** can have continuous and/or discrete domains. The SBB algorithm utilizes tree search where the branching decisions are made on candidate solutions $\bar{\mathbf{x}}$, and the optimal value of the objective function $f(\bar{x})$ is bounded at each node of the search tree until some optimality gap parameter g is reached.

Domain Simulators

Domain simulators, such as VAL (Howey *et al.* 2004) or RDDLsim (Sanner 2010), can be used to validate a candidate solution $\pi = \langle a_1, \Delta s_1 \ldots, a_H, \Delta s_H \rangle$ to the planning problem Π by checking whether $T(\ldots T(T(f_0, a_0, s_0, \Delta s_0), a_1, s_1, \Delta s_1) \ldots, a_H, s_H, \Delta s_H)$ satisfies constraints $c \in C$ by discretizing the duration of transitions Δs_t for all decisions $t \in \{1, \ldots, H\}$.

Constraint Generation for Metric Nonlinear Hybrid Planning

In this section, we introduce our novel SCIP-based planner (SCIPPlan) to plan in metric nonlinear hybrid planning domains with general metric considerations as outlined by Algorithm 1. The novelty of our work is that we decompose Π into a master problem $\mathcal{M}(\Pi, H)$ and a subproblem $\mathcal{S}(\pi)$, where $\mathcal{M}(\Pi, H)$ solves a relaxation of Π for a bounded number of decisions H, and $\mathcal{S}(\pi)$ checks whether π is valid for Π. If π is found invalid, we return the first time intervals $[s, s + \Delta s]$ that violate C, and construct an informative temporal constraint to be added back to the master problem.

Master problem

The master problem $\mathcal{M}(\Pi, H)$ modifies Π by first assuming a bound on the number of decisions $t \in \{1, \ldots, H\}$ and relaxing the constraint $\nexists \Delta s', \quad 0 \leq \Delta s' \leq \Delta s \wedge C(T(f, a, s, \Delta s'), a, s + \Delta s') = \mathit{false}$ that checks for zero-crossing between two consecutive decisions t and $t+1$. This allows us to construct the master problem as a MINLP using the same methodology described in (Raghavan *et al.* 2017) for compiling the RDDL formalism into a Mixed-Integer Linear Program (MILP) for piecewise linear domains, except that now we simply allow fully nonlinear domains and we also add goal constraints as $G(f, ., \sum_{t \in \{1,\ldots,H\}} \Delta s_t)$.

Subproblem

Given a candidate plan π, the subproblem $\mathcal{S}(\pi)$ checks every pair of consecutive decisions $t, t+1 \in \{1, \ldots, H-1\}$ for $T(\bar{f}_t, \bar{a}_t, \sum_{t' \in \{1,\ldots,t-1\}} \bar{\Delta s}_{t'}, \bar{\Delta s}_t) \neq \bar{f}_{t+1}$ where parameters $\bar{f}_t$, $\bar{a}_t$ and $\bar{\Delta s}_t$ denote the optimal values of the variables f_t, a_t and Δs_t from the master problem. If such a discrepancy is found between decisions t and $t+1$, it must be true that there exists an interval $[s_t^1, s_t^2]$ such that some conditional expression (i.e., if-else-then condition) or a global constraint that is not satisfied at times s_t or s_{t+1}, is satisfied for the duration of the interval $[s_t^1, s_t^2]$ where $s_t < s_t^1 \leq s_t^2 < s_{t+1}$. We use our own domain simulator to simulate π, and either validate π or return $[s_t^1, s_t^2]$ and the violated constraint $g_t \in C$ for decision t.

Temporal Constraint Generation

Given the interval $[s_t^1, s_t^2]$ identified by the domain simulator for decision $t \in \{1, \ldots, H-1\}$ and the respective violated constraint $g_t(f_t, a_t, s_t \in [s_t^1, s_t^2]) \leq 0$, we generate a nonlinear constraint of the form:

$$g_t(f_t, a_t, k s_t) \leq 0 \tag{1}$$

$$k = [1, \ldots, \frac{s_t^2 + s_t^1 - 2 \sum_{t' \in \{1,\ldots,t-1\}} \bar{\Delta s}_{t'}}{2 \bar{\Delta s}_t}, \ldots, 1] \tag{2}$$

to be added to $\mathcal{M}(\Pi, H)$ with k denoting the coefficient vector for the duration variables Δs_t for all decisions $t \in \{1, \ldots, H\}$ where all the elements of the vector are equal to 1 except the t-th entry which has a value between 0 and 1. Note that unlike SMTPlan, the function g_t is not restricted to polynomials.

The benefits of our constraint generation methodology are threefold. The first benefit is that we preserve the functional relationship between duration of the control input Δs_t and the violated constraint g_t, which removes symmetrical zero-crossings in the form of $g_t(f_t, a_t, s_t) = g_t(f_t', a_t', s_t)$, where $f_t \neq f_t'$ and $a_t \neq a_t'$. The second benefit is that SCIPPlan only generates the temporal constraints as they are needed. This behavior is desired as not all zero-crossings lead to invalid plans. Similar to the zero-crossings constraints of SMTPlan, Constraint (1) is also a sufficient condition to validate π for Π. The addition of unnecessary instances of Constraint (1) can potentially eliminate candidate plans π from the solution space of $\mathcal{M}(\Pi, H)$ that are in fact valid for Π. The last benefit is computational. Since Constraint (1)

only perturbs function g_t by changing its coefficients and not adding any additional decision variables, it allows SCIPPlan to preserve some important information between iterations (e.g., warm start features).

Algorithm 1 SCIPPlan

$H \leftarrow 1, \pi \leftarrow \emptyset, s_t^1, s_t^2, t \leftarrow \emptyset$
while π is $\emptyset$ **do**
  $\pi \leftarrow \mathcal{M}(\Pi, H)$
  if π is $\emptyset$ **then**
    $H \leftarrow H + 1$.
  else $[s_t^1, s_t^2, t] \leftarrow \mathcal{S}(\pi)$
    if s_t^1, s_t^2, t are $\emptyset$ **then**
      return π
    else $\mathcal{M}(\Pi, H) \leftarrow Constraint(1)$

Illustrative Example

To illustrate how our metric nonlinear hybrid planner works for one iteration, let us consider the following simple navigation domain with two continuous action variables $A^r = \{a_x, a_y\}$ for moving the agent in x and y directions, two continuous fluent variables $F^r = \{f_x, f_y\}$ to represent the location of the agent, with constraints $c_1, c_2, c_3 \in C$ on s_1, s_2, a_1, a_2 such that:

$$c_1 : 0 \leq f_{x,t}, f_{y,t} \leq 10 \quad \forall t \in \{1, \ldots, H+1\}$$
$$c_2 : -1 \leq a_{x,t}, a_{y,t} \leq 1 \quad \forall t \in \{1, \ldots, H\}$$
$$c_3 : 4 \geq f_{x,t} \vee 6 \leq f_{x,t} \vee 4 \geq f_{y,t} \vee 6 \leq f_{y,t} \quad \forall t \in \{1, \ldots, H\}$$

over horizon $H = 2$ where constraints c_1, c_2 denote bounds on the domains of fluents $f_x, f_y \in F^r$ and constraints c_3 represents an obstacle located in the middle of the maze, and initial and goal constraints:

$$I : f_{x,1}, f_{y,1} = 0$$
$$G : f_{x,2}, f_{y,2} = 8\ ,$$

respectively. Given the transition function:

$$T : f_{x,t+1} = f_{x,t} + a_{x,t}\Delta s_t \quad \forall t \in \{1, \ldots, H\}$$
$$T : f_{y,t+1} = f_{y,t} + a_{y,t}\Delta s_t \quad \forall t \in \{1, \ldots, H\}$$

and the metric of interest $\min \sum_t \Delta s_t$, the master problem $\mathcal{M}(\Pi, H)$ can return a candidate plan $\pi = [a_{x,1} = 1, a_{y,1} = 1, \Delta s_1 = 8, a_{x,2} = 0, a_{y,2} = 0, \Delta s_2 = 0]$ as visualized by Figure 1. The subproblem $\mathcal{S}(\pi)$ will detect the interval of zero-crossing by simulating the transition function T for all decisions $t \in \{1, \ldots, H\}$ and detect the first violation of constraint c_3 which occurs within the interval $[s_1^1 = 4, s_1^2 = 6]$. Given the identified zero-crossing interval $[s_1^1 = 4, s_1^2 = 6]$ and the violated constraint c_3, the following constraint:

$$\begin{aligned} g_1^1 : 4 &\geq f_{x,1} + (0.625)a_{x,1}\Delta s_1 \\ &\vee 6 \leq f_{x,1} + (0.625)a_{x,1}\Delta s_1 \\ &\vee 4 \geq f_{x,1} + (0.625)a_{x,1}\Delta s_1 \\ &\vee 6 \leq f_{x,1} + (0.625)a_{x,1}\Delta s_1 \\ g_1^2 : 4 &\geq f_{y,1} + (0.625)a_{y,1}\Delta s_1 \\ &\vee 6 \leq f_{x,1} + (0.625)a_{x,1}\Delta s_1 \\ &\vee 4 \geq f_{y,1} + (0.625)a_{y,1}\Delta s_1 \\ &\vee 6 \leq f_{y,1} + (0.625)a_{y,1}\Delta s_1 \end{aligned}$$

will be added to the master problem. The master problem would then be re-solved and further constraints generated if needed. Once no zero-crossings were detected in a solution, that plan would be returned as the final feasible plan.

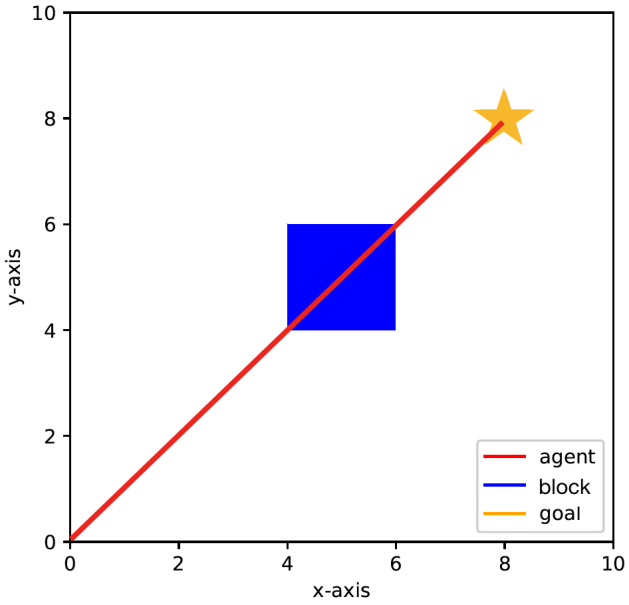


Figure 1: Visualization of the candidate plan generated by the master problem of SCIPPlan for the illustrative hybrid navigation domain. The candidate plan violates the constraints of the problem for the interval [4,6], which corresponds to the zero-crossings.

Next, we describe the experimental setup under which we test SCIPPlan over complex metric nonlinear hybrid domains.

Experimental Results

In this section, we present four metric nonlinear hybrid domains to test the computational efficacy of SCIPPlan, namely: `HVAC` (Agarwal *et al.* 2010), `ComplexPouring` (Scala *et al.* 2016), (Bryce *et al.* 2015), `NavigationJail` and `NavigationMud` (Say *et al.* 2017), against ENHSP (Scala *et al.* 2016) and a Tensorflow-based planner (Wu *et al.* 2017) with respect to computation time and solution quality.

Domain Descriptions

In this section, we describe the benchmark domains in detail. The domains were chosen to test the capabilities of SCIPPlan on metric optimization, handling nonlinear transitions and concurrency.

Heating, Ventilation and Air Conditioning is the problem of heating different rooms $r \in R$ of a building upto a desired temperature by sending heated air b_r. The temperature of a room $h_{r,s+1}$ is bilinear function of its current temperature $h_{r,s}$, the volume of heated air sent to the room b_r, the temperature of the adjacent rooms $h_{r',t}$ and the duration Δt_s of the control input at time point s where $r' \in Adj(r) \subset R$ denotes the set of adjacent rooms to room r. The dynamics of the domain are described as follows:

$$h_{r,s+1} = h_{r,s} + \frac{\Delta t_s}{C_r}(b_r + \sum_{r' \in Adj(r)} \frac{h_{r',s} - h_{r,s}}{W_{r,r'}}) \quad (3)$$

for all $r \in R, s \in \{1, \ldots, H\}$ where C_r and $W_{r,r'}$ are parameters denoting the heat capacity of room r and the heat resistance of the wall between r and r', respectively. Moreover, the initial and the goal constraints are described as $h_{r,1} = H_r^{init}$ and $h_{r,H} = H_r^{goal}$ for all rooms $r \in R$ where the parameters H_r^{init} and H_r^{goal} denote the initial and goal temperatures of the rooms, respectively.

ComplexPouring is the problem of filling buckets $b \in B$ upto a desired volume with the water that is initially stored in the tanks $t \in T$. The volume of a bucket $v_{b,s+1}$ (or a tank) is a nonlinear function of its current volume $v_{b,s}$ (or $v_{t,s}$), volume of water poured in (and out) from (and to) other tanks and Δt_s at time point s. The dynamics of the domain are described as follows:

$$v_{b,s+1} = v_{b,s} + v_{b,s}^+ - v_{b,s}^- \qquad \forall b \in B \cup T \quad (4)$$

$$v_{b,s}^+ = \sum_{t \in T} \Delta t_s p_{t,b,s} \; (2R_t\sqrt{v_{t,s}} - R_t^2) \qquad \forall b \in B \cup T \quad (5)$$

$$v_{b,s}^- = \sum_{t \in B \cup T} \Delta t_s p_{b,t,s} \; (2R_b\sqrt{v_{b,s}} - R_b^2) \qquad \forall b \in T \quad (6)$$

$$0 \leq v_{b,s} \leq V_{b,s}^{max} \qquad \forall b \in B \cup T \quad (7)$$

for all $s \in \{1, \ldots, H\}$ where $p_{b,t,s} \in \{0, 1\}$ is a binary decision variable denoting whether tank b pours into bucket (or tank) t at time point s, and R_b and $V_{b,s}^{max}$ are parameters denoting the flow rate and capacity of bucket (or tank) b, respectively. Further, the initial and the goal constraints are described as $v_{b,1} = V_b^{init}$ for all buckets and tanks $b \in B \cup T$ and $v_{b,H} \geq V_b^{goal}$ for all buckets $b \in B$ where the parameters V_b^{init} and V_b^{goal} denote the initial and goal volumes of tanks and buckets, respectively.

NavigationJail is a two-dimensional $d \in \{x, y\} = D$ path-finding domain that is designed to test the ability of planners to handle instantaneous events. The location of the agent $l_{d,s+1}$ is a nonlinear function (i.e., cubic polynomial) of its current location $l_{d,s}$, speed $v_{d,s}$, acceleration $a_{d,s}$ and Δt_s at time point s. Moreover, the agent can be instantaneously relocated to its initial position L_d^{init} for all dimensions $d \in D$ and set its speed to 0 if it travels through a two-dimensional jail area that is located in the middle of the maze with the corner points J_d^{min}, J_d^{max} for all $d \in D$. The system dynamics of the domain is described as follows:

$$l'_{d,s+1} = l_{d,s} + v_{d,s}\Delta t_s + 0.5 a_{d,s}\Delta t_s^2 \qquad \forall d \in D \quad (8)$$

$$v'_{d,s+1} = v_{d,s} + a_{d,s}\Delta t_s \qquad \forall d \in D \quad (9)$$

$$\textbf{if} \quad \forall d \in D \quad J_d^{min} \leq l'_{d,s+1} \leq J_d^{max} \quad (10)$$

$$\textbf{then} \quad l_{d,s+1} = L_d^{init},\ v_{d,s+1} = 0 \qquad \forall d \in D \quad (11)$$

$$\textbf{else} \quad l_{d,s+1} = l'_{d,s+1},\ v_{d,s+1} = v'_{d,s+1} \qquad \forall d \in D \quad (12)$$

$$L_d^{min} \leq l_{d,s} \leq L_d^{max} \qquad \forall d \in D \quad (13)$$

$$A_d^{min} \leq a_{d,s} \leq A_d^{max} \qquad \forall d \in D \quad (14)$$

for all $s \in \{1, \ldots, H\}$ where (L_d^{min},L_d^{max}) and (A_d^{min},A_d^{max}) are the minimum and the maximum boundaries of the maze and the control input for dimension $d \in D$, respectively. The goal of the domain is to find a path from the initial location L_d^{init} to the goal location L_d^{goal} for all dimensions $d \in D$. The initial and the goal constraints are described as $l_{d,1} = L_d^{init}$, $v_{d,1} = 0$, and $l_{d,H} = L_d^{goal}$ for all dimensions $d \in D$, respectively.

In all three domains, the total makespan $\sum_{s \in \{1,\ldots,H\}} \Delta t_s$ is minimized where Δt_s is always non-negative such that $0 \leq \Delta t_s \quad \forall s \in \{1, \ldots, H\}$. While SCIPPlan supports general metric optimization, the `aibr` heuristic of ENHSP does not support general metric optimization with processes and events. Therefore we picked a metric that was captured by the heuristic calculation of ENHSP.

NavigationMud is a two-dimensional $d \in \{x, y\} = D$ RDDL domain that is designed to test the ability of planners to handle transcendental functions with general optimization metrics. The location of the agent $l_{d,s+1}$ is a nonlinear function (i.e., exponential) of its current location $l_{d,s}$ and positional displacement action $p_{d,s}$ at time point s due to higher slippage in the center of the maze. The system dynamics of the domain is described as follows:

$$l_{d,0} = L_d^{init} \qquad \forall d \in D \quad (15)$$

$$l_{d,s+1} = l_{d,s} - 0.99 + p_{d,s}\frac{2.0}{1.0 + e^{-2y_s}} \qquad \forall d \in D \quad (16)$$

$$y_s = \sqrt{\sum_{d \in D} (l_{d,s} - \frac{L_d^{max} - L_d^{min}}{2.0})^2} \quad (17)$$

$$L_d^{min} \leq l_{d,s} \leq L_d^{max} \qquad \forall d \in D \quad (18)$$

$$P_d^{min} \leq p_{d,s} \leq P_d^{max} \qquad \forall d \in D \quad (19)$$

for all $s \in \{1, \ldots, H\}$ where (P_d^{min},P_d^{max}) are the minimum and the maximum boundaries of the positional displacement for dimension $d \in D$, respectively.

The objective of the domain is to find a path from the initial location L_d^{init} with the minimum total Manhattan dis-

tance $\sum_{s\in\{1,\ldots,H\}}\sum_{d\in D}|l_{d,s}-L_d^{goal}|$ from the goal location L_d^{goal} over all time points s.

Computational Performance

In this section, we investigate the efficiency of using a SCIPPlan for solving metric hybrid planning problems. We ran the experiments on MacBookPro with 2.8 GHz Intel Core i7 16GB memory. We optimized the nonlinear encodings using SCIP 4.0.0 (Maher *et al.* 2017) with 1 thread, and 30 minutes total time limit per domain instance.

Modeling differences between SCIPPlan versus PDDL+

In SCIPPlan, we modeled the control inputs $b_{r,s}$ and $a_{d,s}$ from `HVAC` and `NavigationJail` domains as decision variables with continuous domains. In PDDL+, we incremented and decremented the control input as a function of time with some constant rate z. Further in the `NavigationJail` domain, we have modeled constraints (11-13) using events in PDDL+ (as opposed to using global constraints) since going into the jail location is not a dead-end. In the `HVAC` and `NavigationJail` domains, we tested ENHSP with relaxed goal settings where the respective equality goal constraints were relaxed to the following constraints:

$$H_r^{goal}-z\leq h_{r,H}\leq H_r^{goal}+z \qquad \forall r\in R \qquad (20)$$

$$L_d^{goal}-z\leq l_{d,H}\leq L_d^{goal}+z \qquad \forall d\in D \qquad (21)$$

due to the continuous domains of fluents f and actions a (i.e., $a, f \in \mathbb{R}$), and the fact that ENHSP identified these domains to be infeasible with equality constraints. We tested SCIPPlan under different optimality gap parameters g. For both parameter settings z and g, we will use the notation $\mathbf{SP^x}$ to report results for SCIPPlan under the optimality gap setting $g = x$, and $\mathbf{EP^x}$ for ENHSP under the rate setting $z = x$.

Comparison of the runtime performance and solution quality

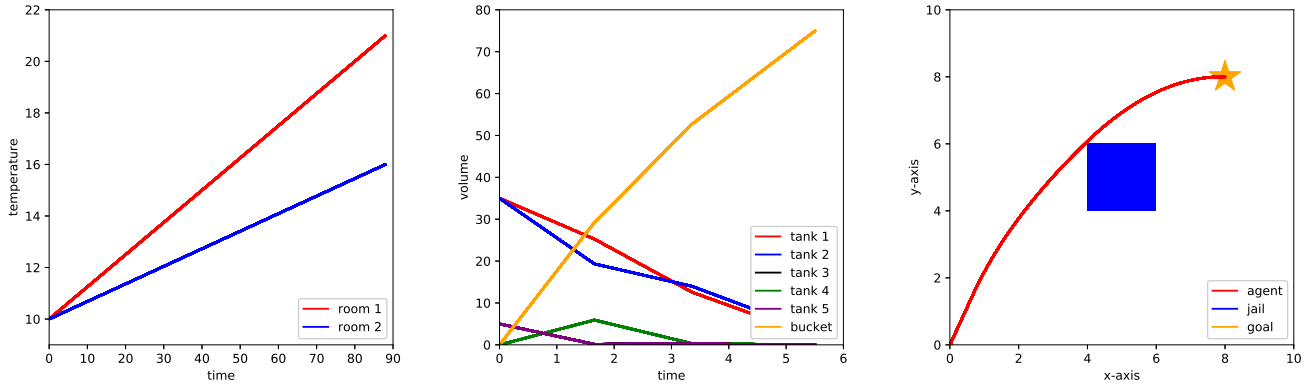
In Table 1, we compare the run times and the quality of plans produced by SCIPPlan, ENHSP and Tensorflow-based planner with respect to the chosen optimization metric under the best performing parameter settings. From left to right, the first column of Table 1 specifies the domains and problem instances solved. The second and third columns present the optimal makespan found by the respective planners. The fourth and firth columns present the computational effort that is required to produce the metrics presented in the second and third columns. The sixth column presents the running time (seconds) that is required to optimize the general metric variants of the domains in cases where a simpler metric was required to meet expressivity requirements for the ENHSP comparison.

Comparison of runtime performances The detailed inspection of the columns associated with solution quality shows that SCIPPlan can successfully find high quality plans in almost all the instances with optimality gap parameter $g \leq 0.05$, except the largest domain `NavigationMud` (-0.2,0.2). In contrast, we observe that in `HVAC` and `ComplexPouring` domains, ENHSP can find plans with on average 60% lower quality compared to SCIPPlan. Moreover in `NavigationJail` domain, neither $EP^{0.1}$ nor $EP^{0.01}$ found feasible plans within time limit. In `NavigationMud` domain, we tested the scalability of SCIPPlan against the Tensorflow-based planner. We found that SCIPPlan is competitive with the Tensorflow-based planner with respect to the solution quality of the plans found in the first two instances, whereas the instance `NavigationMud(-0.2,0.2)` is hard to optimize (i.e., the plan does not contain actions other than no-ops) for SCIPPlan. We note that unlike the Tensorflow-based planner, we currently do not leverage parallel computing, which is one of the main strengths of Tensorflow to handle large scale optimization problems. In Figure 2, we visualize the plan traces to get a better understanding of what makes a domain hard in terms of plan computability. The inspection of plan traces shows from left to right: linear, piecewise linear and nonlinear state transitions. Together with the computational results presented in Table 1, we confirm that domains with nonlinear state transitions (e.g., NavigationJail) are significantly harder to compute compared to linear (e.g., HVAC) and piecewise linear (e.g., ComplexPouring) domains.

Table 1: Comparison of the run times and the quality of plans produced by SCIPPlan, ENHSP and Tensorflow-based planner with respect to the given domain metrics.

	Makespan				General
Domain	Quality		Run Time		Run Time
HVAC	SP^0	$EP^{0.1}$	SP^0	$EP^{0.1}$	SP^0
(2,R)	**88.00**	145.00	$\leq$ **0.01**	1.02	0.02
(2,D)	**88.00**	145.00	$\leq$ **0.01**	1.02	0.19
Pouring	SP^0	EP	SP^0	EP	SP^0
(3,1)	**4.30**	11.00	0.10	0.32	**0.01**
(5,1)	**5.51**	19.00	1.38	**0.41**	0.87
(4,2)	**7.67**	22.00	0.93	**0.37**	0.58
(9,2)	**1.69**	10.00	0.90	0.37	**0.08**
NJail	$SP^{0.05}$	$EP^{0.1}$	$SP^{0.05}$	$EP^{0.1}$	-
(-1.0,1.0)	**13.59**	-	**281.75**	$\geq$ 1800	-
(-0.5,0.5)	**13.63**	-	**60.94**	$\geq$ 1800	-
(-0.2,0.2)	**13.35**	-	**59.29**	$\geq$ 1800	-
NMud	$SP^{0.05}$	TF	$SP^{0.05}$	TF	-
(-1.0,1.0)	**64.25**	65.23	**15.46**	30.00	-
(-0.5,0.5)	140.35	**136.55**	232.84	240.00	-
(-0.2,0.2)	800.00	**360.38**	1800.00	**960.00**	-

Comparison of solution qualities The inspection of the last two columns show that SCIPPlan finds high quality plans with little computational effort in `HVAC` and `ComplexPouring` domains, whereas it takes on average 135 seconds and 125 seconds to find high quality plans for `NavigationJail` and `NavigationMud` domains, with the exception of the largest instance `NavigationMud` (-0.2,0.2,50). The benefit of spending computational resources to provide stronger optimality guarantees is justi-



(a) HVAC instance with 2 rooms and actions with continuous domains.

(b) ComplexPouring instance with 5 tanks and 1 bucket.

(c) NavigationJail domain with acceleration bounded within the interval [-0.5,0.5].

Figure 2: Visualization of example plans generated by SCIPPlan. The inspection of plan traces show from left to right: linear, piecewise linear and nonlinear fluent transitions as a function of time. The visualization of the plan traces confirm the results presented in Table 1; domains with nonlinear state transitions are significantly harder to compute compared to linear and piecewise linear fluent transitions.

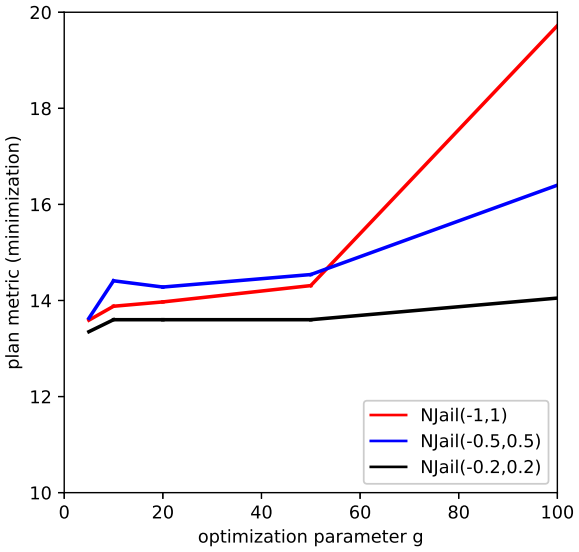


Figure 3: The increase in plan quality (lower is better for minimization) as a function of optimality gap parameter g for the SCIP-based planner on NavigationJail domain.

fied in Figure 3, where we plot the increase in plan quality as a function of optimality gap parameter g. Figure 3 shows that spending more computational resources can significantly improve the quality of the plan found as the instances get harder to solve.

General metric specifications

We test SCIPPlan on general metrics of interest in `HVAC` and `ComplexPouring` domains and measure its effect on run time performance. In the `HVAC` domain, we modify the metric to minimize the total cost $\sum_{s\in\{1,\ldots,H\}}\sum_{r\in R} c_r b_{r,s}$ of heating all the rooms $r \in R$ of a building across all time points where the parameter c_r denotes the unit cost of heating room $r \in R$. Similarly in `ComplexPouring` domain, we minimize the total number of times we pour from one tank to the bucket (or other tanks) across all time points such that $\sum_{s\in\{1,\ldots,H\}}\sum_{t\in T}\sum_{b\in B\cup T} p_{t,b,s}$.

The results presented in the last column of Table 1 show that the performance of SCIPPlan is on average the same for general metric optimization and makespan optimization. As demonstrated in `NavigationMud` and modified `HVAC` and `ComplexPouring` domains, SCIPPlan efficiently finds high quality plans with respect to the metric.

Conclusion

In this paper, we presented a novel SCIP-based planner (SCIPPlan) that can plan in metric nonlinear hybrid planning domains with metric specifications, transcendental functions such as exponentials and instantaneous continuous actions. In SCIPPlan, we leveraged the spatial branch-and-bound solver of SCIP inside a nonlinear constraint generation framework where candidate plans are iteratively checked for temporal infeasibility using a domain simulator, and the sources of infeasibilities are repaired through a novel nonlinear constraint generation algorithm. Experimentally, we have shown that SCIPPlan can plan effectively on a variety of domains and outperformed ENHSP in terms of plan quality and runtime performance. We have further shown that SCIPPlan is competitive with a Tensorflow-based planner in highly nonlinear domains with exponential state transition functions and general metric specifications.

References

Yuvraj Agarwal, Bharathan Balaji, Rajesh Gupta, Jacob Lyles, Michael Wei, and Thomas Weng. Occupancy-driven energy management for smart building automation. In *ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, pages 1–6, 2010.

Daniel Bryce, Sicun Gao, David Musliner, and Robert Goldman. SMT-based nonlinear PDDL+ planning. In *29th AAAI*, pages 3247–3253, 2015.

Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. A compilation of the full PDDL+ language into SMT. In *ICAPS*, pages 79–87, 2016.

Amanda J. Coles, Andrew I. Coles, Maria Fox, and Derek Long. COLIN: Planning with continuous linear numeric change. *JAIR*, pages 1–96, 2012.

Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *JAIR*, 27(1):235–297, 2006.

Maria Fox, Derek Long, and Daniele Magazzeni. Plan-based policies for efficient multiple battery load management. *CoRR*, abs/1401.5859, 2014.

Richard Howey, Derek Long, and Maria Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301, 2004.

Johannes Löhr, Patrick Eyerich, Thomas Keller, and Bernhard Nebel. A planning based framework for controlling hybrid systems. In *ICAPS*, pages 164–171, 2012.

Stephen J. Maher, Tobias Fischer, Tristan Gally, Gerald Gamrath, Ambros Gleixner, Robert Lion Gottwald, Gregor Hendel, Thorsten Koch, Marco E. Lübbecke, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The scip optimization suite 4.0. Technical Report 17-12, ZIB, Takustr.7, 14195 Berlin, 2017.

L. G. Mitten Mitten. Branch-and-bound methods: General formulation and properties. *Operations Research*, 18(1):24–34, 1970.

Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. UPMurphi: A tool for universal planning on PDDL+ problems. In *ICAPS*, pages 106–113, 2009.

Wiktor Mateusz Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. Heuristic planning for hybrid systems. In *AAAI*, pages 4254–4255, 2016.

Aswin Raghavan, Scott Sanner, Prasad Tadepalli, Alan Fern, and Roni Khardon. Hindsight optimization for hybrid state and action mdps. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, San Francisco, USA, 2017.

Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. 2010.

Buser Say, Ga Wu, Yu Qing Zhou, and Scott Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 750–756, 2017.

Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. Interval-based relaxation for general numeric planning. In *ECAI*, pages 655–663, 2016.

Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artificial Intelligence*, 166(1-2):194–253, August 2005.

Ga Wu, Buser Say, and Scott Sanner. Scalable planning with tensorflow for hybrid nonlinear domains. In *Proceedings of the Thirty First Annual Conference on Advances in Neural Information Processing Systems (NIPS-17)*, Long Beach, CA, 2017.