The 27th International Conference on
Automated Planning and Scheduling

June 19-23, 2017, Pittsburgh, USA

Proceedings of the 11th

Scheduling and Planning Applications woRKshop (SPARK)

Edited by:

Sara Bernardini, Shirin Sohrabi, Kartik Talamadupula, Simon Parkinson

Organising Committee

Sara Bernardini, Royal Holloway University of London, UK

Shirin Sohrabi, IBM, USA

Kartik Talamadupula, IBM, USA

Simon Parkinson, University of Huddersfield, UK

Program Committee

Chiara Piacentini (University of Toronto)

Christophe Guettier (SAFRAN)

Gabriella Cortellessa (CNR-ISTC, National Research Council of Italy)

Minh Do (NASA Ames Research Center)

Mark Johnston (JPL/California Inst. of Technology)

Mauro Vallati (University of Huddersfield)

Lukas Chrpa (University of Huddersfield)

Alexandre Albore (Onera & INRA)

Ramiro Varela (University of Oviedo)

Simone Fratini (European Space Agency - ESA/ESOC)

Bram Ridder (King's College London)

Terry Zimmerman (University of Washington - Bothell)

Tiago Stegun Vaquero (MIT and Caltech)

Angelo Oddi (ISTC-CNR, Italian National Research Council)

Riccardo Rasconi (ISTC-CNR)

Nicola Policella (ESA/ESOC)

Patrik Haslum (Australian National University)

Foreword

Application domains that entail planning and scheduling (P&S) problems present a set of compelling challenges to the AI planning and scheduling community that from modeling to technological to institutional issues. New real-world domains and problems are becoming more and more frequently affordable challenges for AI. The international Scheduling and Planning Applications woRKshop (SPARK) was established to foster the practical application of advances made in the AI P&S community. Building on antecedent events, SPARK'17 is the eleventh edition of a workshop series designed to provide a stable, long-term forum where researchers and practitioners can discuss the applications of planning and scheduling techniques to real-world problems. The series webpage is at http://decsai.ugr.es/~lcv/SPARK/ We are once more very pleased to continue the tradition of representing more applied aspects of the planning and scheduling community and to perhaps present a pipeline that will enable increased representation of applied papers in the main ICAPS conference. We thank the Program Committee for their commitment in reviewing. We thank the ICAPS'17 workshop and publication chairs for their support.

The SPARK’17 Organizers

Table of Contents

Traverse Planning with Temporal-Spatial Constraints

John L. Bresina[*], Paul H. Morris[*], Matthew C. Deans[*], Tamar E. Cohen[+], David S. Lees[+]

[*] NASA / [+] SGT, Inc.

NASA Ames Research Center, Mail Stop 269-2, Moffett Field, CA 94035

John.L.Bresina@nasa.gov, Paul.H.Morris@nasa.gov, Matthew.Deans@nasa.gov, Tamar.E.Cohen@nasa.gov, David.S.Lees@nasa.gov

Abstract

We present an approach to planning rover traverses in a domain that includes temporal-spatial constraints. We are using the NASA Resource Prospector mission as a reference mission in our research. The primary objective of this mission is to assess the feasibility of in-situ resource utilization (ISRU) on the lunar surface. One of the mission operations constraints is that the rover is generally required to avoid being in shadow, because it is solar-powered. This requirement depends on where the rover is located and when it is at that location. Such a temporal-spatial constraint makes traverse planning more challenging for both humans and machines. We present a mixed-initiative traverse planner which helps address this challenge.

This traverse planner is part of the *Exploration Ground Data Systems (xGDS)*, which we have enhanced with new visualization features, new analysis tools, and new automation for path planning, in order to be applicable to the Resource Prospector mission. The key concept that is the basis of the analysis tools and that supports the automated path planning is *reachability* in this dynamic environment due to the temporal-spatial constraints.

Introduction

We address the problem of mission planning for a robotic mission that includes temporal-spatial constraints. We are investigating this problem within the context of the NASA Resource Prospector (RP) mission. Within this problem domain, there are two important mission requirements that define temporal-spatial constraints on rover operations: (1) avoiding spending time in shadows, because the rover is solar-powered and (2) maintaining direct-to-earth communications with the operations center (unless the rover is idle), because mission operations requires tight interaction with the ground. Both of these constraints depend on where the rover is located and when it is at that location.

These dynamic constraints make traverse planning more challenging. We have enhanced an existing traverse planner to address these challenges. The traverse planner is part of the Exploration Ground Data Systems (xGDS), developed at NASA Ames Research Center. We have enhanced the xGDS traverse planner by adding new reachability analyses and automated path planning, and we have developed a variety of new visualizations to present the geographical and temporal data. In the following sections, we first present background on the Resource Prospector mission and xGDS and then describe these new enhancements and how they can be employed to manage domains with temporal-spatial constraints.

Resource Prospector Mission

Resource Prospector (RP) is a robotic mission currently in formulation by NASA's Advanced Exploration Systems Division within the Human Exploration and Operations Mission Directorate. The mission's primary objective is to demonstrate the feasibility of in-situ resource utilization (ISRU) on the lunar surface (Andrews, et al., 2014, Colaprete, et al., 2014). In service of this objective, the mission will characterize the distribution of water and other volatiles at the lunar poles, with the goal to map the distribution of surface and subsurface hydrogen-rich materials. The areas of interest fall into one of four categories, defined by their thermal character and *ice stability depth*:

- *Dry*: Temperatures in the top meter expected to be too warm for ice to be stable
- *Deep*: Ice expected to be stable between 50-100 cm of the surface
- *Shallow*: Ice expected to be stable within 50cm of surface
- *Surface*: Ice expected to be stable at the surface; i.e., within a Permanently Shadowed Region, (PSR)

The ISRU processing demonstration will use a hydrogen reduction process to extract oxygen from lunar regolith, demonstrating hardware in the lunar environment and capturing, quantifying, and displaying water generated from ISRU processing.

To keep costs low, the mission will use a solar-powered rover and will use direct-to-earth (DTE) communications to uplink commands and downlink telemetry and science data. Solar power will require landing and traversing in

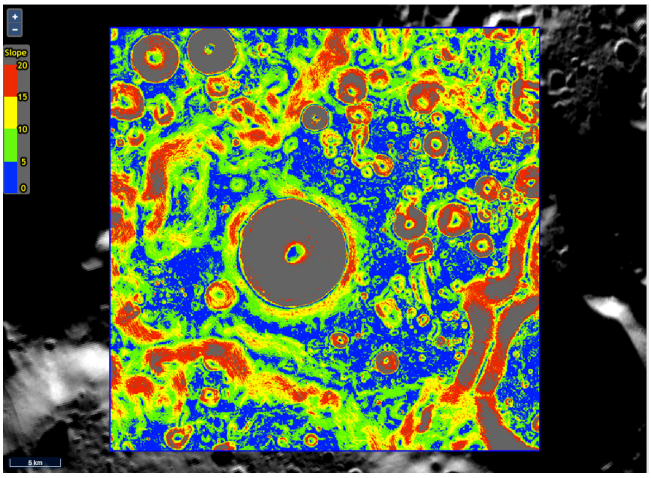


Figure 1: Map of the degree of slope in xGDS map server; the key on the left indicates blue: 0-5, green: 5-10, yellow: 10-15, and red: 15-20.

sunlit areas to maintain power levels, and periodically operating on battery power to collect measurements of volatiles in permanently shadowed regions. DTE communications are required because there are no plans for a concurrent orbital mission and adding an orbiting communication relay is cost prohibitive. Staying in sunlight and in line-of-site of the Earth are temporal-spatial constraints. A third operations constraint is that the rover must avoid slopes that are over fifteen degrees. This is a spatial constraint, with no temporal aspect.

Mission planning for RP involves constructing a *tour* from a lunar landing site and landing time to a number of sites, or *stations*, at which activities are performed (e.g., payload operations), such that the rover is kept safe and the mission objectives are met.

A landing ellipse is considered safe if it satisfies the slope constraint and satisfies the sunlight and comm constraints with a temporal margin of forty-eight hours; that is, the two constraints are satisfied at the time of landing and remain satisfied throughout the next forty-eight hours. A station is considered safe if its location satisfies the slope constraint and from the arrival time until the departure time (i.e., throughout the station *dwell* time), the location satisfies the sunlight and comm constraints. A traverse between two stations is considered safe if each location on the path satisfies the slope constraint and satisfies the sunlight and comm constraints at the time the rover is expected to be at that location.

xGDS Overview

The Exploration Ground Data Systems (xGDS), synthesizes real world data from sensors, robots, ROVs, mobile devices, etc., and from human observations into rich, digital maps and displays for planning, analysis, and decision making. xGDS is a highly collaborative, interactive suite of web software. xGDS has been employed in rover field tests and analog missions, e.g., the Mojave Volatiles Prospector, MVP (Heldmann, et al., 2016).

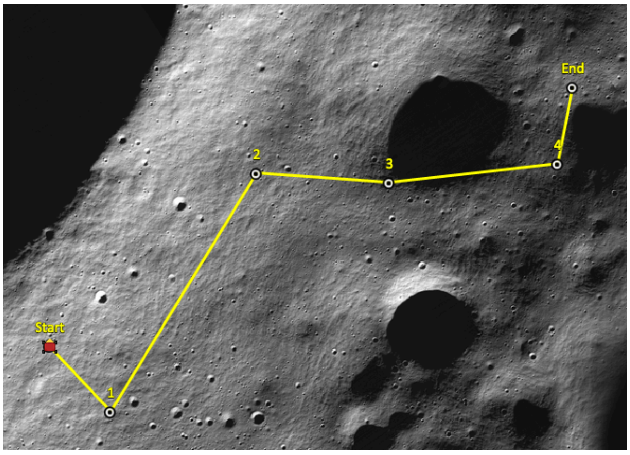


Figure 2: Six-station traverse plan in xGDS traverse planner.

The map server is a repository for organizing multiple layers of map data. xGDS users can manage a tree of map layers, including simple markup, kml imports, and complex tiled geoTIFF data layers. Figure 1 shows an image of the map server displaying slope data for an area of the lunar north pole.

The traverse planner is a map based tool for manually composing traverse plans that involve a sequence of *stations* or places where specific science tasks should be performed. Figure 2 shows an image of a traverse plan with six stations. At each station the user can specify the sequence of activities to be executed at that location. In this case, the sum of the activity durations determines the station *dwell* time. The resulting traverse plans may be passed to a rover for automated execution, or may be printed for use by an astronaut, ROV, or submersible pilot.

With the unenhanced version of this system (e.g., the version used for the MVP field test), constructing traverse plans for the RP mission, that satisfy the sunlight and comm temporal-spatial constraints, is a manually intensive, tedious, and error-prone process. In order to validate a traverse between two stations, the user has to examine each time frame of the sunlight and comm data and check that the rover's location on the traverse path (given the specified speed) is safe at that time. Similarly, in order to validate the activity sequence at a station, the user has to manually determine that the station's location is safe at each time frame within the station's dwell time.

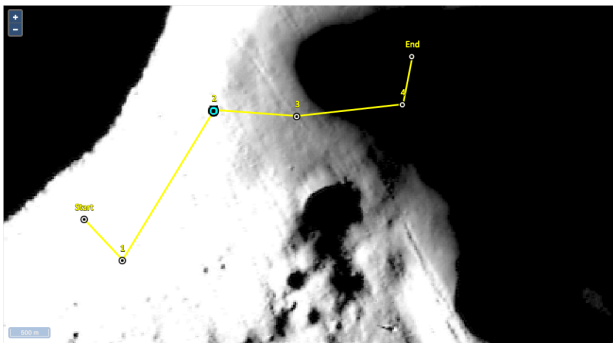


Figure 3a: Sunlight at arrival time of station 2 (blue circle).

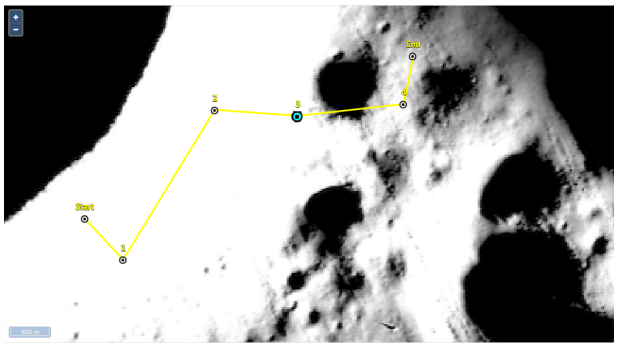


Figure 3b: Sunlight at arrival time of station 3 (blue circle).

In the following sections we describe the key capabilities we have added to xGDS to make the planning process easier and safer via new visualization capabilities and new automation in terms of reachability analyses and path generation. The new automation capabilities reduces the burden on the user and makes the xGDS traverse planner an effective mixed-initiative system.

Visualization Enhancements

For this project, we have adapted xGDS' map server and traverse planner to meet the needs of the RP mission. For planning lunar traverses, we have configured the map server to render lunar maps in polar stereographic projection, using base map data synthesized at Arizona State University from LROC imagery (http://lroc.sese.asu.edu/).

In order to help address the difficulties of manually evaluating the temporal-spatial constraints, we have added the capability to display time varying interactive map layers; for example, to show the sunlight and comm coverage maps over time. As a user composes and edits a traverse plan, the sunlight and comm coverage maps automatically update to display the conditions at the currently selected time. Figures 3a and 3b show the sunlight maps at the arrival time of station 2 and 3, respectively; locations are white if the visible area of the Sun's disc is over the 80% threshold. All the layers can be toggled to appear or not. In addition, the degree of transparency of each layer can be controlled in order to allow users to simultaneously view geographic features and other map data.

We have also added plots of scalar data along the traverse plan against time (see Figure 9). We have incorporated several relevant data layers representing ice stability depth, slope, elevation, and water equivalent hydrogen, in order to help the user evaluate the suitability of a site for meeting the science objectives. As users compose and edit traverse plans, they can see detailed specific values over time in the plots and correlate them with positions in the map.

These new visualization capabilities increase the user's situation awareness of the dynamic lunar environment, and enable a richer, more informed interaction with the mixed-imitative system.

Mixed-Initiative Enhancements

In this section, we describe xGDS enhancements involving tools that facilitate the mixed-initiative construction of a mission plan in the face of temporal-spatial constraints. The three new tools we discuss are: (1) reachability from a station within N hours, (2) temporal bounds on station arrivals and departures, and (3) automated generation of detailed paths between stations. These tools are user-invoked within the xGDS traverse planner interface. The response time of each tool varies depending on the specific inputs, but xGDS imposes a maximum of ten seconds. If this limit is exceeded, computation is halted and an error message is reported to the user.

A landing location and time can be chosen such that several areas of scientific interest will be reachable within the duration of the mission. Once that has been determined, a finer grained analysis of the reachability information will show temporal intervals when a location of scientific interest can be visited. Generally, there may be a disjunction of such intervals depending on the movement of shadows across the moonscape. We use the earliest interval to provide an earliest and latest arrival time. By repeating the reachability analysis, station to station, we can assemble a tour of science sites of interest.

Mission Planning Example

For the purpose of this example, we assume that the landing site (Start station) and landing time has been determined. Furthermore, we assume that the checkout activities at the Start station take four hours.

Our initial task is to choose a location for the next station. The first enhancement involves a tool that supports station selection by computing a *reachability map*. The inputs to this tool include the current station's location and departure time, and the maximum number (N) of hours to compute reachability. The tool determines all the locations

that can be reached from the current station in N hours or less, taking into account the dynamic and static constraints. This is depicted in the xGDS traverse planner as shown in Figure 4, where N is 96.

In addition to the reachability map, Figure 4 also shows an ice stability depth map layer, which helps identify regions of scientific interest. The colors in this layer indicate the following depth ranges in meters: green is 0-0.3, yellow is 0.3-0.7, red is 0.7-1.0, black is greater than 1.0. This ability to overlay different maps in xGDS allows the user to take into account both reachability and scientific value when selecting candidates for the next station.

Station 1 is chosen and activities are added in order to acquire and analyze a subsurface sample; these activities

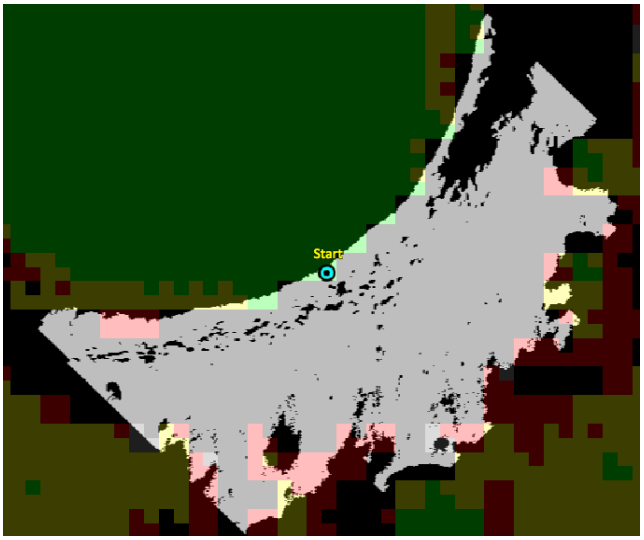


Figure 4: Reachability within 96 hours from the first station.

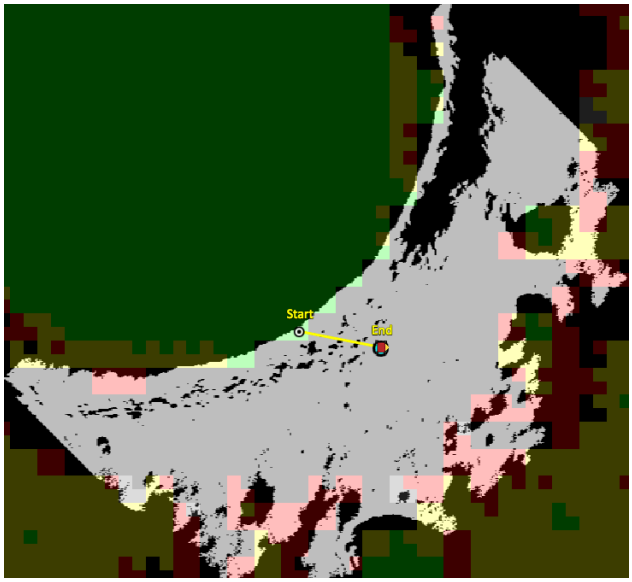


Figure 5: Reachability within 96 hours from the second station.

take a total of four hours. The reachability computation is repeated, based on the departure time from station 1, to help choose the End station, as depicted in Figure 5. Though the shapes of the two reachability maps look similar, they cover a different region of the terrain, as indicated by the differences in the ice stability depth map layer. Figure 6 shows the resulting three-station traverse plan, along with the ice stability depth layer, which is rendered partially transparent to let the terrain layer be seen as well. This provides a small candidate tour.

Based on the reachability analyses, we are ensured that this tour is safe; that is, the dynamic and static constraints are never violated. However, given the temporal uncertainties of execution, we would like to know how robust the tour is. One important measure of robustness is how much *temporal margin* does the traverse plan have, that is, how much flexibility does the tour have in terms of arrival and departure times?

Our next xGDS enhancement provides a tool for computing earliest arrival and latest departure time bounds that must be adhered to in order to satisfy the dynamic constraints. There are two variants of these bounds. One involves the interval of the earliest and latest times one can be at a station without directly violating a shadow constraint. These are called the *local* bounds; they are important for ensuring the safety of the rover. The second variant involves the earliest arrival and latest departure times needed in order to ensure the completion of the tour. Essentially, this involves propagating the local bounds from each station to the other stations in the tour, taking into account the traverse constraints. We call these the *global* bounds. These help to determine the feasibility of the tour. As an example usage of this tool, Figure 7 shows the computation of both the local and global bounds for station 1.

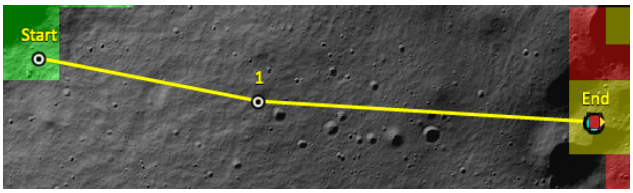


Figure 6: Three station traverse plan.

Station 1:

Local Earliest Arrival:	2020-09-22T14:00:00.000Z
Local Latest Departure:	2020-09-24T18:00:00.000Z
Global Earliest Arrival:	2020-09-23T20:00:00.000Z
Global Latest Departure:	2020-09-24T16:00:00.000Z

Figure 7: Local and global temporal bounds for station 1.

In the display of the traverse plan in xGDS, the straight-line traverse segments between two stations denote the *existence* of safe paths between the station pairs, based on the reachability analysis. At some stages of mission plan-

ning, this level of abstraction may be sufficient. However, when more detailed evaluation of the traverse plan is needed, then a specific detailed path between every pair of stations must be generated. This is the case, for example, if the power generated by the solar panels and the power consumed by the rover must be computed in order to ensure that the battery state of charge adheres to mission flight rules throughout the tour.

In the unenhanced xGDS, it was the user's responsibility to manually construct the detailed paths, using the multiple layers of visual data in order to satisfy safety constraints, e.g., slope. In a domain like RP, this task becomes quite difficult and error-prone. Hence, our final enhancement involves a path planning tool that automatically generates a detailed path between two given stations. The xGDS user invokes the automated path planner on a subsequence of the traverse plan by specifying the start and end stations. The returned best paths found are then displayed in the xGDS traverse planner (Figure 8).

All these tools make use of a more basic analysis of *reachability* from one or more specified locations at a start time to one or more specified locations at an end time. In effect, this computes a compressed representation of all of the valid paths from any of the locations at the start time to any of the locations at the end time. More precisely, for each of the intermediate location/time pairs, it determines whether any such valid path passes through that location and time. This analysis directly supports the reachability map tool. In this case, the initial set of locations consists of a single location: the starting point.

For the temporal global bounds, we apply the analysis to each successive pair of locations, for example, from the Start station to station 1, and from station 1 to the End station. In both cases, we have a single start location/time and can search forward in time to find the earliest time that the end location is reached. This gives the lower global bound for the destination. We then take the latest local bound for the last station and do successive *backwards* reachability analyses to provide the global upper bounds.

For path planning, the reachability data from a single starting point provides, in effect, a perfect minimum-time heuristic for path planning backwards from the destination location at its earliest reached time, since all backward paths from there lead to the start location in the same amount of time. These candidate paths can be searched to select the best solution based on additional criteria.

The next two subsections go into further detail about how the reachability maps are computed and how the detailed paths are automatically generated.

Computing Reachability

The majority of the RP mission involves traverse plans that are restricted to sunlit areas. There may be short sorties to shadowed areas but for our calculations we are requiring traverse plans to be in sunlight. The input data includes a set of images, or *frames*, covering the entire period of interest taken at 2-hour intervals with each pixel representing a 20 by 20 meter area. The individual pixels in the frames have boolean values representing a threshold level of sufficient sunlight. There is also a threshold on slopes that are considered traversable but this restriction can be effectively combined with the boolean sunlight frames.

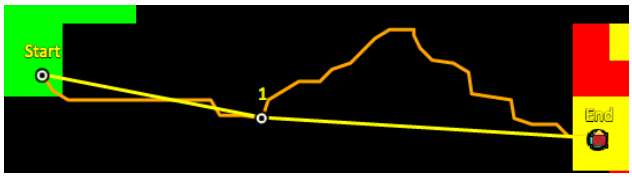


Figure 8: Results of automatic generation of two paths.

Traverse plans are also constrained by communication (comm) availability. This restriction differs from sunlight in that the rover is permitted to be at a sunlit location with no comm, but may not move until comm is restored. The absence of comm requires the rover to be idle.

Consider a particular frame F_i and the next frame F_{i+1}. A location in F_{i+1} is reachable from a location in F_i if there is a path between the locations that is entirely in sunlight in both frames. The path cannot be longer than is feasible given the top speed of the rover. Furthermore, the path must be continuously in comm. Conservatively, the sun conditions need to be satisfied in both F_i and F_{i+1} to determine a valid path; thus, we intersect them to get a combined frame G_i that restricts the paths. Similarly, we intersect the comm conditions.

The no-comm path between frames is straightforward since it involves the rover staying at the same location. Otherwise we require both sun and comm for the entire path between frames. We form an intersected sun/comm frame to enforce this requirement. Rather than directly computing valid paths between successive frames, we construct intermediate subframes where the time slice between subframes is just long enough to permit possible movement to an adjacent pixel. For example, since the pixels are 20 meters apart and assuming a top speed of 80 meters per hour, then the subframes are 15 minutes apart and there are 8 subframes between the original 2 hour frames. The sun/comm frames provide gate conditions for the transition between subframes. The criterion for movement between subframes is that only paths of length 0 or 1 in pixel length are allowed. For simplicity, we exclude diagonal moves; thus, the rover can move up, down, left or right, or stay in the same location, depending on the values of the bits in the combined sun/comm frame.

In our current examples, each frame constitutes a 2176 x 2176 boolean array with 108 successive frames covering

the time period of interest. With such a large space of values, it is useful to take advantage of bit-array compression and significant parallelism. When the initial set of locations is represented by a bit-array, the movements can be simulated by up-shift, down-shift, left-shift, and right-shift operations. The shifted and original location bit-arrays can then be ORed together and the resulting bit-array ANDed with the sun/frame condition bit-array to get a bit-array of the reached locations.

In our implementation, we represent the bit-arrays as C++ Standard Template Library (STL) bitsets in row-major order. Then a 2-D up-shift operation can be simulated by a 1-D shift of 2176 bit locations in the bitset, and similarly for a down-shift. The 2-D left-shift operation is essentially the same as a single 1-D shift. However, it has to be modified to ensure that zeros are shifted into the rightmost bit-positions in each row. This approach takes advantage of the low-level parallelism in the standard implement of STL bitsets. For example, the boolean and shift operations can be performed on 64-bit word chunks as single CPU operations. In the future, we would like to investigate adding high-level parallelism by using some form of GPU acceleration.

Generating Detailed Paths

Our approach to automatically generating traverse paths uses a sampling approach, called *Heuristic-Biased Stochastic Sampling (HBSS)*. The sampling is performed within a space of minimum-time paths, which is determined via the reachability analysis. This method was chosen for two reasons: (1) it is space-efficient, compared to algorithms that guarantee an optimal solution (e.g., A*), and (2) it is well-suited to problems that have a large, not well characterized search space, such that an effective greedy heuristic is difficult to determine.

Heuristic-Biased Stochastic Sampling was first introduced in (Bresina, 1996), and it encompasses a family of search techniques that combine some degree of heuristic guidance and some degree of stochastic search, with greedy search and unbiased random search being extreme members. Within HBSS, the desired balance between heuristic adherence and exploration in the search space is determined by specifying a *bias function* and a *ranking function*. The ranking function partitions the heuristic's range into equivalence classes and determines the magnitude of the quality differences between classes. The bias function is applied to each choice's rank to determine its weight. A stronger bias tends to follow the heuristic's advice more often and a weaker bias tends to explore farther off the greedy trajectory in the search tree. The accuracy of the search heuristic is an important factor in choosing the bias function to use; typically, the less accurate the heuristic, the weaker the bias should be. Another important factor is the amount of solution generation time available; if there is not much time available, then exploration must be limited.

For our path planning approach, we use a version of HBSS called *Multi-Bias HBSS*. Instead of using the same bias function on each sample, a set of bias functions are given and used alternatively. This method makes the results less dependent on guessing the best bias to use.

The inputs to the path generation tool are the following.

- The start time for the plan.
- The rover maximum speed.
- A sequence of stations, each associated with the dwell time at the station.
- The end time of the mission.

The solution is a sequence of position and time pairs (p_j, t_j), that satisfies the constraints, starts at the traverse plan start time, dwells at each station as specified in the traverse plan, and the traverses do not exceed the maximum speed.

Another solution requirement is "survivability until the end of the mission". After reaching the end station and dwelling there as specified, the rover must be able to survive (i.e., satisfy all the constraints) until the mission end time. This survival might be achievable by staying at the end station until the end time, but typically will involve moving to avoid shadows. Where the rover ends up at mission end time is not constrained.

From a position, we restrict the rover movements to the four adjacent positions: up, down, left, or right; or the rover can stay in the same position. Since the pixels (or position cells) are 20 meters apart, the time delta between t_j and t_{j+1} is 20 / maximum speed. However, not all of these next pairs are valid. Hence we want to restrict the sampling space to only valid moves, which is exactly what the reachability analysis determines.

The following is how we use reachability to restrict the sampling space to the solution space. We first perform the reachability forward-sweep from the start station at the plan start time, through all the stations, respecting the dwell times, then forward in time until the mission end time. This determines the earliest arrival time at each station and determines all the valid intermediate position-time pairs. For each sample, we then start at the end station and work backwards to the start station. For each consecutive pair of stations, we find a valid path between (p_{j+1}, t_{j+1}) and (p_j, t_j) where the times represent the station's earliest arrival time. The search finds earliest time paths; that is, it does not consider paths where the rover stays in the same position longer than necessary.

At each step in this process, we select from the valid neighbors determined by the reachability forward-sweep. Thus, by using the reachability analysis to restrict the sampling space, we ensure that every path generated is valid. The next section describes how this sampling search is guided and how the solutions are evaluated in order to return the best solution found among the samples.

In order to instantiate HBSS for a specific problem domain, we define the following: a heuristic function, a ranking function, a bias function (or multiple bias functions in this case), and a solution evaluation method. Our path planning heuristic is Manhattan distance. The ranking function groups choices in the same equivalence class only if they have the same heuristic score, where better score means lower rank. As mentioned, we use multiple bias functions; they are all polynomial functions, of the rank, with exponents of -1 through -10.

Our solution evaluation method uses two criteria. The primary criterion is the minimal distance travelled, based on the sum of the Manhattan distance between all consecutive pairs of position-times. The secondary criterion is based on temporal margin within the reachability space. The *temporal margin* at a position-time is how much longer the rover can stay at that position and still be able to survive until the mission end time. The metric we are using is the minimum of the margin over all position-time pairs in the path, where the largest minimum margin is preferred. This secondary criterion is only used to break ties with respect to the distance criterion. To generate each of the two paths in Figure 8, 1, 000 samples were used.

Concluding Remarks

In this section, we mention some related work, describe future directions we are pursuing, and conclude.

Related Work

An early system that integrated activity planning, resource management, and traverse planning is TEMPEST (Tompkins, Stentz, Wettergreen, 2004). The system employed the *Incremental Search Engine (ISE)*, which is a graph-theory based, heuristic search algorithm that supports planning in high-dimensional spaces.

There is other ongoing work based on the RP problem domain; see, for example, (Cunningham, et al., 2014). To cope with the large search space, they perform a temporal compression of the dynamic terrain data and use an hierarchical planning approach. A low-resolution planner computes feasible paths based on pre-computed results from a high-resolution planner. The search is carried out by an A* approach. The state space is reduced via the temporal compression and by using state dominance to prune states during the search. A similar approach is taken in (Otten, et al., 2015), which uses a combination of connected components analysis to find interconnected regions in space and time, and A* to find optimal routes within those components.

Another effort that addresses similar constraints is that of Peng and Hehua (2013). The constraints include sunlight and earth visibility, as well as constraints on thermal, energy, and storage resources. Their approach collapses all constraints to temporal ones and solves a time-lining problem. The approach is applied to a lower latitude site, where the lighting constraint can be modeled as temporal where the sun elevation is above a threshold elevation angle.

The *Surface Exploration Traverse Analysis and Navigation Tool (SEXTANT)* is a related system, because not only does it automatically generate paths using multi-criteria optimization, it also has been integrated with xGDS as part of the Minerva Architecture (Deans, et al., 2017) employed in the Biologic Analog Science Associated with Lava Terrains (BASALT) project at NASA. A unique aspect of SEXTANT (Johnson, et al., 2010) is that in addition to being applicable to rover traverse planning, it is well-suited to human traverse planning as it models human consumable resources as well as thermal load. This is important for BASALT as the explorations are carried out by geologists (simulated astronauts). The traverse optimization in SEXTANT is accomplished by an A* search algorithm.

Fink, et al., (2015) present a related automated path planner called *Rover Traverse Optimization Planner (RTOP),* which employs a multivariate stochastic optimization framework using Simulated Annealing. The criteria they have considered include 3D Euclidian distance, traverse roughness, and slope-change. However, they have not dealt with dynamic constraints like shadow avoidance.

Future Work

One of the aspects that we would like to improve is the solution evaluation within the HBSS path generation. There are a number of other solution quality measures that could be considered. Currently we are using terrain slope as one of the hard constraints, but we could also use slope in an "ease of travel" metric in the solution evaluation.

Another important solution quality consideration is energy; the solution path affects both the amount of energy generated by the solar panels and the energy consumed by the traversal. However, this is a non-trivial metric to model, and would increase the computational time of HBSS.

We are pursuing the support of more complex activity planning. The xGDS system does support generation of simple sequential activity plans at the stations. This facility has been sufficient for the many rover field tests and analog mission that employed xGDS; however, some deployments require more complex forms of activity plans and activity planning capabilities. We are still evaluating whether the RP mission is one of those domains.

As a first step towards achieving this objective, we have integrated xGDS with a powerful activity planning system, called OpenSPIFe, which is derived from the Ensemble system that has been deployed on NASA missions (Phoenix, MSL, LADEE), as well as on the ISS and analogue missions. The Scheduling and Planning Interface for Exploration (SPIFe) provides a rich environment for creating

activity plans. OpenSPIFe also includes a back-end, powerful constraint reasoning system, called *Dynamic Europa*, which detects temporal violations and state resource violations, and provides a mixed-initiative facility for resolving these violations. For more details see (Morris, et al., 2011).

Conclusion

In this paper, we presented a mixed-initiative traverse planning approach applicable to NASA's Resource Prospector mission. We enhanced the xGDS traverse planner with new visualization capabilities, new reachability maps from a station, new temporal bounds computations on station arrivals and departures, and new automated detail path generation between stations (Figure 9). The primary planning challenge inherent in this mission is dealing with temporal-spatial constraints involving maintaining line-of-sight for communications and shadow avoidance. The temporal-spatial concept of *reachability* is key in supporting both the human user in the mixed-initiative construction of traverse plans and in supporting the HBSS algorithm.

The RP mission is still in the early stage of planning and design, and the work reported here is part of an ongoing research project. We are in frequent communication with members of the RP operations team and science team to facilitate technology transfer to the mission and to help focus our research on area that would most benefit the mission. We plan to collect feedback from the RP mission team, as part of our comparative evaluation of these two system architectures, in order to help determine which one is the best fit for the mission.

Acknowledgements

This work is part of the Autonomous Systems and Operations project, lead by Jeremy Frank and funded by the Advanced Exploration Systems program.

We would like to acknowledge Rick Elphic (RP Mission Scientist), Mark Shirley (System Engineer for RP Mission Operations), and Andy McGovern (RP Science Team member). Mark has been pursuing similar concepts in his prototype traverse planner and we have benefitted from exchanging ideas. Rick has been our primary source of planning requirements of the RP Science team, and he has provided valuable feedback on the usability of our enhanced xGDS traverse planner. Andy generated the lighting and communication maps we used.

We would also like to acknowledge Trey Smith and Anthony Colaprete (Lead of the RP Science team). Trey helped with the interface between xGDS and the new tools. Anthony has been very supportive of our work and its application to the Resource Prospector mission.

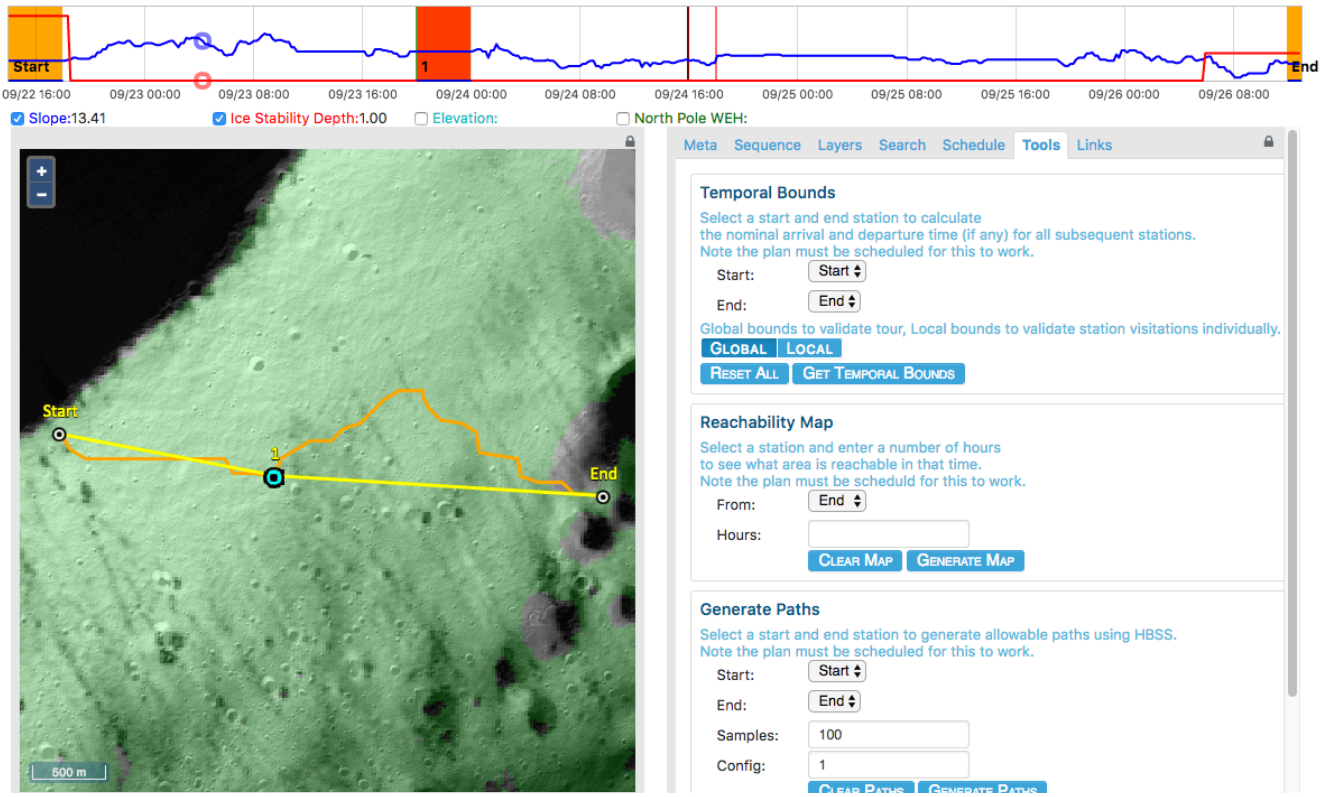


Figure 9: The enhanced xGDS traverse planner. The bottom, left is the enhanced display of the traverse plan; the bottom, right shows the interfaces to the three new tools; and the top shows the new timeline of the plan overlaid with the new data plots versus plan time.

References

Andrews, D. R., Colaprete, A., Quinn, J., Chavers, D., and Picard, M. 2014. Introducing the Resource Prospector (RP) Mission. In *Proceedings of the AIAA SPACE 2014 Conference and Exposition*. AIAA SPACE Forum.

Bresina, J.L. 1996. Heuristic-Biased Stochastic Sampling. In *Proceedings of the 13th AAAI Conference.* Portland, OR. AAAI Press.

Colaprete, A. 2014. Resource Prospector: A Lunar Volatiles Prospecting and ISRU Demonstration Mission. *NASA Exploration Science Forum*.

Cunningham, C., Jones, H., Kay, J., Peterson, K.M., Whittaker, W.L. 2014. Time-Dependent Planning for Resource Prospecting. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*. Montreal, Canada.

Deans, M., Marquez, J., Miller, M., Hoffman, J., Lim, D. 2017. Minerva: User-Centered Science Operations Software Capability for Future Human Exploration. In *Proceedings of the IEE Aerospace Conference*. Big Sky, Montana. IEEE.

Fink, W., Baker, V.R., Flammia, M., Tarbell, M.A. 2015. Rover traverse-optimizing planner for multi-objective deployment scenarios. In *IEEE Aerospace Conference*. Big Sky, MT. IEEE.

Heldmann, J.L., Colaprete, A., Elphic, R., Lim, D., Deans, M., Cook, A., Roush, T., Skok, J., Button, N., Karunatillake, S., Stoker, C., Marquez. J., Shirley, M., Kobayashi, L., Lees, D., Bresina, J., Hunt, R. 2016. Lunar polar rover science operations: Lessons learned and mission architecture implications from the Mojave Volatiles Prospector terrestrial field campaign. In *Advances in Space Research*, Vol 58, Issue 4, pp. 545-559. Elsevier.

Johnson, A.W., Hoffman, J.A., Newman, D.J., Mazarico, E.M., Zuber, M.T. 2010. An Integrated Traverse Planner and Analysis Tool for Planetary Exploration. In *Proceedings of the AIAA SPACE 2010 Conference & Exposition.* Anaheim CA. pp. 1-28.

Morris, P., Bresina, J.L., Barreiro, J., Iaturo, M., Smith, T., 2011. State-Based Scheduling via Active Resource Solving. In *Proceedings of the Fourth IEEE International Conference on Space Mission Challenges for Information Technology*. Palo Alto, CA, pp.29-34, IEEE.

Otten, N.D., Jones, H.L., Wettergreen, D.S., Whittaker, W.L. 2015. Planning routes of continuous illumination and traversable slope using connected component analysis. In *IEEE International Conference on Robotics and Automation*, pp. 3953-3958. IEEE.

Peng, Wu and Hehua, Ju. 2013. Mission-Integrated Path Planning for Planetary Rover Exploration. In *Journal of Software*, Volume 8, Number 10, pp. 2620-2627. Academy Publisher.

Tompkins, P., Stentz, A., Wettergreen, D. 2004. Global path planning for Mars rover exploration. In *Proceedings of the IEEE Aerospace Conference*. Volume 6. IEEE.

A Case for Collaborative Construction as Testbed for Cooperative Multi-Agent Planning*

Sven Koenig
Department of Computer Science
University of Southern California
skoenig@usc.edu

T. K. Satish Kumar
Department of Computer Science
University of Southern California
tkskwork@gmail.com

Abstract

Cooperative multi-agent planning is an understudied but important area of AI planning due to the increasing importance of multi-agent systems. In this paper, we make the case for using collaborative construction as testbed for cooperative multi-agent planning. Planning for single agents is already difficult due to the large number of blocks and long plans. Planning for multiple agents is even more difficult since it needs to reason about how to achieve a high degree of parallelism without agents obstructing each other even though many agents operate together in tight spaces. In previous research, we developed a first (domain-dependent, centralized and non-optimal) multi-agent planning method for this domain. Here, we explain the advantages of using collaborative construction as multi-agent planning domain, formalize the planning problem and relate it to existing planning problems in the hope that other researchers will adopt it as testbed for cooperative multi-agent planning.

Introduction

Cooperative multi-agent planning is an important area of AI planning due to the increasing importance of multi-agent systems. For example, teams of agents are more fault-tolerant and allow for more parallelism than single agents. Multi-agent planning promises to coordinate agents much more efficiently than alternative coordination strategies, such as — for example — behavior-based, stigmergy-based or market-based methods, which are typically very myopic. Yet, cooperative multi-agent planning is currently understudied. For example, only two out of 20 sessions at the International Conference on Automated Planning and Scheduling 2016 were on distributed and multi-agent planning.

In this paper, we make the case for using collaborative construction as testbed for cooperative multi-agent planning. Building on our previous research (Kumar, Jung, and Koenig 2014; Cai et al. 2016), we suggest to simulate the Harvard TERMES robots, actual robots that were inspired by termites. Among many other species of animals, termites are capable of building mounds that are much larger than themselves. Inspired by termites and their building activities, the Harvard TERMES project investigated how multiple robots can cooperate to build user-specified three-dimensional structures much larger than themselves (Petersen, Nagpal, and Werfel 2011). The agents need to build ramps to reach high places and avoid obstructing each other. Currently, the TERMES robots coordinate using local behavior-based and stigmergy-based rules, which make it impossible for them to construct complex structures. Planning is required, even for single agents, to build structures effectively since they need to build ramps to reach high places, for example when building towers. Ramps consist of many blocks and are time-consuming to build. Thus, agents need to plan carefully when and where to build ramps and, once built, how to utilize them best. Planning for single agents is already difficult due to the large number of blocks and long plans. Planning for multiple agents is even more difficult since it needs to reason about how to achieve a high degree of parallelism without agents obstructing each other even though many agents operate together in tight spaces. Furthermore, single agents no longer have to carry blocks all the way to their destinations since agents can hand over blocks to each other. For example, they can form bucket brigades to transport blocks. Ideally, one would like to plan for a large number of agents, such as one hundred agents or more.

Advantages of Collaborative Construction

Advantages of using collaborative construction as testbed for cooperative multi-agent planning include: The TERMES robots are very simple to simulate and allow for deterministic and symbolic planning. Robotics domains are often continuous domains with a substantial amount of sensor and actuator uncertainty, while AI planning is often studied in the context of symbolic domains without uncertainty.

*This paper re-uses a small amount of text from our previously published feasibility studies at ICAPS and AAMAS. We thank Marcello Cirillo, Tansel Uras and Liron Cohen for their suggestions and helpful discussions, Tansel Uras and Liron Cohen also for their help by experimenting with general-purpose planners in our domain, and Radhika Nagpal for commenting on a draft of this paper, her interest in our project and her encouragement. Our research was supported by NSF under grant numbers IIS- 1319966 and IIS-1409987 and ONR under grant number N00014-09-1-1031. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

Collaborative construction fits the latter assumptions well. For example, the TERMES robots move on the blocks and the environment is thus automatically discretized into square cells. The TERMES robots use hardware features to achieve close-to-perfect execution. For example, a white cross on a black background of each block helps them to track both their position and orientation. Moreover, a circular indentation on each block helps them to turn in place without accumulating drift. Collaborative construction is thus a good domain for demonstrating the applicability of AI planning to robot planning, which will help to bridge the current gap in planning between AI and robotics. Collaborative construction pushes the state-of-the-art of AI planning but is not too difficult. There is a potential progression of research from centralized planning for single agents, via centralized planning for multiple agents to decentralized planning for multiple agents. At the same time, collaborative construction is also rich in structure that can be exploited for efficient and effective planning. In particular, spatial constraints, different from temporal constraints, are an understudied but very important area of AI planning due to the increasing importance of physical agents, such as robots, that operate in tight spaces. Collaborative construction subsumes some previously studied planning problems with spatial constraints. For example, parallelism comes with the overhead of having to coordinate multiple agents so that they neither collide with each other nor block each other. This introduces combinatorial problems akin to multi-agent path finding. Multi-agent path finding is concerned with multiple agents having to navigate effectively in tight spaces (Sharon et al. 2015; Wilt and Botea 2014), such as spaces with long narrow corridors where agents cannot pass each other. Multi-agent path finding is often studied in the context of automated warehouse domains, such as the Amazon fulfillment centers (Wurman, D'Andrea, and Mountz 2008), but is also relevant for collaborative construction since ramps are time-consuming to build and thus will typically be so narrow to not let two agents pass each other. Overall, we expect research on collaborative construction to yield insights into spatial multi-agent planning that go well beyond collaborative construction.

Hardware System

Our description of the TERMES hardware system follows (Petersen, Nagpal, and Werfel 2011; Werfel, Petersen, and Nagpal 2011; 2014), see Figure 1.[1] It consists of small autonomous mobile robots and a reservoir of passive "building blocks," simply referred to as "blocks." The robots gather blocks from the reservoir to collaboratively build a user-specified structure. The robots are roughly of the same size as the blocks. Yet, they can manipulate these blocks to build structures that are much larger and taller than themselves. They do so by stacking the blocks onto each other and building ramps to scale to greater heights.

The robots are equipped with four small whegs that allow for different kinds of locomotion using the same action of simply driving forward. The whegs allow the robots to move on level ground, climb up one block (to reach higher levels) and climb down one block (to reach lower levels) without any additional hardware or software capabilities, making this a reliable operation without complicated low-level control and allowing one to focus on high-level planning. The robots can navigate on a partially built structure (or the ground) without losing track of where they are or falling down. They are equipped with an arm and a gripper to facilitate picking-up, carrying and dropping-off blocks, one at a time. Mechanical features of the blocks also help the robots to perform these operations reliably with the use of only one actuator. One case where actuation is often not sufficiently accurate, which we ignore in the formalization below, is that it is difficult for robots to drop off a block directly between two other blocks (Petersen, Nagpal, and Werfel 2011).

[1] See www.eecs.harvard.edu/ssr/projects/cons/termes.html for more information.

Formalization of Collective Construction

We are given a start configuration and a desired goal configuration in form of user-specified 2D matrices of non-negative integers, referred to as workspace matrices. The cells of the matrices represent physical locations on a grid frame of reference, and the non-negative integers represent the heights of the towers (that is, vertical stacks of blocks) that need to be constructed by stacking blocks at those cells. (A height of zero means no tower, represented by a missing number in our figures.) Thus, the configurations do not have blocks that rest only partially on top of other blocks nor completely enclosed spaces, such as rooms in a house. As an example, consider an empty start configuration and a goal configuration that represents a castle that consists of a tower of height three surrounded by a wall of height one, as shown in Figure 2(d).

At any intermediate stage, the top of a tower is called *reachable* if and only if an agent, starting from the ground level, can reach the top of that tower by repeatedly turning left, turning right and driving forward. Turning left and right turns the agent 90 degrees in place. Moving the agent forward moves it to the neighboring tower in front of it as long as the agent moves at most one block up or down. Each agent can carry at most one block. The agent can pick up a block from the top of a tower if and only if there is a neighboring tower (in one of the four main compass directions) of height one less, the top of which is reachable, because it can then move to this neighboring tower and pick up the block. The agent can drop off a block on top of a tower if and only if there is a neighboring tower of equal height, the top of which is reachable, because the agent can then move to this neighboring tower and drop off the block. The problem is to build a user-specified structure with a given number of agents.

State-of-the-Art of Collective Construction

Currently, the TERMES robots coordinate using local behavior-based and stigmergy-based rules (Petersen, Nagpal, and Werfel 2011), which make it impossible for them to construct complex structures. There has

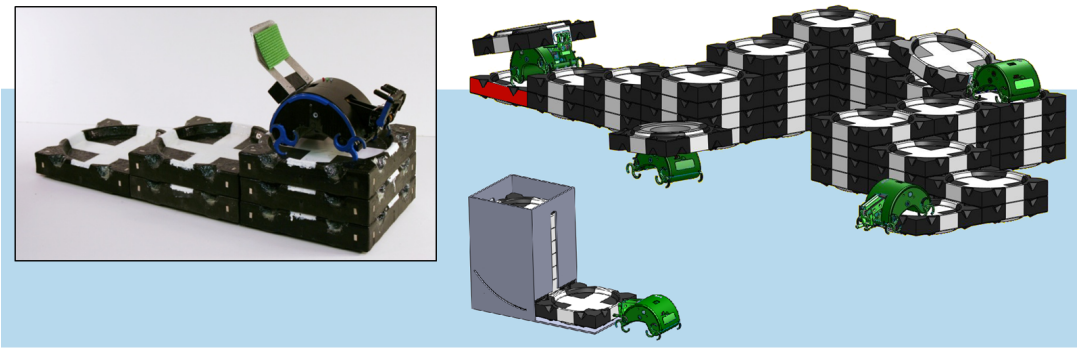


Figure 1: The TERMES system consisting of robots, blocks and the reservoir [figure courtesy of (Petersen, Nagpal, and Werfel 2011)].

been a fair amount of theoretical work on collective construction with different assumptions, including – but not limited to – (Jones and Mataric 2004; Grushin and Reggia 2008; Napp and Klavins 2010; Yun and Rus 2010). Many of these models are abstractions that do not model real hardware and often also make additional simplifications to allow either for a theoretical analysis or the development of simple planning methods.

Existing Planning Problems

The collective construction problem has similarities to the blocksworld and logistics problems, both of which have extensively been used in planning competitions (McDermott 2000). The blocksworld problem is concerned with building towers of blocks. The constraints are due to the vertical arrangements of blocks since only the top block of each tower can be accessed. How the blocks are moved does not impose any constraints, except that only a given number of blocks can be moved at the same time. The logistics problem, on the other hand, is concerned with moving objects to given cities. The constraints are due to how objects can be moved from city to city, given by the transportation options and their capacities. The spatial arrangement of objects does not impose any constraints. Planning researchers have also studied combinations of the blocksworld and logistics domains (Ghallab, Nau, and Traverso 2016) where both the spatial arrangement of objects and how they are moved imposes constraints, for example, in the context of moving containers in container terminals.

Collaborative construction imposes more constraints than the blocksworld, logistics or container terminal problems since blocks can be picked up from the tops of towers only when certain spatial conditions hold, blocks can be put down on tops of towers only when certain spatial conditions hold, and they need to be carried from their pick-up locations to their drop-off locations along spatially-feasible paths. This presents a variety of issues not present in the existing planning problems. For example, ramps of many blocks need to be built to satisfy the spatial conditions, resulting in long plans with many objects — which makes planning very time consuming already for a single agent. Collective construction with multiple agents is even more difficult since one needs to figure out how to achieve a high degree of parallelism even though the state space grows exponentially in the number of agents and multiple agents can easily obstruct each other in tight spaces.

Feasibility Study

No planning methods existed for collaborative construction with agents that correspond to the Harvard TERMES robots. We therefore considered it important to develop a planning method for this domain in a feasibility study before advocating its use as testbed for cooperative multi-agent planning. We assume in the following for simplicity that the reservoir (that provides the blocks) is unlimited and that the start configuration is empty, that is, all blocks are initially stored in the reservoir.

One intuitive single-agent planning method, called the tower-by-tower planning method, is to build the towers one by one, starting from one of the corners, each time constructing a ramp and then deconstructing it again. In this approach, the agent needs to build a ramp consisting of towers of heights $h-1, h-2 \ldots 1$ to build a tower of height h. The ramp is then deconstructed, resulting in $O(h^2)$ total number of block (pick-up and drop-off) operations to build a tower of height h. This intuitive planning method is correct, is complete, runs in polynomial time and performs a polynomial number of block operations for any user-specified structure. It demonstrates that any structure can be built by one or more agents provided that there is sufficient empty space around it. However, the tower-by-tower planning method is not very effective even for simple structures.

We therefore first developed a better single-agent planning method for this domain in a feasibility study. This planning method attempts to minimize the total number of block operations but is heuristic in nature, that is, is not guaranteed to achieve its objective (Kumar, Jung, and Koenig 2014). It is based on the idea of performing dynamic programming on a tree that spans the cells of the workspace

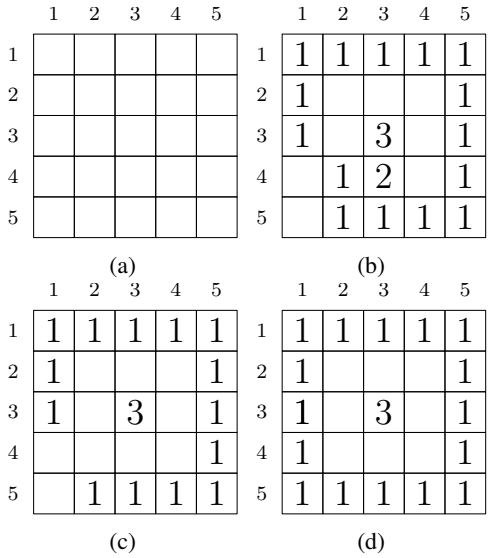


Figure 2: Shows an example of the phases of our planning algorithm. (a) shows the empty workspace matrix, which is the start configuration. (b) shows the workspace matrix after Phase 1, which adds blocks. A ramp for the tower and the tower itself have now been completely constructed, while the wall has been partially constructed. (c) shows the workspace matrix after Phase 2, which removes blocks. The ramp has now been deconstructed. (d) shows the workspace matrix after Phase 3 (goal configuration), which adds blocks. The remainder of the wall has now been constructed, finishing the goal configuration.

matrix and restricts the movements of the agent to the edges of this tree. The use of dynamic programming allows us to exploit common substructures and keep the number of block operations small. We then generalized it to a centralized multi-agent planning method (Cai et al. 2016). In the remainder of this section, we discuss both our single-agent planning method and its generalization.

Both planning methods operate on an undirected graph constructed from the workspace matrix. Each vertex corresponds to a cell, and each edge connects neighboring cells in the four compass directions. A special vertex S represents the reservoir and is connected to those vertices whose cells are neighbors of the reservoir, for example, the vertices whose cells are the boundary cells of the workspace matrix (since an agent carrying a block to or from the reservoir must cross the boundary of the workspace). The agents move on a spanning tree of this graph, rooted at S, as shown in Figure 3. They build a user-specified structure in phases. They add blocks to the structure in odd phases and remove blocks from the structure in even phases.

Our single-agent planning method uses dynamic programming on the spanning tree from its leaves to the root to first decide on the heights of the towers in each cell and then on the movements of the agent between the reservoir and the cells where blocks need to be added or removed. Consider, for example, a vertex in the spanning tree with two children

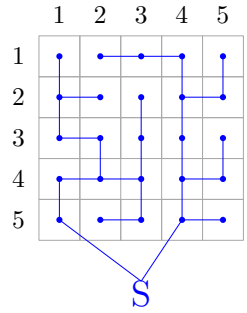


Figure 3: Shows the spanning tree used for the example.

during a phase where blocks are added to the structure. If dynamic programming has decided that towers of heights three and five are needed in the cells that correspond to the first and second child (respectively) of the vertex, then it can deduce that a tower of height four is needed in the cell that corresponds to the vertex itself since this height is necessary for an agent to put down the block that builds the tower of height five in the cell that corresponds to the second child. Details are given in (Kumar, Jung, and Koenig 2014), and an example is shown in Figure 2.

Our single-agent planning method is correct, is complete, runs in polynomial time and performs a polynomial number of block operations for any user-specified structure. It yields, within seconds, construction plans for problem instances with many blocks that require only 50-60 percent of block operations compared to the tower-by-tower planning method. However, our planning method can be improved. For example, its plan quality crucially depends on the spanning tree since the movements of the agent are restricted to the edges of the spanning tree, which can result in longer paths for the agent than necessary. One can find a good spanning tree for a given goal configuration in at least two ways: First, one can use heuristics for choosing good spanning trees. For example, minimum spanning trees on edge-weighted graphs can reduce the total number of block operations if the weight of an edge is the absolute value of the difference between the heights of the cells in the goal configuration that corresponds to the vertices it connects. (All edges neighboring S have weight zero.) Intuitively, a minimum spanning tree for this edge-weighted graph finds paths with minimum height variations in the goal configuration. Second, local search methods in an outer loop of our planning method can improve the initial spanning tree further over several iterations. Details are given in (Cai et al. 2016).

Our single-agent planning method can easily be generalized to multi-agent planning. Spanning trees allow for some parallelism in movements since the block operations carried out in the cells that correspond to the vertices of one subtree do not affect the operations for another subtree. For agents moving outside of their subtrees, one can exploit that multi-agent path finding is easier on trees and with identical agents (Yu and LaValle 2012). One simple technique to simplify coordination and avoid deadlocks is, for example, to move the agents in phases. All agents move along the tree

from the reservoir during odd phases and to the reservoir during even phases. Details are given in (Cai et al. 2016).

Conclusions

In this paper, we made a case for using collaborative construction with agents that correspond to the Harvard TERMES robots as testbed for cooperative multi-agent planning.

We have developed a first multi-agent planning method for this domain in a feasibility study. While elegant, it has a number of disadvantages. For example, it is domain-dependent, centralized, non-optimal, attempts to minimize the total number of block operations (rather than, say, the makespan, which is a more natural objective) and restricts the movements of the agents to the edges of the spanning tree, which can result in longer paths for the agents than necessary. The spanning tree also limits the amount of achievable parallelism for multiple agents. Agents can be assigned to different subtrees and then operate independently within their subtrees but still need to coordinate outside of their subtrees to avoid obstructing each other, for example, when picking up blocks from the reservoir. Finally, single agents do not have to carry blocks all the way to their destinations since agents can hand over blocks to each other, which our multi-agent planning method does not consider. Domain-independent, distributed and/or optimal planning raises a number of additional research issues that have not been addressed yet.

A student project at Ben-Gurion University of the Negev (Israel) recently encoded collaborative construction with the TERMES robots in a MA-PDDL format (Kovacs 2012) and used two existing (domain-independent) MA-STRIPS planners from the 2015 CoDMAP competition on toy instances that are much smaller than those that our (domain-dependent) planning method has been applied to (Yogev and Segal 2016). We predict that planning methods can be developed that are more efficient and effective than the ones existing one so far. Many different domain-dependent and domain-independent planning methods with different advantages and disadvantages are imaginable, such as learning a "ramp construction" macro to shorten the plan lengths and make planning more efficient. Smart macros are needed in this case since a crucial component of planning is to figure out how to fit ramps into the available space and how to amortize their construction and deconstruction effort among several towers that need to be built.

References

Cai, T.; Zhang, D.; Kumar, S.; Koenig, S.; and Ayanian, N. 2016. Local search on trees and a framework for automated construction using multiple identical robots [short paper]. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1301–1302.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Grushin, A., and Reggia, J. 2008. Automated design of distributed control rules for the self-assembly of prespecified artificial structures. *Robotics and Autonomous Systems* 56(4):334–359.

Jones, C., and Mataric, M. 2004. Automatic synthesis of communication-based coordinated multi-robot systems. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 381–387.

Kovacs, D. 2012. Multi-agent extension of PDDL3.1. In *Proceedings of the ICAPS Workshop on the International Planning Competition*, 19–27.

Kumar, S.; Jung, S.; and Koenig, S. 2014. A tree-based algorithm for construction robots. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

McDermott, D. 2000. The 1998 AI planning systems competition. *Artificial Intelligence Magazine* 21(2):35–55.

Napp, N., and Klavins, E. 2010. Robust by composition: Programs for multi-robot systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2459–2466.

Petersen, K.; Nagpal, R.; and Werfel, J. 2011. TERMES: An autonomous robotic system for three-dimensional collective construction. In *Proceedings of the International Conference on Robotics: Science and Systems (RSS)*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Werfel, J.; Petersen, K.; and Nagpal, R. 2011. Distributed multi-robot algorithms for the TERMES 3D collective construction system. In *Proceedings of the Workshop on Reconfigurable Modular Robotics at the IEEE International Conference on Intelligent Robots and Systems (IROS)*.

Werfel, J.; Petersen, K.; and Nagpal, R. 2014. Designing collective behavior in a termite-inspired robot construction team. *Science* 343(6172).

Wilt, C., and Botea, A. 2014. Spatially distributed multiagent path planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Wurman, P.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29(1):9–20.

Yogev, E., and Segal, A. 2016. A new problem domain for Classical MAS: 3D construction. Technical report, Department of Information System Engineering, Ben-Gurion University of the Negev. Supervisor: R. Stern.

Yu, J., and LaValle, S. 2012. Multi-agent path planning and network flow. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR)*, 157–173.

Yun, S., and Rus, D. 2010. Adaptation to robot failures and shape change in decentralized construction. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2451–2458.

Optimizing Electric Vehicle Charging Through Determinization

Sandhya Saisubramanian and **Shlomo Zilberstein** and **Prashant Shenoy**
College of Information and Computer Sciences
University of Massachusetts Amherst
Amherst, MA 01003, USA
{saisubramanian, shlomo, shenoy}@cs.umass.edu

Abstract

We propose a determinization based approach to optimize the charging policies of an electric vehicle (EV) operating in a *vehicle-to-grid* (V2G) setting. By planning when to charge or discharge electricity from the vehicle, the long-term cost of operating the EV can be minimized, while being consistent with the owner's preferences. For an EV operating under price uncertainty caused by the dynamic pricing of electricity, this problem needs to be solved on-the-fly. Therefore, we model this problem as a *Stochastic Shortest Path* (SSP) problem and employ a determinization technique to solve it. Since it is hard to predict *a priori* the performance of a determinization method on a given problem, we introduce the notion of *Lossless Determinization* (LLD) that produces optimal action selection via determinization and present an approach that achieves lossless determinization by adjusting the cost of actions to account for the ignored outcomes. We also present *Approximate Lossless Determinization* (ALLD)—an effective method for approximating the cost of actions based on state features. We evaluate the performance of ALLD and demonstrate its effectiveness on a range of settings for the electric vehicle charging problem.

Introduction

Electric vehicles function primarily as consumers of electricity from the grid. However, recent developments in cyber-physical systems allow electric vehicles to act as both consumers and producers of electricity when connected to a smart grid. Specifically in the *Vehicle-to-Grid* (V2G) setting, connections are added to electric vehicles to allow the flow of electricity from the vehicles to the smart grid, thus enabling electric vehicles to act as consumers and producers of electricity (Guille and Gross 2009; Kempton and Letendre 1997).

The efficiency of an electric vehicle largely depends on its efficient battery charging schedule. Donadee and Ilic model the EV charging problem under price uncertainty as an MDP with continuous space of decision variables and solve it using stochastic dynamic programming. The price uncertainty is modeled using a Gaussian copula (Donadee and Ilic 2014). Donadee, Ilic, and Karabasoglu model the EV charging problem operating under price uncertainty with stochastic driver behavior as an infinite horizon average reward MDP. The price uncertainty is modeled using a Gaussian copula and the MDP is solved offline using the value iteration algorithm (Donadee, Ilic, and Karabasoglu 2014). Ruelens et al. consider the stochasticity in the arrival and departure time for a fleet of Plug-in Hybrid Vehicles (PHEVs) and optimize the charging schedule for the fleet, using approximate policy iteration to minimize the cost (Ruelens et al. 2012). Shi and Wong optimize the charging policies of an EV operating under price uncertainty using Q-learning technique (Shi and Wong 2011). Most researchers have focused on devising policies for EV charging in the traditional setting only (Donadee, Ilic, and Karabasoglu 2014; Sortomme and El-Sharkawi 2011; Vayá and Andersson 2012; Donadee and Ilic 2014; Ruelens et al. 2012). Since electric vehicles can charge and discharge electricity in a V2G setting, it is possible to exploit this feature to further minimize the long-term costs associated with battery charging (Ma et al. 2012).

Hence, our objective is to optimize the charging schedule for an electric vehicle that is parked and connected to a smart grid in a V2G setting. By planning when to buy or sell electricity, the EV can devise a robust schedule for charging and discharging that is consistent with the owner's preferences, while minimizing the long-term cost of operating the vehicle. This problem needs to be solved quickly and on-the-fly due to price uncertainty caused by the dynamic pricing of electricity. Hence, we model it as a *Stochastic Shortest Path* (SSP) problem.

Solving large SSPs is an active research area in automated planning. Among the different techniques for solving SSPs that have been explored, *determinization* has attracted significant interest because it greatly simplifies the problem and can quickly solve large SSPs on-the-fly. Determinization ignores the stochastic transitions, leverages efficient off-the-shelf classical planners to solve the corresponding deterministic problem, and uses online replanning when an unexpected state is encountered. Since the policies for the EV charging problem needs to be obtained on-the-fly, we solve the EV charging SSP using determinization.

While determinization could be extremely effective, it is often hard to predict when it will work particularly well as policies produced via existing determinization techniques do not guarantee bounded-optimal performance. Since the value of the deterministic policy is a loose lower bound on the optimal value of the SSP (i.e., the expected cost of reaching the goal), a large difference between the values may be indicative of the deviation of the optimal policy for the deterministic problem from the optimal policy for the SSP. We call this difference the *loss*. For example, in the EV charging

problem, a suboptimal policy for the SSP yielded by determinization could be very expensive for the owner or could even lead to battery depletion at an unfavorable time. Therefore, it is beneficial to minimize the loss. When the loss is zero, it means that action values according to the deterministic policy match action values according to the optimal solution of the SSP, allowing for an easy derivation of the optimal actions. We examine the conditions under which this can be achieved and, more broadly, how to minimize the loss and thereby derive better policies.

To this end, our contributions in this paper are as follows,

1. We model the EV charging problem in a V2G setting as an SSP and consider different cost function scenarios;
2. We present the notion of *Lossless Determinization* (LLD) that requires the loss to be zero, and an approach called *Cost Adjustment for Lossless Determinization* (CALLD) that achieves zero loss by altering the costs of actions to account for the cost of ignored outcomes;
3. Naturally, it is challenging to accurately estimate the value discrepancy associated with each outcome and adjust the cost of each action without solving the original SSP. Hence, we propose an approximation technique to adjust the cost of actions in each state without calculating the true value of the outcomes. Specifically, in a factored MDP, the states are represented by feature vectors. The *Approximate Lossless Determinization* (ALLD) exploits the feature vector to derive an approximate cost for every action in a state;
4. We test the performance of ALLD on a range of settings for the EV charging problem.

We begin with a description of the model of EV charging as an SSP in Section 2. Section 3 defines determinization of an SSP and the notion of lossless determinization, and explains the CALLD algorithm for achieving zero loss. In Section 4 we describe our approach to approximating cost adjustments. Section 5 summarizes the performance of our approach in different settings of the EV charging domain.

The Model

By modeling the EV charging problem as an SSP, we aim to derive a sequence of actions that would minimize the long-term operational cost for an electric vehicle that is parked in a parking lot (parked and connected to a smart grid). On average, vehicles are parked for about 96% of the time (Kempton and Tomić 2005). Therefore, we restrict the decision process to the duration for which the vehicle is parked and connected to a smart grid. We begin with a formal background description of an SSP followed by a detailed explanation of modeling EV charging problem as an SSP.

Stochastic Shortest Path (SSP)

An SSP is a *Markov Decision Process* (MDP) with a start state and goal or terminal states, where the objective is to find a policy that minimizes the expected cost of reaching a goal state from the start state. A Stochastic Shortest Path MDP is denoted by the tuple $M = \langle S, A, T, C, s_0, S_G \rangle$, where,

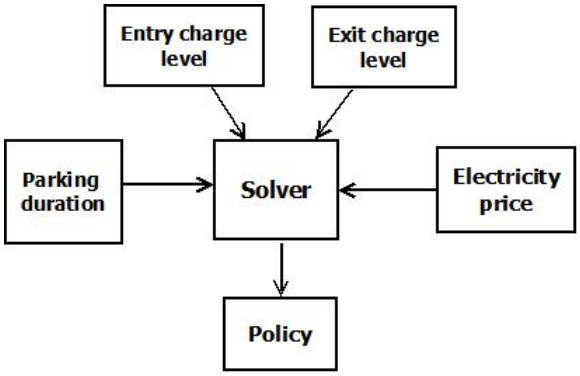


Figure 1: An illustration of EV charging problem

- S is a finite set of states;
- A is a finite set of actions with A_s denoting the set of actions available in state $s \in S$;
- T : $S \times A \times S \rightarrow [0, 1]$ is the transition function specifying the probability of moving to a state s' by executing an action $a \in A_s$ in state $s \in S$, denoted by $T(s, a, s')$;
- C : $S \times A \rightarrow \mathbb{R}^+ \cup \{0\}$ is the cost of executing action a in state s, denoted by $C(s, a)$. The cost of an action is positive in all states except goal states, where it is zero;
- s_0 is the initial state of the SSP, $s_0 \in S$; and
- S_G is the set of goal states of the SSP, $S_G \subseteq S$.

The solution of an SSP is a policy $\pi : S \rightarrow A$ that minimizes the expected cost of reaching a goal state. The Bellman equation defines a value function over states, $V^*(s)$, from which the optimal policy π^* can be extracted by:

$$V^*(s) = \min_a \; Q^*(s, a) \qquad \forall s \qquad (1)$$

$$Q^*(s, a) = C(s, a) + \sum_{s'} T(s, a, s') V^*(s') \quad \forall s, a \qquad (2)$$

where $Q^*(s, a)$ denotes the optimal Q-value of the action a in state s in the SSP M.

Modeling EV Charging Problem as an SSP

Since the decision process is restricted to the duration of parking for the electric vehicle, we model the EV charging problem as a finite horizon SSP with the parking duration as the horizon H. We assume that the vehicle can charge to a maximum limit (l_{max}) which is either the battery capacity (B_c) of the vehicle or some maximum threshold set by the vehicle owner, $0 < l_{max} \leq B_c$. Since we consider the electric vehicle in a V2G setting, we assume that the vehicle can discharge energy up to a minimum threshold level (l_g) which is either zero or some threshold set by the vehicle owner, $0 \leq l_g \leq B_c$. Assuming the parameters – l_{max}, l_g, H – are known, we can model this problem as an SSP with:

- S is the finite set of states that an electric vehicle can be in. It is defined by the tuple $\langle l, t, d, p \rangle$,where l denotes the current level of charge of the vehicle, $l \in [0, l_{max}]$, $t \in H$ denotes the current timestep, d denotes the current demand level for electricity, and p denotes the price distribution of electricity.

- A is the set of actions available to the vehicle. The vehicle can charge (Ch_i^+) and discharge (Ch_i^-) at three different speed levels, where i denotes the speed level, or remain idle (NOP). Therefore, there are seven actions in total, $A = \{Ch_1^+, Ch_2^+, Ch_3^+, Ch_1^-, Ch_2^-, Ch_3^-, NOP\}$. A_s denotes the set of actions available to the vehicle in state s. The charging and the discharging actions are stochastic, while the NOP action is deterministic.
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function denoted by $Pr(s'|s, a)$. It denotes the probability of reaching state s' by executing action a in state s. The state transition function also accounts for the demand level transitions and the pricing distribution transitions, as each state encapsulates the current demand level and current pricing distribution.
- $C : S \times A \rightarrow \mathbb{R}^+ \cup \{0\}$ is the cost function denoted by $C(s, a, s')$. It denotes the cost of executing action a in state s and reaching state s'. The costs for charging and discharging depend on the electricity pricing, and the speed setting. The cost for a NOP action is a constant. Based on real-world, the cost for discharging is modeled as a negative value (since the user profits by selling electricity), the cost for charging is modeled as a positive value (since the user has to pay for charging), and the cost for NOP action is modeled as zero.
- $s_0 \in S$ is the start state. It it defined by the tuple $s_0 = \langle l_0, t_0, d, p\rangle$, where $l_0 \in [0, l_{max}]$, and t_0 denote the charge level of the vehicle and the time when the vehicle is parked, respectively. t_0 also denotes the beginning of the decision process. d denotes the demand level at time t_0 and p denotes the price distribution at time t_0.
- $S_G \subseteq S$ is the set of goal states. It is denoted by the set of all states that match the tuple $\langle l, t_g, d, p\rangle$, where $l_g \leq l \leq l_{max}$, and t_g denotes the end of decision process when the vehicle is unplugged from the smart grid. d denotes the demand level at time t_g and p denotes the price distribution at time t_g.

The objective is to find a cost minimizing policy $\pi^* : S \rightarrow A$ that maximizes goal reachability, given the schedule of the vehicle owner. Figure 1 illustrates the EV charging problem.

Modeling Price Uncertainty

We consider four different types of cost functions that model the price uncertainty in a progressively more realistic way.

- Case 1 : The cost of discharging is the negation of the cost of charging and the costs are assumed to be known in advance. $C(s, Ch_i^-, s') = -C(s, Ch_i^+, s'), \forall i, \forall s, s' \in S$, where i denotes the speed level for charging or discharging.
- Case 2: The cost of discharging is the negation of the cost of charging plus a constant k. We again assume that the costs are known in advance. $C(s, Ch_i^-, s') = -C(s, Ch_i^+, s') + k, k \geq 0, \forall i, \forall s, s' \in S$, where i denotes the speed levels for charging and discharging. When $k = 0$, this case is the same as the previous case.
- Case 3: We assume that the cost of charging is known in advance, but the cost of discharging varies dynamically based on the actual level of demand. We assume that there is a known distribution for the demand level fluctuation based on the time of the day, with $P(d'|d, t)$ denoting the probability of demand d' at time t if the demand at time $t-1$ was d. $P(C(s, Ch_i^-, s') = r|t, d)$ denotes the probability of the discharge cost r for state s, given the demand d at time t.
- Case 4: We assume that the cost of charging is known in advance, but the cost of discharging varies dynamically based on the pricing distribution, which in turn depends on the current demand level. We assume that there is a known distribution for the demand level and for the pricing. For a price distribution p, $P(p|t, d)$ denotes the probability of the pricing distribution p at time t with demand level d, and $P(d'|d, t)$ denotes the probability of demand d' at time t if the demand at time $t - 1$ was d. Given the demand d and the pricing distribution p, $P(C(s, Ch_i^-, s') = r|t, d, p)$ denotes the probability of the discharge cost r for state s.

Generally SSPs are defined with non-negative costs as this would avoid any negative cost cycles. In a finite horizon SSP, negative costs cycles cannot be formed because it is not possible to transition to a state with a lower or equal time step from the current state. Since the EV charging problem described in this paper is modeled as a finite horizon SSP, the negative costs in the model would not lead to negative cost cycles.

Determinization of SSPs

A *determinization* yields a simplified variation of the SSP, with deterministic transition function that can be solved quickly using an off-the-shelf solver. Interest in determinization increased after the success of *FF-Replan* (Yoon, Fern, and Givan 2007) which won the 2004 IPPC, using the *Fast Forward* (FF) technique to generate fast deterministic plans (Hoffmann 2001). FF-Replan generates a deterministic version of the problem and solves it using FF. If an unexpected state is reached during plan execution, the process repeats with the unexpected state as the initial state, until a goal state is reached.

Following the success of FF-Replan, researchers have proposed various methods to improve determinization. Specifically, *Robust FF* (RFF) reduces the frequency of replanning by generating a plan for an envelope of states such that the probability of reaching a state outside the envelope is below some predefined threshold (Teichteil-Königsbuch, Kuter, and Infantes 2010). *HMDPP* generates plans with low probability of deviations using self-loop determinization and using a pattern database to avoid dead ends (Keyder and Geffner 2008). *FF-hindsight* uses hindsight optimization to approximate the value function of the MDP by sampling multiple deterministic futures that are solved using FF. These efforts resulted in a rich collection of determinization-based planning algorithms (Kolobov, Mausam, and Weld 2009; Yoon et al. 2008; 2010; Issakkimuthu et al. 2015; Keller and Eyerich 2011; 2012).

In the deterministic version of the SSP M, the start state s_0 and the goal states S_G are unaltered. Hence, the deterministic version M_d of the SSP M is denoted by the tuple $M_d = \langle S, A_d, T_d, C_d, s_0, S_G \rangle$, where, A_d is the finite set of actions (in this paper, $A_d = A$), $T_d : S \times A \to S$ denotes the deterministic transition function of M_d and $C_d : S \times A \to \mathbb{R}^+ \cup \{0\}$ specifies the cost function of the deterministic problem M_d. Conventional determinization techniques do not alter the cost function.

The optimal Q-value of M_d is computed as follows:

$$Q^*_d(s,a) = C_d(s,a) + V^*_d(T_d(s,a)) \quad \forall s \in S, a \in A_d. \tag{3}$$

In general, a determinization may introduce dead ends by ignoring an outcome that is crucial for goal reachability, making the goal unreachable in some states. However, it is possible to derive determinizations that preserve the goal reachability (for example, by using heuristics to devise the deterministic transition function that preserves goal reachability).

We define the loss of a determinization, l_d, as the *maximum* difference between the optimal Q-value of actions in the SSP M and the optimal Q-values in the determinized problem M_d.

Definition 1. *The **loss** associated with a determinization M_d of an SSP M is $l_d = \max_{s,a} |Q^*(s,a) - Q^*_d(s,a)|$.*

This difference is treated as a loss as it could potentially cause the model to yield a policy that significantly deviates from the optimal policy. As this could affect the cost of the plan and goal reachability in many problems, it is beneficial to minimize the loss. However, conventional determinization techniques may have arbitrary non-zero loss, $l_d > 0$.

Lossless Determinization

In this section, we describe a simple technique to modify the cost function of M_d such that the loss is provably zero.

Definition 2. *A determinization is a **lossless determinization** (LLD) when the corresponding loss is zero: $l_d = 0$.*

Note that solving a lossless determinization of a given SSP guarantees that the selected actions are optimal for the SSP.

We can achieve zero loss, $l_d = 0$, by adjusting the cost of actions in each state. The costs are modified for every (s,a) pair to account for the values of the outcomes ignored by the determinization. Consequently, the optimal Q-values of M_d are equal to the optimal Q-values of M, reducing the loss to zero. The process of arriving at a lossless determinization by adjusting the cost of each action in every state is referred to as *Cost Adjustment for Lossless Determinization* (CALLD).

Algorithm 1 describes a cost modification technique that produces a lossless determinization. The input is the SSP and a deterministic transition function, and the output is the cost function for the determinized problem, C_d. Line 3 is the cost adjustment step, where $V^*(s')$ is the optimal value of the successor s' and $V^*(T_d(s,a))$ is the optimal value of the successor in the deterministic transition function in the SSP M. Since the cost adjustment in Algorithm 1 depends on the difference between outcome values, the costs yielded by CALLD algorithm may be negative. In general, in an infinite horizon SSP, negative costs may lead to negative cost cycles, affecting the validity of the SSP. Therefore, we discuss the necessary and sufficient conditions under which CALLD produces non-negative cost.

Algorithm 1: CALLD (M, T_d)

```
1 foreach s ∈ S do
2     foreach a ∈ A_s do
3         C_d(s,a) ← Σ_{s'} ( T(s,a,s') V*(s') ) +
            C(s,a) − V*(T_d(s,a)) ;
4     end
5 end
6 return C_d
```

Proposition 1. *The necessary and sufficient condition for having non-negative costs in a CALLD, $C_d(s,a) \geq 0$, is that the deterministic transition function chooses outcomes in M such that $Q^*(s,a) \geq V^*(T_d(s,a))$.*

Proof. We consider each one of the implication directions:

Case 1: $\Big(Q^*(s,a) \geq V^*(T_d(s,a))\Big) \implies \Big(C_d(s,a) \geq 0\Big)$

Assume $Q^*(s,a) \geq V^*(T_d(s,a))$. Using the definition of Q-values (Equation (2)), we get:

$$C(s,a) + \sum_{s'} \Big(T(s,a,s')V^*(s')\Big) - V^*(T_d(s,a)) \geq 0.$$

Substituting for $C_d(s,a)$ from Algorithm 1, we get $C_d(s,a) \geq 0$. Thus, $Q^*(s,a) \geq V^*(T_d(s,a))$ is a sufficient condition for non-negative cost in CALLD.

Case 2: $\Big(C_d(s,a) \geq 0\Big) \implies \Big(Q^*(s,a) \geq V^*(T_d(s,a))\Big)$

Assume $C_d(s,a) \geq 0$. Substituting for $C_d(s,a)$ from Algorithm 1,

$$C(s,a) + \sum_{s'} T(s,a,s')V^*(s') \geq V^*(T_d(s,a)).$$

Using the definition of Q-values, Equation (2), we get:

$$Q^*(s,a) \geq V^*(T_d(s,a)).$$

Hence, we conclude that the proposition holds. □

In the case of a finite horizon stochastic planning problem, negative cost cycles cannot be formed and therefore, satisfying the necessary and sufficient conditions for non-negative costs in a CALLD is non-mandatory.

Proposition 2. *CALLD produces a lossless determinization when a deterministic transition function that preserves the goal reachability is used.*

Proof. We need to show that given an SSP M and its determinization M_d that preserves the goal reachability,

the optimal Q-values in M_d are equal to the Q-values in M, $Q^*_d(s,a) = Q^*(s,a)$, if the cost function $C(s,a)$ is modified to account for the outcomes ignored during determinization.

Substituting for $C_d(s,a)$ from Algorithm 1 in Equation (3) and using Equation (2) we get

$$Q^*_d(s,a) = Q^*(s,a) - V^*(T_d(s,a)) + V^*_d(T_d(s,a)).$$

Since the cost adjustment is performed for every (s,a) pair and the determinization preserves goal reachability, $V^*(T_d(s,a)) = V^*_d(T_d(s,a))$ and hence $l_d = 0$. □

Corollary 1. *There exists a lossless determinization for every SSP.*

Since it is possible to derive a goal reachability preserving determinization for every SSP and using CALLD would produce a lossless determinization (Proposition 2), it is possible to arrive at a lossless determinization (Definition 2) for any SSP.

Approximate Lossless Determinization

In many real-world problems, it is challenging to derive a complete cost adjusted lossless determinization of the problem without solving the SSP and this defeats the purpose of determinization. Therefore, we propose an approximation technique, referred to as *Approximate Lossless Determinization* (ALLD). An ALLD estimates the cost for each action in a state for a determinization of the SSP. We consider sampling and machine learning techniques for estimating the cost adjustment for a large SSP, which we will refer to as our target problem for simplicity. In this paper, the approximate cost adjustments for the target problem are learned from sampled small problems using a feature-based cost function.

Definition 3. *A **feature-based cost function** estimates the cost of an action in a state using the features of the state,* $C_d(s,a) = g(\vec{f}(s), a)$.

In a factored MDP, a state s is characterized by a set of features and these can be used to predict the cost of an action in the state. Let $\vec{f}(s) = \langle f_1(s), ..., f_n(s) \rangle$ be a set of features in a state s that significantly affect the cost of actions. Such features can be identified using machine learning techniques such as regression.

In order to estimate the feature-based approximate cost, sample problems are generated and solved. The costs adjustment values for the samples are computed in hindsight. The samples are obtained either from known small problem instances in the target domain or generated automatically by sampling states from the target problem. If the target problem has unavoidable dead ends, then sampling states may not be a good representative of the target problem. In such cases, smaller problem instances from the domain can be used. In this paper, smaller problems are created by multiple trials of depth limited random walk on the target problems and solved using LAO* (Hansen and Zilberstein 2001). The cost adjustments are computed for the samples using their exact solutions and the feature-based costs are learned. The learned values are projected onto the target problem using the feature-based cost function.

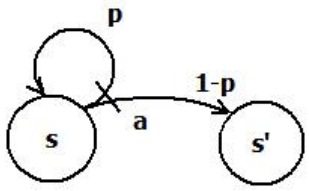


Figure 2: Example 1:Illustration

We also consider an extreme case, where the feature set characterizing each state is empty.

Definition 4. *A **state independent cost adjustment** assigns a* constant *cost adjustment per action, regardless of the state, resulting in a constant cost* $C_d(s,a) = g(a)$.

This simple form of generalization of the cost adjustment ignores the state altogether. In particular, PPDDL description of problems (Younes and Littman 2004) have a single description per action and hence having constant cost adjustment for actions in a problem instance can be extended to having constant cost adjustment for those actions in the various instances of problems in the domain.

For example, consider an action a and the set of states in which a is applicable. If the relative discrepancy between the values of the outcomes of a is the same in every state and the cost of a, $C(s,a)$, is the same in every state, then the cost adjustment can be trivially generalized with a state independent cost adjustment.

Although it is challenging to balance the trade-off between using state independent costs and solution quality, the following examples suggest that state independent cost adjustment could be effective.

Example 1. *Consider an SSP in which an action can achieve a successful outcome with probability* $1-p$ *or fail with probability* $p>0$. *When an action fails, the state remains unchanged. Let* s *denote a state of the SSP for which a successful execution of action* a *with cost* $C(s,a)$ *results in outcome state* s'. *Figure 2 is an illustration of this example.*

Proposition 3. *State independent cost adjustment produces zero loss for the class of problems identified in Example 1.*

Proof. In a goal reachability preserving determinization of the problem, the failure outcome would be ignored and the cost, $C_d(s,a)$, calculated by Algorithm 1 is,

$$C_d(s,a) = C(s,a) + \sum_{s'} \Big(T(s,a,s') V^*(s') \Big) - V^*(T_d(s,a)).$$

Since a fails with a probability p, we get

$$C(s,a) + \sum_{s'} \Big(T(s,a,s') V^*(s') \Big) \;=\; \frac{C(s,a)}{1-p} + V^*(T_d(s,a)).$$

Instance# ($\|S\|, \|A\|$)	%Charge (entry,exit)	Optimal Cost	Cost(Greedy)	Cost(MLO)	Cost(ALLD)	%DE (Greedy)	%DE(MLO)	%DE (ALLD)
P1 (909,7)	(80,20)	-7.60	-1.98 $\pm$ 0.07	-5.47 $\pm$ 0.02	-5.55 $\pm$ 0.01	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P2 (909,7)	(60,20)	-6.15	-1.93 $\pm$ 0.07	-4.42 $\pm$ 0.08	-4.52 $\pm$ 0.06	0 $\pm$ 0	21.40 $\pm$ 0.04	0 $\pm$ 0
P3 (909,7)	(30,40)	1.12	2.53 $\pm$ 0.07	1.86 $\pm$ 0.02	1.59 $\pm$ 0.01	0 $\pm$ 0	20.00 $\pm$ 0.20	16.00 $\pm$ 0.3
P4 (909,7)	(50,50)	-0.31	0 $\pm$ 0.01	0.09 $\pm$ 0.03	-0.62 $\pm$ 0.02	0 $\pm$ 0	20.00 $\pm$ 0.01	3.00 $\pm$ 0.02
P5 (909,7)	(30,60)	3.58	4.38 $\pm$ 0.05	4.12 $\pm$ 0.03	3.58 $\pm$ 0.03	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P6 (909,7)	(20,60)	4.12	5.58 $\pm$ 0.07	4.56 $\pm$ 0.03	4.33 $\pm$ 0.05	0 $\pm$ 0	23.05 $\pm$ 0.20	12.00 $\pm$ 0.3
P7 (909,7)	(40,90)	4.36	4.93 $\pm$ 0.06	4.86 $\pm$ 0.03	4.42 $\pm$ 0.05	0 $\pm$ 0	20.00 $\pm$ 0.02	15.00 $\pm$ 0.03

Table 1: Plan quality for seven instances of Electric Vehicle Charging- Cost Case 1

Instance# ($\|S\|, \|A\|$)	%Charge (entry,exit)	Optimal Cost	Cost(Greedy)	Cost(MLO)	Cost(ALLD)	%DE (Greedy)	%DE(MLO)	%DE (ALLD)
P1 (909,7)	(80,20)	-4.87	-1.55 $\pm$ 0.04	-2.72 $\pm$ 0.02	-3.17 $\pm$ 0.01	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P2 (909,7)	(60,20)	-3.97	-1.52 $\pm$ 0.05	-2.5 $\pm$ 0.02	-2.62 $\pm$ 0.02	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P3 (909,7)	(30,40)	1.16	3.06 $\pm$ 0.04	1.92 $\pm$ 0.06	1.54 $\pm$ 0.02	0 $\pm$ 0	1.00 $\pm$ 0.03	0.50 $\pm$ 0.03
P4 (909,7)	(50,50)	0.00	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P5 (909,7)	(30,60)	4.26	5.57 $\pm$ 0.08	4.63 $\pm$ 0.02	4.29 $\pm$ 0.02	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P6 (909,7)	(20,60)	4.44	5.54 $\pm$ 0.08	4.55 $\pm$ 0.09	4.46 $\pm$ 0.01	0 $\pm$ 0	3.00 $\pm$ 0.02	0.50 $\pm$ 0.02
P7 (909,7)	(40,90)	4.68	5.25 $\pm$ 0.03	4.85 $\pm$ 0.02	4.74 $\pm$ 0.02	0 $\pm$ 0	16.00 $\pm$ 0.02	1.10 $\pm$ 0.02

Table 2: Plan quality for seven instances of Electric Vehicle Charging- Cost Case 2

Substituting the above equation in the first equation, we get

$$\begin{aligned} C_d(s,a) &= \frac{C(s,a)}{1-p} + V^*(T_d(s,a)) - V^*(T_d(s,a)) \\ &= \frac{C(s,a)}{1-p}. \end{aligned} \tag{4}$$

Thus, the proposition illustrates a class of problems for which state independent cost (Equation 4) is perfectly accurate with *zero loss*. □

The above proposition highlights the potential benefits of ALLD that can achieve a perfectly accurate cost adjustment for problems such as the Blocksworld, that satisfies the required conditions. In the Blocksworld problem—an IPPC problem with stochastic actions— given an initial configuration of a collection of blocks, the blocks need to be rearranged to satisfy some goal conditions. Since the actions are stochastic, an action, for example, "pick block" may be successful or unsuccessful. If unsuccessful, the block slips and is dropped on the table and the action is repeated until it is successful. Since the relative discrepancy in the values of the outcomes is constant, a state independent cost adjustment that is accurate with zero loss is feasible. Consider the setting with unit cost actions that fail with a probability of 0.25. Our experiments show that regardless of the specific block, the state independent cost for this action is constant that matches the value of 1.33 obtained using Equation 4. This example illustrates the scope of generalized constant cost. However, not all domains satisfy this property. Identifying actions and domains that exhibit this property would alleviate the need for the preprocessing and help exploit the hidden structure in the given domain.

Experimental Results

The performance of ALLD is tested on four different cost function settings of EV. For each cost function, we consider seven different entry and exit level charges. In each cost function scenario, the charging costs are assumed to be known ahead of the decision process, and the costs associated with discharge actions may or may not be known ahead of time, depending on the cost function case. For the costs known in advance, we use the Time-of-Use (ToU) pricing (Eversource 2017). In case 3 of the cost function, we consider four demand levels– super off-peak, off-peak, mid-peak, and peak demand. In cost function case 4, we consider two pricing distributions – off-peak, and peak pricing levels, in addition to the four demand levels. The peak and the non-peak hours are based on real world peak hours and non-peak hours (Eversource 2017).

In all our experimental test cases, we consider an EV parked for a span of two hours and the duration of each timestep t is equivalent to 15 minutes in real time. The battery capacity and the three charge speed settings for the EV were selected based on Nissan Leaf EV configuration. We assume the discharge speeds to be the same as that of charge speeds. We also assume that the battery efficiency of the vehicle is not 100% and therefore, it may be required to buy more electricity from the grid than needed, and the electricity that reaches the grid during discharge would be lesser than the actual quantity discharged from the vehicle. We account for this battery inefficiency by adding a penalty of 15% to the current charging and discharging costs.

The goal in this domain is to devise a robust policy for EV charging such that the long-term operational cost of the vehicle is minimized, while being consistent with the

Instance# ($\|S\|, \|A\|$)	%Charge (entry,exit)	Optimal Cost	Cost(Greedy)	Cost(MLO)	Cost(ALLD)	%DE (Greedy)	%DE (MLO)	%DE (ALLD)
P1 (3636,7)	(80,20)	-4.60	-1.12 $\pm$ 0.02	-1.63 $\pm$ 0.04	-2.84 $\pm$ 0.03	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P2 (3636,7)	(60,20)	-2.60	-1.10 $\pm$ 0.01	-1.14 $\pm$ 0.02	-1.70 $\pm$ 0.01	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P3 (3636,7)	(30,40)	1.42	3.53 $\pm$ 0.03	1.62 $\pm$ 0.05	1.46 $\pm$ 0.02	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P4 (3636,7)	(50,50)	-0.94	0 $\pm$ 0	-0.02 $\pm$ 0.02	-0.32 $\pm$ 0.01	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P5 (3636,7)	(30,60)	2.49	4.20 $\pm$ 0.01	2.79 $\pm$ 0.02	2.49 $\pm$ 0.02	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P6 (3636,7)	(20,60)	3.28	4.52 $\pm$ 0.04	3.93 $\pm$ 0.02	3.72 $\pm$ 0.02	0 $\pm$ 0	5.00 $\pm$ 0.02	0 $\pm$ 0
P7 (3636,7)	(40,90)	3.47	4.86 $\pm$ 0.03	4.25 $\pm$ 0.04	3.70 $\pm$ 0.02	0 $\pm$ 0	15.00 $\pm$ 0.02	2.00 $\pm$ 0.02

Table 3: Plan quality for seven instances of Electric Vehicle Charging- Cost Case 3

Instance# ($\|S\|, \|A\|$)	%Charge (entry,exit)	Optimal Cost	Cost(Greedy)	Cost(MLO)	Cost(ALLD)	%DE (Greedy)	%DE (MLO)	%DE (ALLD)
P1 (7272,7)	(80,20)	-4.76	-3.22 $\pm$ 0.09	-3.30 $\pm$ 0.09	-3.52 $\pm$ 0.01	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P2 (7272,7)	(60,20)	-3.71	-3.08 $\pm$ 0.08	-3.21 $\pm$ 0.06	-3.38 $\pm$ 0.01	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P3 (7272,7)	(30,40)	1.10	1.78 $\pm$ 0.09	1.77 $\pm$ 0.03	1.34 $\pm$ 0.01	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P4 (7272,7)	(50,50)	-1.00	0 $\pm$ 0	-0.25 $\pm$ 0.04	-0.35 $\pm$ 0.02	0 $\pm$ 0	0 $\pm$ 0	0 $\pm$ 0
P5 (7272,7)	(30,60)	2.90	3.79 $\pm$ 0.04	3.65 $\pm$ 0.03	3.07 $\pm$ 0.01	0 $\pm$ 0	10.00 $\pm$ 0.02	3.00 $\pm$ 0.02
P6 (7272,7)	(20,60)	3.29	5.09 $\pm$ 0.02	4.73 $\pm$ 0.03	3.91 $\pm$ 0.01	0 $\pm$ 0	20.00 $\pm$ 0.02	7.00 $\pm$ 0.02
P7 (7272,7)	(40,90)	3.59	4.84 $\pm$ 0.04	3.97 $\pm$ 0.02	3.59 $\pm$ 0.02	0 $\pm$ 0	12.80 $\pm$ 0.04	0 $\pm$ 0

Table 4: Plan quality for seven instances of Electric Vehicle Charging- Cost Case 4

owner's preferences. Any state from where the goal (exit level charge) cannot be reached in the remaining duration of the parking is treated as a dead end in the SSP.

A feature-based cost function is used to estimate the cost in ALLD in all our experiments. While ALLD requires a preprocessing step, estimating the costs is only required once per domain. The scalability of ALLD is preserved as we limit the size of the required sampled problems; in our experiments a depth of 4-8 was sufficient in most cases. The quality of the plan generated by ALLD is compared with:

- Quality of the plan generated by solving the *Most Likely Outcome* determinization of the SSP (MLO),
- Quality of the plan generated by using a greedy heuristic based on a naive human decision making.
- Optimal cost obtained by solving the SSP offline.

The value of the plan (average cost of reaching the goal) and fraction of dead end visits (%DE) are used as metrics to estimate the quality of the generated plan. Standard errors are reported for the value of the plan and the dead end visits based on 1000 trials per setting.

In MLO determinization and ALLD, the deterministic transition function chooses the most likely outcome for an action in a state. Since both the techniques share the determinisitic transition function, we use an optimal solver to solve the determinisitic problems, and to efficiently evaluate and compare the performance of ALLD. Therefore, the MLO determinization and ALLD are solved using the A* algorithm (Hart, Nilsson, and Raphael 1968), which is maximally efficient, and are complemented by replanning when necessary. It is assumed that the time taken for replanning is negligible.

Greedy heuristic We model a naive, and risk-averse human decision making as a simple greedy heuristic. Since most people prefer to ensure that the vehicle achieves the predefined exit charge over the monetary profits, we consider this as a risk-averse decision making. It is also considered naive and greedy because the decision is based only on the current state of the system. If the current charge of the vehicle is equal to the predefined exit charge for the vehicle, then the heuristic policy is to do NOP. If the current charge level is less than the exit charge, then the heuristic policy is to charge the vehicle at the maximum charge speed. If the current charge level is greater than the required exit charge, then the heuristic policy for that state is to discharge the electricity in medium speed as that would ensure profit for the vehicle without draining the battery quickly. Using these heuristic guidelines, a greedy policy can be devised for the EV charging.

Discussion The performance of ALLD in the different cost function scenarios is discussed below. The costs in the tables account for the expenses related to charging and profits from discharging.

In case 1, we assume that the cost of discharging is the negation of the cost of charging. The result of the 1000 trials are tabulated in table 1. In most cases, ALLD performed better than MLO and greedy approach, in terms of cost. In case 2, we assume that the cost of discharging is the negation of the cost of charging plus some non-zero constant which depends on the time and is known ahead of the decision process. The result of the 1000 trials are tabulated in table 2. In all our test cases, ALLD performed better than MLO and greedy approach, in terms of cost, with significantly lower

dead end visits.

In case 3, we assume that the cost of charging is known in advance and is based on the Time-of-Use pricing. The discharging cost depends on the current demand level and we assume that the distribution is known. The result of the 1000 trials are tabulated in table 3. In most of our test cases, ALLD performed better than MLO and greedy approach, in terms of dead ends and cost. In case 4, we assume that the cost of charging is known in advance and is based on the Time-of-Use pricing. The discharging cost depends on the current demand level and the current pricing distribution. We assume that the demand and pricing distributions are known in advance. The result of the 1000 trials are tabulated in table 4. In all our test cases, ALLD performed better than MLO and greedy approach, in terms of cost, with significantly lower dead end visits.

For every test case in each case of the cost function discussed above, the greedy approach consistently avoids dead ends. This is an expected behavior as the greedy approach is conservative with respect to discharging electricity. However, the cost obtained by executing the greedy policy is much higher than the optimal, compared to the two determinization techniques. In cases where the entry charge is lower than the exit charge, the greedy policy is very expensive as it tries to charge as fast as possible to avoid dead end. However, this may be unnecessary if the parking duration is long enough. Also, the greedy policy does not consider the price variation which significantly affects the total cost.

Overall, while greedy approach is simple and easy, it is not effective when there is price variation with respect to time or in complicated settings such as the price depending on the current demand level. This reinforces the need for automated planning for EV charging under price uncertainty. ALLD always performs better than the greedy approach in terms of cost, and better than MLO determinization in terms of both cost and dead end visits, illustrating the potential of ALLD in achieving near-optimal policy for an EV operating under price uncertainty.

Conclusion and Future Work

Until recently, electric vehicles were primarily perceived as consumers of electricity from the smart grid. In the Vehicle-to-Grid (V2G) setting, electric vehicles can act as both consumers and producers of electricity. Using this feature, we aim to derive a charging policy for the EV that minimizes the long-term operational cost of the vehicle and that is consistent with the owner's preferences. Due to price uncertainty, this problem needs to be solved on-the-fly. Hence, we model this problem as a stochastic shortest path problem and employ determinization technique to solve it.

Since the policies yielded by conventional determinization techniques can significantly deviate from the optimal policy, we introduce the notion of lossless determinization that produces optimal action selection via determinization. We present cost adjustment for lossless determinization, an approach to achieve lossless determinization by adjusting the costs of actions in the deterministic problem. Since it is difficult to compute the exact cost adjustment without knowing the optimal values of the states, we propose approximation techniques to compute estimated costs. Our experiments show that ALLD can effectively use approximate costs to get better results than conventional determinization techniques.

The model presented in this paper aims to optimize the charging policies of an electric vehicle in a V2G setting. However, we do not consider the ancillary services like frequency regulation that an EV can offer. In future work, we plan to explore the role of determinization in general, and ALLD in particular, for other ancillary services of an EV in a V2G setting. While we assume that the duration of parking is known in advance, planning with stochastic parking duration is an interesting direction for the future work.

References

Donadee, J., and Ilic, M. D. 2014. Stochastic Optimization of Grid to Vehicle Frequency Regulation Capacity Bids. *IEEE Transactions on Smart Grid* 5(2):1061–1069.

Donadee, J.; Ilic, M.; and Karabasoglu, O. 2014. Optimal Autonomous Charging of Electric Vehicles with Stochastic Driver Behavior. In *Vehicle Power and Propulsion Conference (VPPC), 2014 IEEE*, 1–6. IEEE.

Eversource. 2017. Eversource Energy - Time of Use Rates. https://www.eversource.com/clp/vpp/vpp.aspx.

Guille, C., and Gross, G. 2009. A conceptual framework for the vehicle-to-grid (v2g) implementation. *Energy policy* 37(11):4379–4390.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence* 129(1):35–62.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Hoffmann, J. 2001. FF: The Fast-Forward Planning System. *AI Magazine* 22(3):57.

Issakkimuthu, M.; Fern, A.; Khardon, R.; Tadepalli, P.; and Xue, S. 2015. Hindsight Optimization for Probabilistic Planning with Factored Actions. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling*, 120–128.

Keller, T., and Eyerich, P. 2011. A Polynomial All Outcome Determinization for Probabilistic Planning. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, 331–334.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic planning based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*.

Kempton, W., and Letendre, S. E. 1997. Electric vehicles as a new power source for electric utilities. *Transportation Research Part D: Transport and Environment* 2(3):157–175.

Kempton, W., and Tomić, J. 2005. Vehicle-to-grid power fundamentals: Calculating capacity and net revenue. *Journal of power sources* 144(1):268–279.

Keyder, E., and Geffner, H. 2008. The HMDP Planner for Planning with Probabilities. In *Proceedings of the International Planning Competition (IPC 2008)*.

Kolobov, A.; Mausam; and Weld, D. S. 2009. ReTrASE: Integrating Paradigms for Approximate Probabilistic Planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 1746–1753.

Ma, Y.; Houghton, T.; Cruden, A.; and Infield, D. 2012. Modeling the Benefits of Vehicle-to-Grid Technology to a Power System. *IEEE Transactions on power systems* 27(2):1012–1020.

Ruelens, F.; Vandael, S.; Leterme, W.; Claessens, B. J.; Hommelberg, M.; Holvoet, T.; and Belmans, R. 2012. Demand Side Management of Electric Vehicles with Uncertainty on Arrival and Departure Times. In *Innovative Smart Grid Technologies (ISGT Europe), 2012 3rd IEEE PES International Conference and Exhibition on*, 1–8. IEEE.

Shi, W., and Wong, V. W. 2011. Real-Time Vehicle-to-Grid Control Algorithm under Price Uncertainty. In *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*, 261–266. IEEE.

Sortomme, E., and El-Sharkawi, M. A. 2011. Optimal charging strategies for unidirectional vehicle-to-grid. *IEEE Transactions on Smart Grid* 2(1):131–138.

Teichteil-Königsbuch, F.; Kuter, U.; and Infantes, G. 2010. Incremental Plan Aggregation for Generating Policies in MDPs. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, 1231–1238.

Vayá, M. G., and Andersson, G. 2012. Smart charging of plug-in vehicles under driving behaviour uncertainty. In *12th International Conference on Probabilistic Methods Applied to Power Systems*, 10–14.

Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic Planning via Determinization in Hindsight. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 1010–1016.

Yoon, S.; Ruml, W.; Benton, J.; and Do, M. B. 2010. Improving Determinization in Hindsight for On-line Probabilistic Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, 209–216.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, 352–359.

Younes, H. L. S., and Littman, M. L. 2004. PPDDL1.0: The Language for the Probabilistic Part of IPC-4. In *Proceedings of the International Planning Competition*.

Planning-based Scenario Generation for Enterprise Risk Management

Shirin Sohrabi and **Anton V. Riabov** and **Octavian Udrea**
IBM T.J. Watson Research Center
1101 Kitchawan Rd, Yorktown Heights, NY 10598, USA
{ssohrab, riabov, udrea}@us.ibm.com

Abstract

Scenario planning is a commonly used method that various organizations use to develop their long term plans. Scenario planning for risk management puts an added emphasis on identifying the extreme yet possible risks that are not usually considered in daily operations. While a variety of methods and tools have been proposed for this purpose, we show that formulating an AI planning problem, and applying AI planning techniques to develop the scenarios provides a unique advantage for scenario planning. Our system, the Scenario Planning Advisor (SPA), takes as input the relevant news and social media trends that characterize the current situation, where a subset of them is selected to represent key observations, as well as the domain knowledge. The domain knowledge is acquired using a graphical tool, and then automatically translated to a planning domain. We use a planner to generate multiple plans explaining the observations and projecting future states. The resulting plans are clustered and summarized to generate the scenarios for use in scenario planning. We discuss our knowledge engineering methodology, lessons learned, and the feedback received from the pilot deployment of the SPA system in a large international company. We also show our experiments that measure planning performance and how balanced and informative the generated scenarios are as we increase the complexity of the problem.

1 Introduction

Scenario planning is a commonly used method for strategic planning (Schoemaker 1995). Scenario planning involves analyzing the relationship between forces such as social, technical, economic, environmental, and political trends in order to explain the current situation in addition to providing insights about the future. A major benefit to scenario planning is that it helps businesses or policy-makers learn about the possible alternative futures and anticipate them. While the expected scenarios are interesting for verification purposes, scenarios that are surprising to the users (e.g., policy-makers businesses) are the ones that are the most important and significant (Peterson *et al.* 2003).

Risk management is a set of principles that focus on the outcome for risk-taking (Stulz 1996). A variety of methods and standards for risk management under different assumptions have been developed (Avanesov 2009). In this paper, we address scenario planning for risk management, the problem of generating scenarios with a significant focus on identifying the extreme yet possible risks that are not usually considered in daily operations. The approach we take in this paper is different from previous work in that we reason about emerging risks based on observations from the news and social media trends, and produce scenarios that both describe the current situation and project the future possible effects of these observations. Our objective is not to find a precise answer, that is to predict or forecast, but rather to project the possible alternative scenarios that may need consideration. Each scenario we produce highlights the potential *leading indicators*, the set of facts that are likely to lead to a scenario, the *scenario and emerging risk*, the combined set of consequences or effects in that scenario, and the *business implications*, a subset of potential effects of that scenario that the users (e.g., policy-makers, businesses) care about. The business implications are akin to the set of possible goals.

For example, given a high inflation observation, economic decline followed by a decrease in government spending can be the consequences or the possible effects in a scenario, while decreased client investment in the company offerings is an example of a business implication (i.e., the resulting goal). Furthermore, an increase in the cost of transportation could have been the leading indicator for that scenario. To the best of our knowledge, we are the first to apply AI planning in addressing scenario planning for enterprise risk management. We believe that AI planning provides a very natural formulation for the efficient exploration of possible outcomes required for scenario planning.

In this paper, we propose to view the scenario planning problem for enterprise risk management as a problem that can be translated to an AI planning problem. An intermediate step is a plan recognition problem, where the set of given business implications forms the set of possible goals, and the observations are selected from the news and social media trends. The domain knowledge is acquired from the domain expert via a graphical tool and is then automatically translated to an AI planing domain. AI planning is in turn used to address the plan recognition problem (Ramírez and Geffner 2009; Sohrabi *et al.* 2016a; 2017). Top-k planning or finding a set of high-quality plans is used to generate multiple plans that can be grouped into a scenario (Riabov *et al.* 2014; Sohrabi *et al.* 2016b). The set of plans is then clustered and summarized to generate the scenarios. Hence, each scenario is a collection of plans that explain the observations and con-

siders the possible cascading effects of the actions to identify potential future outcomes.

2 System Architecture

The system architecture for our system, Scenario Planning Adviser (SPA), is shown in Figure 1. There are three major components. The planning engine, shown under the *Scenario Generation and Presentation* component, takes as input the output of the other two components: the *News Aggregation* component and the *Domain Knowledge* component. The *News Aggregation* component deals with analyzing the raw data coming from the news and social media feeds. To this end, several text analytics are implemented in order to find the information that is relevant for a particular domain as filtered by the provided Topic Model. The Topic Model, provided by the domain expert, includes the list of important people, organization, and keywords. The result of the *News Aggregation* component is a set of relevant key observations, a subset of which can be selected by the business user and is fed into the *Scenario Generation* component. The *Domain Knowledge* component captures the necessary domain knowledge in two forms, Forces Model and Forces Impact. The Forces Model is a description of the causes and effects for a certain force, such as social, technical, economic, environmental, and political trends, and is provided by a domain expert who have little or no AI planning background. Forces Model are captured by a Mind Map (`http://freemind.sourceforge.net/wiki/`), a graphical tool that encodes concepts and relations. An example of a Mind Map for the currency depreciation force is shown in Figure 3. The Forces Impact, describes potential likelihoods and impact of a cause (i.e., concepts with an edge going into the main force) or an effect (e.g., concepts with an edge going from the main force and all other cascading concepts). The *Scenario Generation* component takes the domain knowledge and the key observations and automatically generates a planning problem whose outcome when clustered in the post-processing step generates a set of alternative scenarios.

Our system is currently deployed for an international organization. We use a company name Acme, for anonymity, in our examples. The system generates thousand plans and presents three to six scenarios to the business user. The extensive feedback we have collected has been encouraging and helpful in improving our system. We report on our knowledge engineering efforts, collected feedback, and the lessons learned in the rest of this paper.

3 Problem Definition

In this section, we briefly review necessary background on AI planning and Plan Recognition before defining the scenario planning for risk management problem.

Definition 1 *A planning problem is a tuple* $P = (F, A, I, G)$*, where* F *is a finite set of fluent symbols,* A *is a set of actions with preconditions,* $\text{PRE}(a)$*, add effects,* $\text{ADD}(a)$*, delete effects,* $\text{DEL}(a)$*, and action costs,* $\text{COST}(a)$*,* $I \subseteq F$ *defines the initial state, and* $G \subseteq F$ *defines the goal state.*

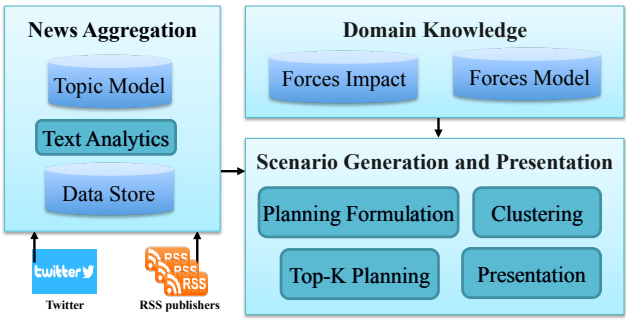


Figure 1: The SPA system architecture

The solution to the planning problem, P, is a sequence of executable actions, $\pi = [a_0, ..., a_n]$ such that if executable from the initial state, I, meets the goal (i.e., $G \subseteq \delta(a_n, \delta(a_{n-1}, \ldots, \delta(a_0, I)))$, where $\delta(a, s) = ((s \backslash \text{DEL}(a)) \cup \text{ADD}(a))$ defines the successor state.

Definition 2 *A Plan Recognition (PR) problem is a tuple* $R = (F, A, I, O, \mathcal{G}, \text{PROB})$*, where* (F, A, I) *is the planning domain as defined above,* $O = \{o_1, ..., o_m\}$*, where* $o_i \in F$*,* $i \in [1, m]$ *is the set of (partially ordered) observations,* $\mathcal{G}$ *is the set of possible goals* G*,* $G \subseteq F$*, and* PROB *is a probability distribution over* $\mathcal{G}$*,* $P(G)$*.*

The solution to the PR problem is the posterior probabilities $P(\pi|O)$ and $P(G|O)$. Plan recognition problem can be transformed to an AI planning problem and the posterior probabilities can be approximated using AI planning (Ramírez and Geffner 2010; Sohrabi *et al.* 2016a). Note, the observations are said to be satisfied by an action sequence if it is either explained or discarded following the work of Sohrabi et al. 2016a. This allows for some observations to be left unexplained in particular if they are out of context with respect to the rest of the observations.

Definition 3 *A scenario planning for enterprise risk management problem is defined as a tuple* $SP = (F, A, I, O, \mathcal{G})$*, where* (F, A, I) *is the planning domain acquired by the domain experts,* $O = \{o_1, ..., o_m\}$*, where* $o_i \in F$*,* $i \in [1, m]$ *is a set of observations selected from the news and social media trends,* $\mathcal{G}$ *is a set of possible goals* $G \subseteq F$*; the set of goals are called business implications in the scenario planning problem.*

As shown in Figure 1, the input to the SPA system are raw social media posts and news articles with RSS feeds. The News Aggregation component analyzes such news and posts and suggests possible observations. In the deployment of the SPA system, we addressed unordered set of observations as input; however, in theory, the observations can be expressed in any Linear Temporal Logic (LTL) formula (Sohrabi *et al.* 2011).

The solution to the SP problem is defined as a set of scenarios, where each scenario is a collection of plans Π such that: (1) each plan $\pi = [a_0, ..., a_i, a_{i+1}, ..., a_n]$ is an action

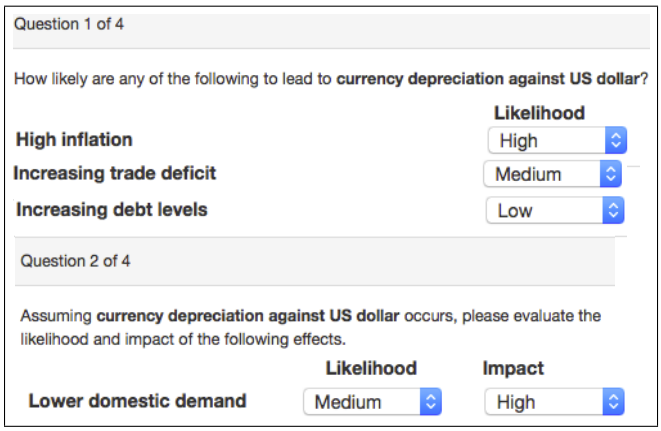


Figure 2: Sample questions

sequence that is executable from the initial state I and results in state $s = \delta(a_n, \ldots, \delta(a_0, I))$, (2) at least one of the goals is met (i.e., $\exists G \in \mathcal{G}$, where $G \subseteq s$), and (3) the set of observations is satisfied by the action sequence $[a_0, ..., a_i]$ (i.e., observations are either explained or discarded). The SP problem can be thought of as a plan recognition problem, where observations and a set of goals are given. Rather than computing $P(\pi|O)$ and $P(G|O)$, the solution to the SP problem is a set of scenarios showcasing the alternative possible outcomes.

4 Knowledge Engineering

While several knowledge engineering tools exists, most of them assume that the domain expert has some AI planning background and these tools provide the additional support in writing the domain knowledge (e.g., (Muise 2016; Simpson *et al.* 2007)). However, we anticipate the lack of proper AI planning expertise in writing the domain knowledge and the unwillingness to learn a planning language. Instead, the domain expert may choose to express their knowledge in a light-weight graphical tool and have this knowledge translated automatically to a planning language such as Planning Domain Description Language (PDDL) (McDermott 1998). In this section, we discuss the representation of the domain knowledge and its translation to planning.

As shown in Figure 1, the domain knowledge comes in two forms: Forces Model and Forces Impact. Forces Model, is the domain knowledge corresponding to the causes and effects of the different forces influencing the risks in a business organization such as the economy, currency, corruption, social unrest, and taxes. The domain experts express these relationship for each force trends (e.g., economic decline and economic growth) in separate Mind Maps. A Mind Map[1] is a graphical method that can be used to express the Forces Model in a simple way. The Mind Maps can be created in a tool such as FreeMind[2] which produces an XML representation of the Mind Maps which can serve as an input to our

[1] https://en.wikipedia.org/wiki/Mind_map
[2] http://freemind.sourceforge.net/wiki/

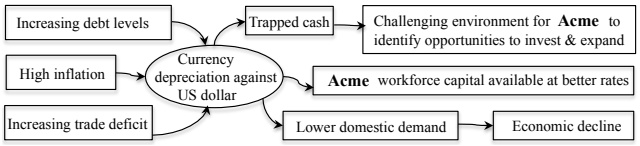


Figure 3: Part of the Mind Map for the currency depreciation against US dollar force.

system. An example Mind Map is shown in Figure 3. The force in this Mind Map is the currency depreciation. The concepts with an edge going towards the force, are the possible causes, and the concepts with an outgoing edge from the force, are the possible effects. The causes and effects can appear in chains, and cascade to other causes, and effects, with a leaf concept of either a business implication (i.e., the planning goal), or another force, with its own separate Mind Map that describes it. For example, "Acme workforce capital available at better rates" is an example of a business implication, where Acme is the name of the organization. Note, one of the leafs of this Mind Map, economic decline, is another force which would be described in a separated Mind Map. Any of the concepts in the Mind Map, except for the business effects, can serve as observations in order to generate the scenarios.

Additional information on the Mind Maps is encoded through the Forces Impact, which is captured by a series of automatically generated questions based on the Mind Maps. These questions are created by a script that reads the XML encoding of the Mind Maps. Sample questions are shown in Figure 2. The domain expert is given options of low, medium, and high in addition to the option of "do not know" in which a default value is selected for them. The answers to these questions determine the weight of the edges in the Mind maps.

The domain knowledge encoded in the Mind Maps (i.e., Forces Model), together with the answers from the questionnaire (i.e., Forces Impact), is automatically translated into a planning language such as PDDL. There are at least two ways to translate the Mind Maps into a planning language. The first method, we call "ungrounded", defines one general and ungrounded set of actions in the PDDL domain file with many possible groundings of the actions based on the given Mind Maps. The domain file includes an action named "indicator" for each of the causes in a Mind Map. There would be three different "indicator" actions, one for each level (i.e., "indicator-low", "indicator-med" and "indicator-high"). The levels are determined based on the answers to the questionnaire. The domain file also includes an action named "next", and "next-bis" for each of the edges in the Mind Map. The "next" action also has three different versions, one for each level. The "next-bis" actions do not have levels and are those that end in a business implication concept (i.e., a concept that includes the name of the company).

Table 1 shows part of the planning domain. For example, the "next-med" action will be grounded by setting the parameter x1 to "increasing trade deficit" and the parameter x2 to the "currency depreciation against US dollar". Each

```
(:action next-med
 :parameters (?x1 - occ ?x2 - occ)
 :precondition (and (occur ?x1)
                    (next-med ?x1 ?x2))
 :effect (and (occur ?x2)
              (not (occur ?x1))
              (increase (total-cost) 10)))

(:action indicator-med
 :parameters (?y - force ?x - occ)
 :precondition (and (indicator-med ?y ?x))
 :effect (and (occur ?x)
              (increase (total-cost) 15)))

(:action next-bis
 :parameters (?x1 - occ ?x2 - bisimplication)
 :precondition (and (occur ?x1)
                    (next-bis  ?x1 ?x2)
 :effect (and (bis-implication-achieved)
              (increase (total-cost) 6)))
```

Table 1: Part of the planning domain.

of the "next" actions (-low, -med, -high) have a cost that maps to the importance of that edge such that lower impact/likelihood answers map to a higher cost. Hence, while the domain is fixed, based on the answers obtained by the domain experts, the actions will have a different set of possible groundings defined in the problem file. The "next-bis" action is the action that if executed, indicates that at least one of the business effects have been reached and the "bis-implication-achieved" predicate is set to true; this is the goal of the planning problem. The problem file (i.e., the initial state) will include all the possible groundings of these actions by including a grounding for the predicates "(next-med ?from ?to)", "(next-bis ?from ?to)", and "(indicator-med ?y ?x)". Note that the size of the Mind Map leads to a larger problem file, as the domain file is fixed. A successful plan maps to an execution of an "indicator" action, followed by the execution of one or more "next" actions, followed by an execution of a "next-bis" action. This maps to a path through the connected Mind Maps.

The second method to translate the Mind Maps into a planning language is called "grounded" which as the name suggests, defines one action per each edge in the Mind Map in addition to one action for each of the causes in the Mind Map in the planning domain itself. So rather than having one fixed planning domain which can get grounded by the problem file, the second approach fully specifies all the possible actions in the planning domain. We evaluate the performance of both methods in the experimental evaluation.

5 Computing Plans

In the previous section, we discussed how to translate the information available in the Mind Maps into a planning domain and problem. However, we are also given the set of observations as the input and we need to compile away the observations in order to use planning. To do so we follow the work of Sohrabi et al. 2013; 2016a which adds a set of "explain" and "discard" actions for each observation. The discard action can be selected in order to leave some observations unexplained. The observations are driven from news and social media posts and not all of them are reliable; in addition, some of them could be mutually exclusive and not all of them could be explainable. Hence, it is important to have the ability to discard some observations. However, to encourage the planner to generate plans that explain as many observations as possible, a penalty is set for the "discard" action in the form of a cost. The penalty is relative to the cost of the other action in the domain; we currently set it to be five times the cost of a "next-med" action. After considering multiple options, this seemed to be good a middle-ground option between the two extremes; a high discard cost will cause the planner to consider many long and unlikely paths, while a low discard will cause the planner to discard observations without trying to explain them. In addition, to ensure all observations are considered, whether explained or discarded, a set of special predicates, one per each observation is used and must hold true for each of the "next-bis" actions. This ensures that a plan that meets one of the goals also has considered all of the observations. To disallow different permutation of the discard action, we discard observations using a fixed order.

The resulting planning problem captures both the domain knowledge that is encoded in the Mind Maps and its associated weights of the edges as well as the given set of observations, and possible set of goals, associated with the plan recognition aspect of the problem. To compute a set of high-quality plans on the transformed planning problem, we use the top-k planning approach proposed in (Riabov *et al.* 2014; Sohrabi *et al.* 2016b). Top-k planning is defined in as the problem of finding k set of plans that have the highest quality. The best known algorithm to compute the set of top-k plans is based on the k shortest paths algorithm called K^* (Aljazzar and Leue 2011) which also allows use of heuristics search. We use the K^* algorithm together with the LM-cut heuristic (Pommerening and Helmert 2012) in our system. Next, we discuss how the generated plans are post-processed into the scenarios.

6 Computing Scenarios

To compute the type of scenarios shown in Figure 4, we perform a set of post-processing steps on the computed set of plans. All of the post-processing steps are done automatically. First, we identify the number of plans out of the top-k plans (e.g., 1000) generated by the planner to consider for scenario generation. We argue that this number is problem-dependent rather than being a fixed number for all problems. To calculate the cost cutoff, we calculate the average and the standard deviation of the cost of all plans among the top-k plans. We then consider plans that have a lower cost than the average cost subtracted by the standard deviation. The number of plans considered for scenario generation is shown under the "# of Plans" column in Table 2.

Next, we cluster the resulting plans to create scenarios. Hence, rather than presenting all plans, we group similar plans and only present 3-6 clusters of plans to the end user. We cluster plans according to the predicates present in the last state. Given that the number of ground predicates (i.e, $\mathcal{F}$) is finite, we first represent each plan through a bit array of the same size such that 1 indicates the predicate is in the

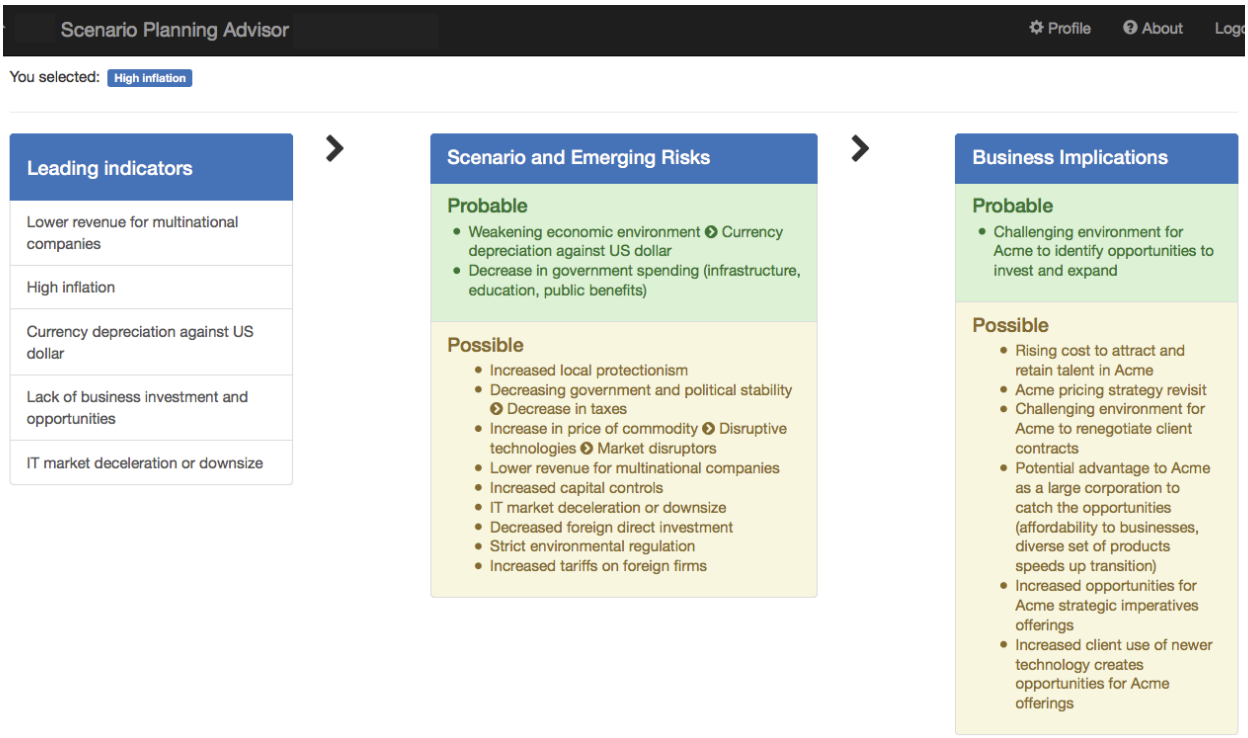


Figure 4: The screenshot of a sample generated scenario for the high inflation observation. Each scenario is divided into three parts, the leading indicators, scenario and emerging risks, and the business implications.

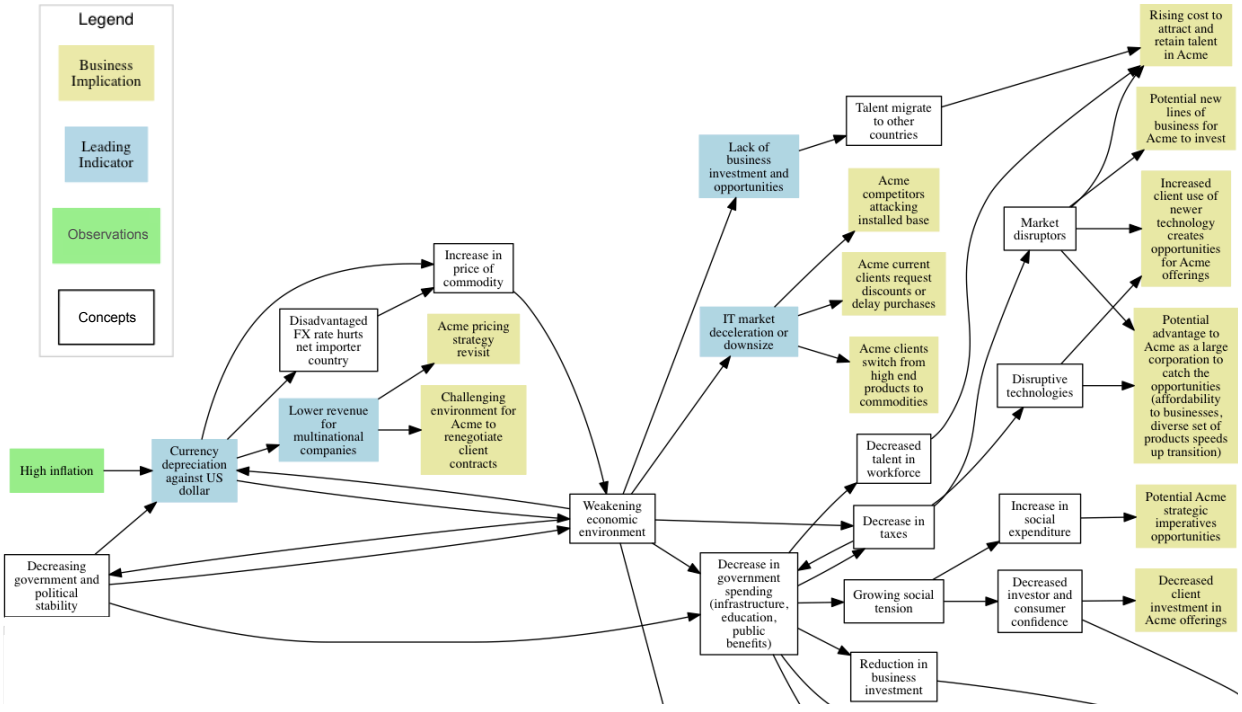


Figure 5: Part of the screenshot of a explanation graph for the scenario shown in Figure 4. Observations are shown in green, leading indicators are shown in blue, and business implications are shown in yellow.

final state, and 0 indicates that the predicate is not in the final state. To determine the Euclidean distance between two plans, we compute a XOR of the corresponding bit arrays and take the square root of the sum of 1 bits. Normally, we want to avoid plans with opposite predicates (e.g., weakening/strengthening economic environment, increase/decrease in inflation, etc.) ending up in the same cluster. To ensure this, we add a penalty factor to the number of 1 bits we use to compute the distance for every pair of opposite predicates. Given this distance function for each pair of plans, we compute a dendrogram bottom-up using the complete-linkage clustering method (Defays 1977). The user can specify a minimum and maximum consumable number of scenarios. These settings are used to perform a cut through the dendrogram that yields the number of plans in the specified interval with the optimal Dunn index (Dunn 1973), a metric for evaluating clustering algorithms that favors tightly compact sets of clusters that are well separated.

After post-processing is complete, we automatically perform several tasks to prepare the scenarios for presentation. First, we separate the predicates in each cluster (scenario) into business implications and regular predicates. At the same time, we separate probable and possible predicates in each of these categories by determine the proportion of plans where the predicate is present in the last state from all plans in the scenario; predicates that appear in more than 66% of plans are put into the probable category, those that appear between 25% and 66% are placed in the possible category. Second, we identify discriminative predicates, i.e. predicates that appear early on the plans that are part of one scenario but not other scenarios (i.e., they tend to lead to this scenario and not others); these are useful to monitor in order to determine early on whether a scenario is likely to occur. Third, we compute a summary of all plans that are part of the scenario and present this as a graph to the user. Figure 5 shows an example of this graph. This serves as an explanatory tool for the predicates that are presented in each scenario. This graph also shows how the different Mind Maps are connected with each other through concepts that are shared between them.

7 Experimental Evaluations

In this section, we evaluate: (1) the performance of the planner, (2) quality of the clusters measured by the size of the cluster, and (3) how informative each cluster measured by number of predicates and business implications. In the next section, we provide details on the pilot deployment of the Scenario Planning Adviser (SPA) tool, feedback and the lessons learned in interacting with the domain experts as well as the business users. All our experiments were run on a 2.5 GHz Intel Core i7 processor with 16 GB RAM.

We compare the performance of the planner on our two proposed methods to translate the Mind Maps into a planning domain: "ungrounded" and "grounded". The "grounded" method creates 670 actions when considering the full set of Mind Maps. We remove some of these Mind Maps creating 403 actions instead and report on that result under the "ungrounded small" method. To increase the difficulty of the problem, we increase the size of the O. Observations are chosen randomly from the set of possible observations.

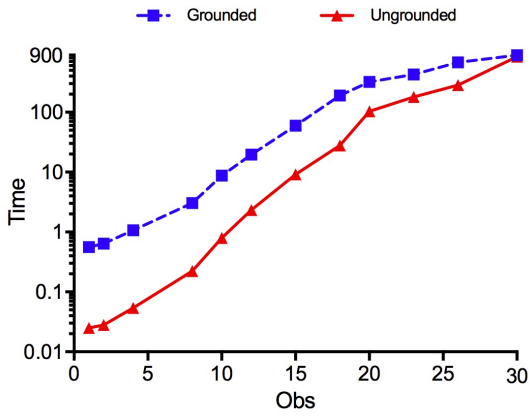


Figure 6: Planning performance comparison between the "grounded" and "ungrounded" methods, as we increase the number of observations. The time is in seconds and is shown in logarithmic scale.

Table 2 presents a comparison between "ungrounded small" and "grounded". The objective of this experiment is to show how the planning domain size influence performance and the generated clusters. All numbers shown in each row are averages over 10 runs of the same type of problem, where the same number of observations is considered in both cases. The columns show the planning performance in seconds, total number of business implications, $\mathcal{G}$, number of actions, $\mathcal{A}$, number of observations O, number of discarded observations in the optimal plan, "# of Discards", number of plans considered for scenario generation, "# of Plans", and number of scenarios generated "# of Scenarios". We also show the average, standard deviation, max, and min count on the number of members of each cluster, number of predicates, and number of business implications in each scenario. We used a timeout of 900 seconds. The problems with 30 or more observations did not finish within the time limit.

The results show that the performance of the planner depends on both the number of observations and the size of the domain, as expected. As the number of observations grow the planner's performance worsens but this does not influence the number of plans, the number of scenarios, size of the clusters, or the number of scenario predicates. However, the number of business implications decreases, as expected, as the observation size grows. Looking at the average number of cluster members, the average number of scenarios predicates, and the average number of bossiness implications, the results show that the clusters in both cases are balanced and informative.

We also compare the planning performance between two methods of translating the Mind Maps. The results in logarithmic scale is shown in Figure 6. Each shown point in the figure is an average over 20 instances. The results show that in our current implementation, as the number of observations increases, planning performance using

	Time	$\|\mathcal{G}\|$	$\|\mathcal{A}\|$	$\|O\|$	#of Discards	#of Plans	#of Scenarios	Cluster Members Avg	Cluster Members σ	Cluster Members Max	Cluster Members Min	Scenario Predicates Avg	Scenario Predicates σ	Scenario Predicates Max	Scenario Predicates Min	Bis Implications Avg	Bis Implications σ	Bis Implications Max	Bis Implications Min
Ungrounded Small	0.03	65	403	1	0.0	129.0	3.8	37.0	28.6	76.9	11.2	9.6	2.7	13.5	6.6	4.8	1.7	7.1	2.7
	0.03	65	403	2	0.5	141.7	3.8	39.9	31.7	83.4	6.5	9.8	3.1	13.7	5.4	4.1	1.7	6.1	1.8
	0.05	65	403	4	1.6	120.5	3.6	34.6	27.9	72.1	7.9	10.9	2.7	13.9	6.9	3.7	1.2	5.3	2.1
	0.22	65	403	8	4.4	122.4	3.8	34.8	33.4	82.6	4.3	10.0	2.4	13.0	6.9	2.1	0.9	3.5	1.3
	0.80	65	403	10	5.0	112.6	4.5	25.6	26.0	71.5	5.6	7.6	2.0	10.1	5.4	2.3	0.8	3.8	1.6
	2.33	65	403	12	5.9	100.1	4.2	25.3	20.7	56.2	4.4	9.4	1.4	11.1	7.4	1.7	0.4	2.6	1.2
	9.16	65	403	15	8.8	104.8	3.9	30.2	25.6	68.5	8.8	10.6	1.2	12.4	8.8	1.9	0.4	2.8	1.5
	27.85	65	403	18	9.9	92.8	4.8	20.2	23.5	61.3	3.0	8.5	1.2	10.3	6.7	1.6	0.5	2.4	1.3
	103.71	65	403	20	11.3	117.7	3.9	30.9	26.8	68.0	3.7	9.0	1.4	11.0	7.3	1.8	0.6	2.5	1.0
	179.90	65	403	23	14.9	103.7	4.1	26.3	21.2	58.6	4.4	9.0	1.4	11.1	6.9	1.9	0.6	2.7	1.2
	282.87	65	403	26	16.9	90.6	4.9	20.3	19.0	53.5	5.3	9.5	1.1	11.3	7.8	1.6	0.3	2.0	1.3
Ungrounded	0.03	112	670	1	0.0	91.5	4.4	24.4	16.6	48.6	6.6	7.0	2.5	10.4	4.3	4.5	1.7	6.6	2.2
	0.04	112	670	2	0.4	132.1	4.3	34.4	32.2	80.3	3.7	8.0	3.0	11.7	4.0	3.8	1.8	6.1	1.5
	0.08	112	670	4	1.5	114.1	3.6	32.9	30.7	77.9	4.3	10.2	2.9	13.2	6.1	3.6	1.4	6.0	2.3
	0.35	112	670	8	3.7	109.7	3.6	31.5	24.6	65.9	7.0	9.1	1.9	11.4	6.5	3.8	1.3	5.4	2.3
	1.17	112	670	10	5.1	139.4	4.2	34.6	27.9	73.2	5.9	7.8	2.0	10.1	4.9	2.6	0.8	3.9	1.6
	3.35	112	670	12	5.4	99.5	4.8	22.8	24.8	64.7	3.5	8.6	1.9	11.0	6.3	1.6	0.4	2.8	1.2
	22.01	112	670	15	8.1	92.3	4.1	23.3	22.1	57.9	3.2	9.9	1.8	12.0	7.0	2.5	1.0	4.0	1.5
	85.73	112	670	18	9.4	88.5	4.3	22.2	19.2	51.6	6.7	7.2	1.1	8.7	5.5	2.2	0.3	2.7	1.7
	144.89	112	670	20	10.7	124.3	5.1	26.0	19.4	57.0	5.0	9.0	1.0	10.0	7.2	2.1	0.2	2.3	1.9
	284.73	112	670	23	14.5	106.8	4.8	24.5	23.9	62.5	4.0	8.6	1.6	10.6	6.5	2.9	0.6	3.6	2.0
	511.95	112	670	26	16.8	80.0	4.7	17.2	9.0	30.2	7.8	7.8	1.0	9.5	6.5	1.7	0.7	2.8	1.2

Table 2: Comparison between "ungrounded" and "ungrounded small" as we increase the number of observations: "grounded" considers all of the Mind Maps, "ungrounded small" considers a smaller set of Mind Maps.

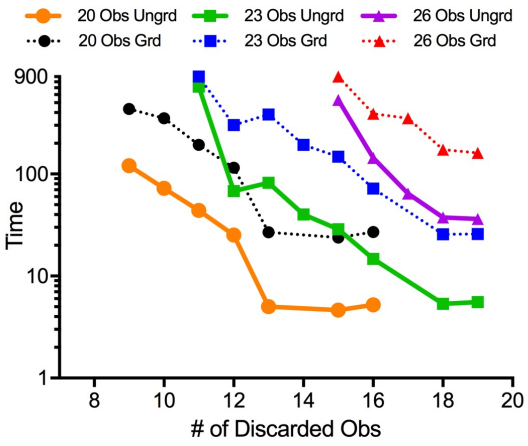


Figure 7: Planning performance comparison between two methods (i.e, "grounded" and "ungrounded") as the number of discarded observations in an optimal plan increases when considering problems with 20, 23, and 26 observations. The time is in seconds and is shown in logarithmic scale.

the "ungrounded" method is significantly better than the "grounded" method.

Considering problems with 20, 23, and 26 observations, we also looked at the number of discarded observations in the optimal plan in each case. This indicates whether or not the observations are explainable in a single path through the Mind Maps. The results in logarithmic scale is shown in Figure 7. The results confirm that the performance of the "ungrounded" method is better than the "grounded" method. It also shows that as the number of discarded observation increases, the planning time decreases. This seems to indicate that the planner identifies the unexplainable observations, through its heuristics, and does not spend time on explaining the unexplainable observations.

Based on these results, we conclude that performance of the planner depends on number of observations, the size of the domain, the method used in the translation of the Mind Maps, as well as the number of unexplainable observations. Given this result, we deployed the "ungrounded" method and use the full set of Mind Maps.

8 User Experience

The SPA tool was evaluated in a pilot deployment with 7 teams of business users, whose responsibilities included risk management within their business area. For those teams SPA was introduced together with the new scenario planning process; hence, there was no pre-automation baseline available to compare against. In addition the functionality provided by the tool cannot be reproduced manually due to the broad news analysis the tool provides.

The Mind Map were developed over the course of three months by one enterprise risk management expert working with an assistant and in consultation with other experts. While Mind Maps in general can be in any form, we briefly educated the domain expert to provide Mind Maps that have one force (e.g., currency deprecation against US dollar) as their main concept and provide causes and consequences of this force in one Mind Map; the concepts with an edge to the central concept and the concepts with an edge from the main concept and their cascading effects where the last effect is either a business implication or another force with its own separate Mind Map. This ensures that we can automatically translate the Mind Maps into a planning language. We used 23 Mind Maps in the pilot deployment and used the "ungrounded" method to translate the Mind Maps. The re-

sulting planning problem that aggregates the knowledge of all Mind Maps (i.e., the grounding of the actions based on the edges on the Mind Maps) has around 350 predicates and 670 actions.

Additionally, the end users (i.e., the analysts) provided us with a list of possible keywords, such as organizations of interest, key people, key topics, and were able to pick the relevant sequence of observations when we presented them with the summary of relevant news and RSS publications. For RSS publications, around 3,000 news abstracts from 64 publishers, and for Twitter, around 73,000 tweets from around 32,000 users matched our keyword search criteria.

The teams have universally found the tool easy to use and navigate. Although no detailed feedback was collected for each scenario, the teams have reported that approximately 80% of generated scenarios had identified implications directly or indirectly affecting the business. By design, the tool is trying to help the business users to think outside the box and it is expected that some of the scenarios it generates will not be relevant. Judging by the provided comments, the teams whose business is affected by frequent political, regulatory and economic change have found the tool more useful than those operating under relatively stable conditions.

In addition, the teams found the explanation graph, visualization of a set of plans, essential to the adaptation of the tool (Figure 5). They believe that the explanation graph "demystifies" the tool by providing them with an explanation of why they are presented with a particular scenario. This is critical for the business users or policy-makers who would be basing their decisions on the generated scenarios.

The suggestions for improvement focused primarily on the need for further automated assistance in selecting observations based on the news, to ensure that no important context is lost, and on the additional information about the scenarios. Several teams have requested confidence levels or at least ranking information provided with the generated scenarios. We believe this is an interesting future direction and believe more accurate models are required in order to provide that additional information.

In working with the domain experts and users from the start of the pilot deployment, we learned several lessons: (1) The users are interested in being presented with several scenarios rather than one along with the explanation of each scenario. This captures the possible alternatives rather than a precise prediction, analogous to a generation of a multiple plans rather than a single (optimal) plan; (2) The users wanted personalized scenarios specific to their particular use case. To address that we consider the Mind Maps as a template that holds true for all use cases and allow personalization of the scenarios by incorporating different weights of the edges of the Mind Maps. As mentioned previously we automatically generate a serious of questions in order to obtain the impact and likelihoods that are specific to a use case. Hence, computing a set of high-quality plans for different use cases results in different set of plans, which in turn results in different scenarios; (3) The domain experts found themselves continuously updating the Mind Maps after interacting with the tool and we had to enable those continuous updates. In addition to building the automated technique of translating the Mind Maps to planning language, we assigned unique identifiers to each of the concepts in the Mind Maps. This allowed us to develop scripts for supervised detection and propagation of the associated knowledge throughout theses changes.

9 Related Work and Summary

There exist a body of work on the plan recognition problem with several different approaches (e.g., (Zhuo *et al.* 2012)). However, most approaches assume that the observations are perfect, mainly because they do not take as input the raw data and that they do not have to analyze and transform the raw data into observations (Sukthankar *et al.* 2014). Also, most plan recognition approaches assume plan libraries are given as input, whereas we use AI planning (Goldman *et al.* 1999). Furthermore, there is a body of work on learning the domain knowledge (Yang *et al.* 2007; Zhuo *et al.* 2013). Our focus in addressing knowledge engineering challenges was to transform one form of knowledge, expressed in Mind Maps, into another form that is accessible by planners. Learning can be beneficial in domains in which plan traces are available.

In this paper, we applied AI planning techniques for a novel application, scenario planning for enterprise risk management and addressed knowledge engineering challenges of encoding the domain knowledge from domain experts. To this end, we designed Scenario Planning Adviser (SPA), that takes as input the raw data, news and social media posts, and interacts with the business user to obtain key observations. SPA also allows upload of Mind Maps, as one way of expressing the domain knowledge by the domain experts, and obtains additional information based on these Mind Maps by an automatically generated questionnaire. SPA then automatically generates scenarios by first generating large number of plans and then clustering the generated plans into a small set (i.e., 3-6) in order to be consumable by a human user. The SPA system is in pilot deployment with several teams of business users. The feedback we have received so far have been positive and show that our approach seems promising for this application.

10 Acknowledgements

We thank Fang Yuan and Finn McCoole at IBM for providing the domain expertise. We thank Nagui Halim and Edward Shay for their guidance and support. We also thank our LAS collaborators. This material is based upon work supported in whole or in part with funding from the Laboratory for Analytic Sciences (LAS). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the LAS and/or any agency or entity of the United States Government.

References

Husain Aljazzar and Stefan Leue. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129–2154, December 2011.

Evgeny Avanesov. Risk management in ISO 9000 series standards. In *International Conference on Risk Assessment and Management*, volume 24, page 25, 2009.

D. Defays. An efficient algorithm for a complete link method. *Computer Journal*, 20(4):364–366, 1977.

J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

Robert P. Goldman, Christopher W. Geib, and Christopher A. Miller. A new model of plan recognition. In *Proceedings of the 15th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 245–254, 1999.

Drew V. McDermott. PDDL — The Planning Domain Definition Language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

Christian Muise. Planning.Domains. In *the 26th International Conference on Automated Planning and Scheduling - Demonstrations*, 2016.

Garry D Peterson, Graeme S Cumming, and Stephen R Carpenter. Scenario planning: a tool for conservation in an uncertain world. *Conservation biology*, 17(2):358–366, 2003.

Florian Pommerening and Malte Helmert. Optimal planning for delete-free tasks with incremental LM-Cut. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

Miquel Ramírez and Hector Geffner. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1778–1783, 2009.

Miquel Ramírez and Hector Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI)*, 2010.

Anton Riabov, Shirin Sohrabi, and Octavian Udrea. New algorithms for the top-k planning problem. In *Proceedings of the Scheduling and Planning Applications woRKshop (SPARK) at the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 10–16, 2014.

Paul JH Schoemaker. Scenario planning: a tool for strategic thinking. *Sloan management review*, 36(2):25, 1995.

Ron M. Simpson, Diane E. Kitchin, and T. L. McCluskey. Planning domain definition using GIPO. *Knowledge Eng. Review*, 22(2):117–134, 2007.

Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. Preferred explanations: Theory and generation via planning. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI)*, pages 261–267, 2011.

Shirin Sohrabi, Octavian Udrea, and Anton Riabov. Hypothesis exploration for malware detection using planning. In *Proceedings of the 27th National Conference on Artificial Intelligence (AAAI)*, pages 883–889, 2013.

Shirin Sohrabi, Anton Riabov, and Octavian Udrea. Plan recognition as planning revisited. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

Shirin Sohrabi, Anton Riabov, Octavian Udrea, and Oktie Hassanzadeh. Finding diverse high-quality plans for hypothesis generation. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, 2016.

Shirin Sohrabi, Anton Riabov, and Octavian Udrea. State projection via ai planning. In *Proceedings of the 31st Conference on Artificial Intelligence (AAAI-17)*, 2017.

René M Stulz. Rethinking risk management. *Journal of applied corporate finance*, 9(3):8–25, 1996.

Gita Sukthankar, Christopher Geib, Hung Hai Bui, David V. Pynadath, and Robert P. Goldman. *Plan, Activity, and Intent Recognition*. Morgan Kaufmann, Boston, 2014.

Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143, February 2007.

Hankz Hankui Zhuo, Qiang Yang, and Subbarao Kambhampati. Action-model based multi-agent plan recognition. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 377–385, 2012.

Hankz Hankui Zhuo, Tuan Nguyen, and Subbarao Kambhampati. Refining incomplete planning domain models through plan traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2451–2457, 2013.

Towards Automated Vulnerability Assessment

Saad Khan and Simon Parkinson
Department for Informatics, School of Computing and Engineering, University of Huddersfield, UK
Email: firstname.surname@hud.ac.uk

Abstract

Vulnerability assessment (VA) is a well established method for determining security weaknesses within a system. The VA process is heavily reliant on expert knowledge, something that is attributed to being in short supply. This paper explores the possibility of automating VA and demonstrates an initial proof-of-concept involving decision-making skills comparable with a human-expert. This is achieved through encoding a domain model to represent expert-like capabilities, and then using model-based VA planning to determine VA tasks. Although security evaluation is a complex task, through the help of such models, we can determine the ways to find potential vulnerabilities without an expert present. This technique allows time constrained assessments, where a 'risk factor' is also encoded to represent the significance of each security flaw. The ultimate goal of this work-in-progress is to realistically simulate a human vulnerability auditor. This paper demonstrates the first step towards that goal; a systematic transformation of the VA knowledge into a PDDL representation, accommodating a broad range of time constrained investigative actions. The output plan and its analysis evidently evinces many potential benefits such as increased feasibility and productivity.

Introduction

Security vulnerabilities exist in IT infrastructures within most organisations, and given the increasing size and importance of the infrastructure on a organisation's daily business, there is a pressing need to identify and mitigate security vulnerabilities. The organisation itself is responsible for protecting their IT resources against potential attacks, and this will often be performed through conducting periodic security assessments. However, an organisation may not always have the necessary expertise in-house and they will be required to pay for external consultancy. If an organisation does have in-house expertise to maintain their security, they are also required to maintain such expertise in this rapidly changing discipline. Both approaches incur a large financial cost and there is wide-scale motivation to decrease costs, as well as make an organisation more agile in that they are quicker to respond to detecting vulnerabilities as new threats develop. The general principle behind vulnerability assessment process can be summarised as (Kamongi et al. 2013):

1. Identify and taxonomise available system resources such as network and operating systems;
2. Prioritise resources or assets based on their importance level such as data sensitivity;
3. Determine threats to each resource and create potential point of vulnerabilities;
4. Based on the importance level, remove the most serious potential problems first and so on; and
5. Create a policy or guideline, that can minimise the consequences if an attack occurs in future.

A major bottleneck behind security assessments is the lack of knowledge and understanding of the latest potential threats. The adversaries are continuously becoming increasingly sophisticated in their attack mechanisms, and anyone who is not improving their security accordingly can become the victim of a damaging attack. Another important factor to consider is the inevitable human error during system configuration and use. In our previous work (Khan and Parkinson 2016), we created an expert system based on If-Then rules but the system produced mutually exclusive actions, was difficult to maintain and had very limited intelligence. A potential solution to these problems is through the use of computational intelligence to generate security evaluation plans in an automated manner. In general terms, the intelligent technique should consider all known evaluation techniques and determine their applicability to the system, and to identify mitigation plans for a complex system. The system should imitate and support the human expert's decision-making ability.

In this paper, we propose a system that applies Automated Planning (AP) to generate vulnerability assessment strategies for manual checking. It uses the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) for developing domain models and encoding problem instances. We use PDDL 2.2 for its support of durative actions, numeric fluents (Fox and Long 2003), and timed initial literals (Edelkamp and Hoffmann 2004). The domain model is essentially the description of knowledge, gathered from expert experience and published work detailing vulnerability assessment techniques. Each vulnerability is transformed into one or more actions. To demonstrate the suitability of the approach, we have also created a simple post-processor

that can automatically translate the output plan file to a human understandable format.

The paper is organised as follows: the first section provides a brief summary of vulnerability assessment and the applications of planning in cyber security systems. The next section is devoted to the detailed explanation and example results of the proposed solution. This leads to the experimental analysis section, whereby solutions are tested under different circumstances. This section also discusses the advantages of the proposed solution. Following this, the paper concludes and the direction of further work is provided.

Related work

Vulnerability assessment

Vulnerability assessment is the process of determining the security gaps of a system, which can be exploited by the attacker from inside or outside of organisation, to gain confidential data, financial benefits, amongst others (Umrao, Kaur, and Gupta 2012). Many generic vulnerability assessment tools are available that can identify known security flaws such as OpenVAS, Burp Suite, Nikto, Vega, AppScan, AVDS and Grabber (Owsap 2017). The purpose of these tools is to determine a system's security issues and stay ahead of attackers by constantly patching and mitigating identified vulnerabilities. Apart from the generic tools and approaches, some vulnerability detection techniques target specific applications. For example, (Benton, Camp, and Small 2013) performed a detailed security assessment for OpenFlow protocol, (Ristov, Gusev, and Donevski 2014) assessed the vulnerabilities of OpenStack's architectural components, (Zhao and Zhao 2015) analysed privacy and security issues of social media sites, (Barrere, Badonnel, and Festor 2014) identified and explained the vulnerabilities of autonomic systems and (Rahman, Ahmad, and Ramli 2014) discussed potential Wireless Body Area Network security vulnerabilities. Many patents also present different vulnerability assessment techniques, e.g. (Webb, Boscolo, and Gilde 2016) created a network appliance that can evaluate security of multiple networks concurrently. These specific evaluations are limited in use but provide deep insight into a particular product.

One of the major shortcomings of aforementioned approaches is that they require extensive knowledge for running and understanding the results. It is very difficult for a non-expert to conduct the security evaluation without first spending significant time to acquire the necessary expertise. Furthermore, most of the approaches do not consider time limits, prioritisation and real-time damages associated with the vulnerabilities. The damages might cause exposure of sensitive data, unavailability of crucial services and many others. These factors motivate the requirement for an automated system, which can decide and prioritise vulnerabilities based on time constraints and the potential for damage, and outputs the most feasible solution without relying on a human expert.

Applications of Planning in Cyber-Security

There have been successful exploration of the use of Automated Planning (AP) in different cyber security domains, mainly for generating attack plans for penetration testing (Riabov et al. 2016). In this work, courses of actions are generated based upon a system configuration; however, the goal is adversarial in that the aim is to compromise the system in efficient shortest path, albeit by a trusted security professional (widely termed white-hat hacking). Recent work by Sohrabi et. al pursues the use of hypothesis exploration for identifying potential attack plans in network security (Sohrabi, Udrea, and Riabov 2013; Sohrabi et al. 2016). Furthermore, recent research presents continued development of AP for penetration testing (*pen testing*) (Shmaryahu 2016a; Hoffmann 2015) discussing the need to overcome scalability limitations. The fundamental difference between pen testing and vulnerability assessment (VA) is that VA is searching for vulnerabilities that exist and mitigate them, where as pen testing is searching to exploit a series of vulnerabilities for adversarial gain.

Penetration testing frameworks are available, both commercial and open-source, which can perform expert-like security assessment through simulated attacks on different systems. One such example is *Metasploit* that can launch exploits and drop payloads to damage remote systems (Maynor 2011). The problem with such frameworks is that expert knowledge is required to manually select and launch the attacks. Although, some security weaknesses such as unpatched software and insecure ports can be identified by vulnerability scanning tools, but their results might not be comprehensive (Holm et al. 2011). Studies suggests that, if the attack plans are generated by a computer, there is potential to discover more plans than human expert, meanwhile helping the non-expert to avoid the complexity and save time, effort and resources. One such commercial tool (*Core Impact*) and uses AP to generate possible attack plans and performs real-time penetration testing (Sarraute, Richarte, and Lucángeli Obes 2011). It uses Probabilistic PDDL (PPDDL), which is capable of extending attack graphs models and handling probabilistic and numerical effects. The system also constructs AND-OR trees to determine candidate attacks paths towards a particular asset. The tool is also efficient in terms of execution time and in the generated network traffic. It's computational complexity is $O(n \log n)$, where n is the total number of actions in domain file. Similar work has been done by (Shmaryahu 2016b), where contingent plan trees are constructed for simulated pen testing.

One initial piece of work involved the use of classical planning to generate hypothetical attack scenarios to exploit the system (Boddy et al. 2005). The study simulates realistic adversary courses of action and mainly focuses on malicious insider's threat. The domain model includes 25 different objects (basic elements of computing), 124 predicates (information about system) and 56 actions (adversarys objectives), whereas each problem contains between 200 to 300 facts. Classical and forward heuristic planners, specifically FF-Metric (Hoffmann 2003) are used to generate attack plans. As writing domain models manually can be labour in-

tensive and prone to errors, M4 macros have been used to design large scale PDDL files, hence avoiding the need for representing actions and facts directly. Their tool also translates the planner output into human-readable format (post-processing) by using a Perl script.

Another paper (Obes, Sarraute, and Richarte 2013) uses planning to assess network security. First, a transformation algorithm is used to convert attack models into PDDL representation. Attack information containing requirements and exploits are encoded into a domain file, while the information about system such as networks, machines, operating systems, ports and running services are stored in problem files. The object types are the system properties such as privileges and operating systems, while predicates are essentially depicts the relationship among objects. This paper analyses the whole network, has up-to 1800 actions and hosts 700 exploits. However, as classical planning is used, the system cannot handle incomplete knowledge.

Partially Observable Markov Decision Processes (POMDP), can be used to overcome the limitation on incomplete knowledge by generating attack plans even if the planner is given incomplete knowledge and uncertainties (Sarraute, Buffet, and Hoffmann 2013a). POMDPs are capable of prioritising actions based on expected reward that is composed of asset value, time and risk of detection, to find the optimal terminal state. Further research (Sarraute, Buffet, and Hoffmann 2013b) investigates how to produce better attack plans for a particular machine within short period of time. Their solution employs intelligent vulnerability scanning actions through using POMDPs to find feasible attacks for each individual machine and inquires targeted network structure approximations on-demand. Despite all the advantages of POMDPs, they are complex and require large computational resources. It is also difficult to design the 'initial belief' for every real-world problem. As a solution, (Hoffmann 2015) presents a middle ground between classical planning and POMDPs called MDPs. The actions work same as before, but they do not perform scanning. Every outcome of action (effect) is assigned a probability regardless of host configuration predicates. The probability value depends on the level of attackers uncertainty in launching that particular action. As PDDL is not equipped to tackle these uncertainties, the paper also suggests a PDDL-like language that can allow probability values inside action.

Literature review shows that planning has been applied to automate pen testing, but to best of our knowledge, there is no such work in automating VA. According to the survey (Shah and Mehtre 2015), VA and pen testing are different in term of motivations and objectives. VA is applied to a system which is likely to have vulnerabilities, unlike penetration testing where system defences are tested. The penetration testing has specific goals and requires particular expertise, whereas VA finds and prioritises the assessment of system vulnerabilities. As the VA is first step towards maturing the security state of the entire system and focuses on both breadth over depth of analysis, we recognise the need for automation and we provide a feasible and resourceful solution.

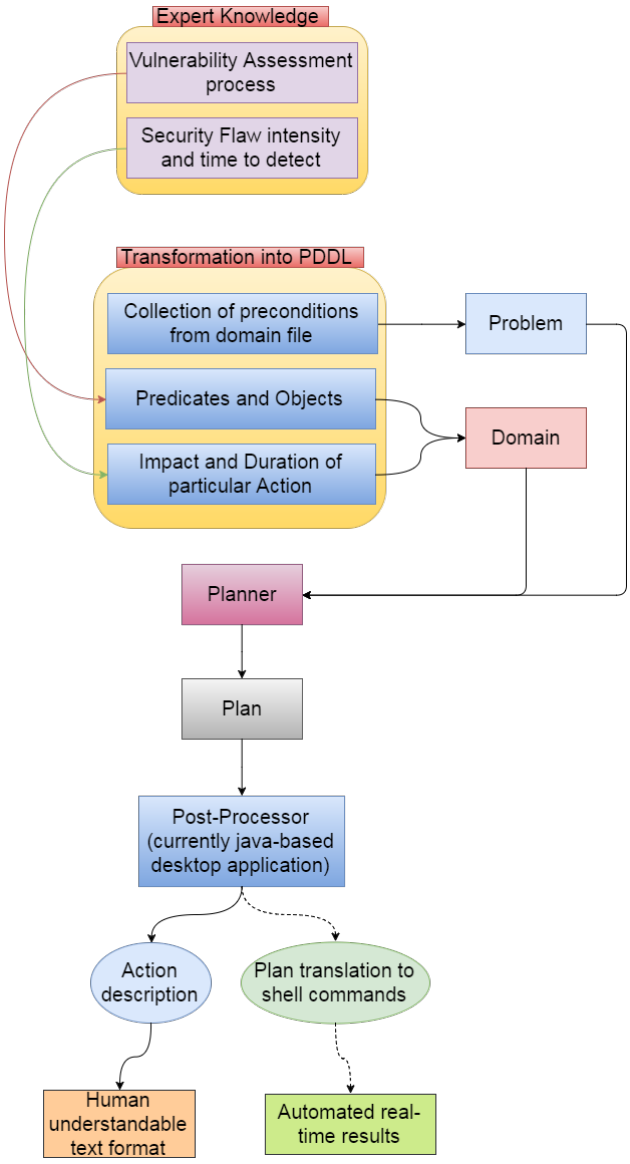


Figure 1: Architecture of proposed system

Automated Planning as a Solution

The purpose of our solution is to help the non-expert users to determine efficient and time-constrained VA checks and perform them manually. The solution is not aimed at replacing the human-expert, but rather in providing decision support aid to users of all technical abilities. The following sections contain a detailed description of the system design, domain modelling, problem description and a sample plan output.

System Design

This section discusses the overall system design and its components. Our proposed solution is inspired by (Sarraute, Richarte, and Lucángeli Obes 2011) as is shown in Figure 1. The explanation of each module is in the following:

Pre Vulnerability Assessment (VA) process – The purpose of VA is to systematically evaluate the security of any given

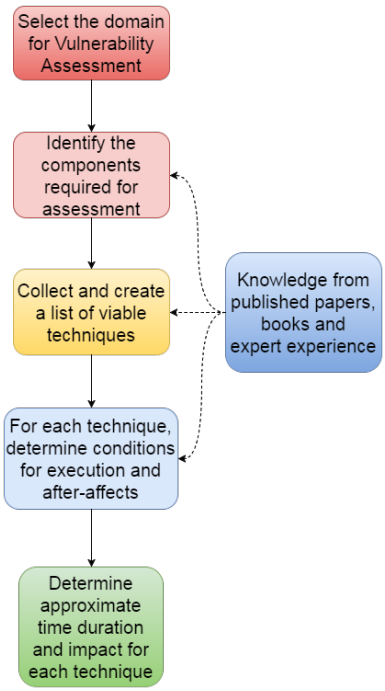


Figure 2: Pre-Vulnerability-Assessment process

system. The VA process should comply to well-established criteria and standards. For now, our solutions is mainly focused on *Authentication*, *Authorisation* and *Permission control* along with few data security assessment techniques. To develop a system that advises on relevant VA procedures, we need to manually collect authentic, expert and verified knowledge on existing VA techniques, e.g. using acquisition software such as (Parkinson and Crampton 2016). The formal preprocess of VA is described in figure 2. It shows the tasks that were conducted before creating the solution. The VA procedures consists of strict steps, which should be followed in the same manner and sequence, hence the need for systematic knowledge acquisition. The first step is to identify the domain whose vulnerabilities are going to be assessed. Then, with the help of expert knowledge and published work, relevant, applicable and useful VA techniques should be extracted manually, along with their preconditions and effects. These conditions and effects are modelled into PDDL later on.

Representing domain knowledge – After information is gathered regarding VA procedures, it is manually converted into predicates and objects, which signify the properties and relationships of individual VA procedures. The group of inter-related predicates form an action, whereas each VA process is defined by one or more actions. For each action, we are also estimating and modelling their durations of identification and potential damage level in case that vulnerability is exploited. The quantification of impact and duration values are then used to provide the most feasible results in accordance with given user-requirements. The user can input deadline value and output plans will inform about the most crucial VA procedures that should be conducted within

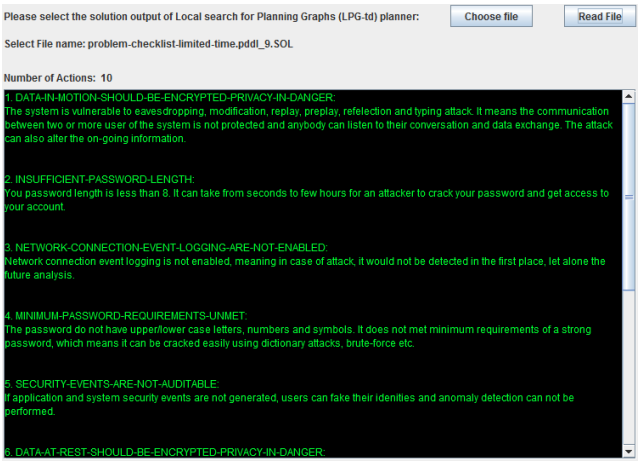


Figure 3: Plan's Post Processor

that limited time.

Problem description – The initial state is a collection of all preconditions from domain file which describe the underlying system. The goal state is empty except from the time duration limit. This is because the solutions aims to find the most feasible plan for VA within given a deadline, whilst maximising the overall impact. The system has static knowledge, which means it will always output the same set of actions. The only variation in plans is brought by imposing time constraints in each action, where the planner will choose actions based on their impact on the optimisation metric.

Planner – LPG-td (Local search for Planning Graphs) planner (Gerevini et al. 2004) is used to extract plans from domain and problem description files. LPG-td, an improved version of LPG, supports durative-actions and plan adaptation (as the goal state is not explicitly described) problems. We used LPG in this initial work due to its general good performance and handling of PDDL features.

Plan – Consists of actions, which represent the actual vulnerability. For example, *(SYSTEM-VULNERABLE-TO-DENIAL-OF-SERVICE-ATTACK SYSTEM)* is a single action of certain plan. It describes that the system is susceptible to denial of service attack. The plan would continue to have a sequence of actions, used to describe mitigation actions to pro-actively eradicate vulnerabilities, minimise threats and prevent future attacks.

Post-processing – The post processor is a Java-based application, whose only purpose is to elaborate the plan in a more human understandable format (shown in figure 3). It contains a simple mapping of actions to their respective descriptions. In future, we aim to enhance the preprocessor in a way, that can convert the actions into appropriate and complete shell commands, which can be executed to perform real-time automated VA operations.

Domain modelling

The domain file consists of 23 durative-actions. Each action contains parameters, duration, conditions and effects. The parameters define objects required for the action to work.

```
(:durative-action Minimum-password-requirements-unmet
  :parameters (?password)
  :duration (= ?duration 5)
  :condition (and
    (over all (No-Uppercase-Letters ?password))
    (over all (No-Lowercase-Letters ?password))
    (over all (No-Numbers ?password))
    (over all (No-Symbols ?password))
    (over all (not
    (Minimum-password-requirements-unmet-found ?password)))
    (over all (action-durations))
    (at start (not (operator-busy) ) )
  )
  :effect (and
    (at end
    (Minimum-password-requirements-unmet-found ?password))
    (at end (increase (total-impact) 8))
    (at end (increase (total-duration) 5))
    (at start (operator-busy))
    (at end (not (operator-busy) ) )
  )
)
```

Figure 4: Example PDDL action

The duration is the approximate time required to assess a particular vulnerability and depends on its complexity level. The conditions are composed of issues that needs to be true for the vulnerability to exist. There are also fixed predicates in each durative-action, called *action-durations* and *operator-busy*, which are used to sum up the duration of each action and to ensure they execute sequentially. The effect explains the damage of a particular vulnerability. It also defines the level of damage in terms of impact value, which is between 1 and 10, where 1 being minimal and 10 being the largest damage.

Table 1 shows the list of all actions, durations and impact levels, extracted from the expert knowledge. The duration of vulnerability assessment action might vary for different systems. The impact level too depends on the sensitivity of data, services and resources of the underlying system.

An example of durative-action is shown in Figure 4, where the purpose is to check if the system has weak password and might be vulnerable to password-cracking attacks. The complete explanation is in the following:

Predicates – The predicates are extracted and derived from the requirements of the vulnerability assessment. The information regarding vulnerabilities is collected from various sources such as books, papers, web articles and expert knowledge. A single vulnerability is represented by one or more predicates. In addition, there are two more predicates, *action-durations* and *operator-busy*. The *action-durations* is used to define a deadline for all actions in the output plan, where the time-limit is given by the user. The *operator-busy* is used to ensure actions are performed sequentially. The total-time of a plan should be less than or equal to *action-durations*, which is used as a timed initial literal to restrict the makespan.

Functions – There are two functions in the domain description: *total-impact* and *total-duration*. The *total-impact* is used to determine the accumulative impact value of all actions in the output. Its value is increased by the impact value of each action. Similarly, *total-duration* is the accumulation of each individual action's duration. By using these functions, the planned is able to select actions that have the great-

Table 1: Transformation of knowledge into Actions, their duration and impact level

#	Action name	Duration	Impact (total 10)
1	Insufficient-password-length	2	8
2	Minimum-password-requirements-unmet	5	8
3	Password-is-guessable	7	5
4	Insecure-password-storage	15	6
5	Password-brute-force-attack	10	4
6	Insecure-forgot-password-option	20	4
7	Insecure-single-factor-authentication	8	5
8	Infeasible-authentication-scheme	20	8
9	Access-vulnerability-File-system	25	7
10	Access-Control-Authorisation-vulnerability	30	5
11	Unmanaged-permission-of-application	10	7
12	Applications-might-be-outdated	15	9
13	Lack-of-network-firewalls	10	8
14	Lack-of-maintenance-in-network-firewalls	7	6
15	Default-Configuration-network-firewall-might-be-useless	10	5
16	Each-Node-Should-Have-Personal-firewall	9	4
17	System-Vulnerable-to-Denial-of-service-attack	5	8
18	Network-connection-events-logging-are-not-enabled	8	5
19	Access-control-events-are-not-auditable	10	6
20	Security-events-are-not-auditing	5	7
21	Data-at-rest-should-be-Encrypted-Privacy-in-danger	5	9
22	Data-at-motion-should-be-Encrypted-Privacy-in-danger	5	9
23	Feasible-encryption-algorithm-not-used	8	7
	Total	**249**	**150**

```
(:init
  (No-Uppercase-Letters password)
  (No-Lowercase-Letters password)
  (No-Numbers password)
  (No-Symbols password)
  <...>
  (= (total-impact) 0)
  (= (total-duration) 0)
  (at 0 (action-durations))
  (at 10 (not (action-durations)))
)
(:goal (and
  (<= (total-duration) 10)
))
(:metric
  maximise (total-impact)
)
```

Figure 5: Example PDDL problem definition

est impact within the available time frame specified through the Timed Initial Literal.

Durative-action – The purpose of using *temporal* actions is to model the time needed to execute a vulnerability assessment action. The action presented in Figure 4 illustrates that a vulnerable password would lack upper and lower-case letters, numbers or symbols. The condition also contains the negation of *effect* because our problem file does not have any explicit goal. Without this, we would have duplicate actions in the plan. The *action-durations* and *operator-busy* are used to limit the number of actions and remove their concurrency respectively.

Effect – Figure 4 shows that any password having the aforementioned issues does not meet the minimum requirements of a strong password, hence it is viable to password-cracking attacks. It states the impact level of the vulnerability as well as identifying the possible amount of damage it can cause, which is 8 out of 10 in this particular case. It also increases the accumulative impact and duration value (*total-impact* and *total-duration*). In adition, the effect starts by stating the *operator-busy* predicate, so that no other action can be executed at that instance. Once the action is completed (at end), *operator-busy* is reverted to its original state, freeing the lock and the planner proceeds to next action.

Problem Description

A sample part of problem file is also shown in figure 5. It contains the initial state, goal state and metric descriptions. The complete explanation on the construction of problem file is in the following:

Init – Describes the complete initial state of system in terms of properties such as the password has no lower or upper case letters and many others. The initial state is a collection of all predicates the represent the system under examination that can subsequently be used for vulnerability assessment. The objects represent the constant names of assets under assessment. For example, *password* is an object, whose strength level is rated in the aforementioned domain description.

```
0.0003: (FEASIBLE-ENCRYPTION-ALGORITHM-NOT-USED
                        ENCRYPTIONALGORITHM)

8.0005: (DATA-IN-MOTION-SHOULD-BE-ENCRYPTED-PRIVACY-IN-DANGER
                        DATAINMOTION)

13.0007: (DATA-AT-REST-SHOULD-BE-ENCRYPTED-PRIVACY-IN-DANGER
                        DATAATREST)

18.0010: (SYSTEM-VULNERABLE-TO-DENIAL-OF-SERVICE-ATTACK
                        SYSTEM)

23.0012: (MINIMUM-PASSWORD-REQUIREMENTS-UNMET
                        PASSWORD)
```

Figure 6: Example PDDL plan output

Goal – Notice there is no explicit goal to reach as we want all of those actions, which should be completed within the given deadline. Thus, we only check whether the accumulative value of duration, *total-duration*, is less or equal to deadline-value (which is 10 in this case). Once the condition is satisfied, the planner should stop generating the plan, even if there can be more actions.

Metric – The requirement for our system is to output those actions, which have the maximum impact and can be completed within deadline. For maximising the impact, we have used *maximize* feature on *total-impact* function. It will enable the planner to only select the most feasible action with respect to their impact value, if various options become available within limited time.

Planer output

A sample plan is shown in figure 6. It took *1 second* to search this plan on *Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz* of processor with *16GB RAM*. The operating system was *32-bit Ubuntu Kylin*. The deadline was specified as 30 minutes and the planner output provides a plan with the total duration of *28* minutes. The plan shows VA tasks that should be conducted to evaluate the security. The planner was executed several times *(-n 10)* to reach a plan better quality. If the plan output file is provided to the post processor (figure 3), it can elaborate the plan in further details. The post processor is a primitive application which matches the actions against pre-determined sentences to provide more detail instructions with context for the user.

Experimental Results and Analysis

This section presents the results of proposed solution and evaluate them to demonstrate the advantages in terms of usability and applicability of this technique. Using the same domain file, we provided the planner with various problem files, all of them having different time constrained deadlines, ranging from 10 to 250 minutes in duration. For each deadline, the planner was executed more than once, i.e. until it could not find any better plan within the specified 5 minutes of cut-off time. The results of various plans are shown in Figure 7. The x-axis shows the total number of actions, while the y-axis shows the deadlines and accumulated impact value of each plan. Furthermore, planner is displaying the correct results as the output matched our expected output that was derived manually.

The number and impact value of the actions are directly proportional to the deadline. This feature significantly maximises the efficiency of VA process, as more important vulnerabilities can be identified within specific amount of time. Furthermore, the total value of duration in domain file is 249 and the total number of actions are 23 (shown in Table 1). With the deadline of 250 minutes, which is more amount than the sum of all durations of actions, the planner outputs all 23 actions as expected. It means that the full potential of solution can be used if user has enough time. It is also observed that some different deadlines (e.g. of 60 and 70 minutes) present the same amount of actions (10), but with the different impact values (72 and 76 respectively). This proves that our resultant plan will try to maximise the overall impact, while choosing only crucial and minimum number of actions. Hence, the user will be able to detect important vulnerabilities in a shorter time span and protect the system against common, yet harmful attacks.

Potential advantages

Our solution is beneficial for both experts and non-experts. It should be noticed that the solution is not supposed to replace human experts, but assist them in a useful, resourceful and practical way. Following are the benefits that our solution can provide in terms of vulnerability assessment.

Cost – One company charge 1495-USD for single vulnerability assessment of an IT infrastructure with up-to 100 individual internal Internet Protocol (IP) addresses or nodes and takes minimum of two-weeks. But with our solution, any company or individual can get the vulnerability assessment free of charge, within a significantly lower timeframe.

Less effort and more productivity – As the planner automatically outputs the VA tasks, there is no effort required to conduct tiresome searching to find suitable techniques and results in reduction of time, without compromising the quality. Also, the plan itself allocates an appropriate amount of time for each assessment task, hence the whole VA process becomes systematic, precise and efficient.

Quality and effectiveness – The quality of solution depends on the quality of knowledge in domain model. The knowledge of our solution is collected from renowned research outlets and experts. So, the solution is capable of mimicking human expert abilities, hence making it somewhat equally effective.

Feasibility – There are some cases where a company does not want to utilise 3rd party contractors or outsourced vulnerability assessment operations. It is possibly that due to lack of access to experts, the company are paranoid of exposing their private data and system configuration. Using this solution, one can perform in-house VA processes on-demand.

Decide custom time frame – Generally, the VA process is performed on monthly, quarterly, semi-annually or annually basis. But, with our solution at hand, there is no restriction of predefined schedule. Furthermore, one can decide their own custom time-frame and obtain the list of VA tasks in a specified time window.

Scope of domain model – The domain models are manually defined and based on human knowledge. Although this paper focuses on a specific aspect of cyber security, the domain model can describe any number of techniques and methodologies from multiple areas simultaneously. It is just a matter of modelling the knowledge into domain file. Hence, it would not be wrong to state that our proposed system can comprehend the knowledge of multiple human experts inside a single domain model and provide a better and holistic plan, as well as strategy by considering various areas of cyber security.

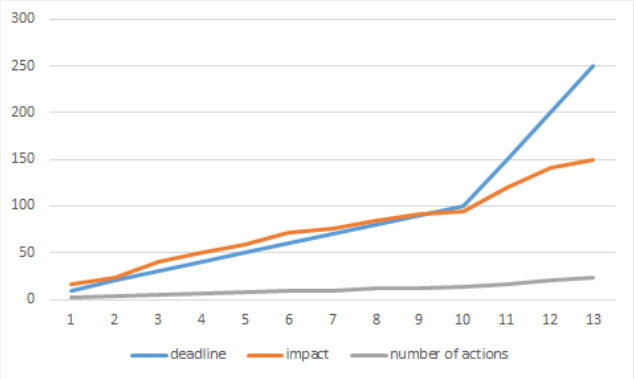


Figure 7: Relationship between different deadlines, and their impact and number of actions. It shows that (deadline durations $\propto$ number of actions & accumulative impact).

Conclusion and Future work

We have shown in this paper that vulnerability assessment techniques can be modelled into planning problem, where they are solved more efficiently along with integrating new functionality such as time constraints. The proposed solution, though currently work-in-progress, has shown significant initial results in aiding the non-expert users to conduct vulnerability assessment tasks own their own. This paper discussed the complete design of proposed solution and the details on how to transform the expert knowledge to the planning domain. The results successfully depicts that it is not necessary for a non-expert to rely on others. The plan itself can inform the user in manually performing comprehensive VA process. The domain model is created from expert knowledge and hence the VA procedures mimic the abilities of an expert as well. By using our domain model, planners such as *LPG-td* and user deadline requirements, optimal plans can be generated based on their threat level and time duration.

As our final goal is to produce a real-time automated VA solution, the following important questions remains as regards to future work. First, how to increase the quality and quantity of knowledge in domain models? It essentially leads towards incorporating better knowledge acquisition techniques from the experts. Second, how to deal with incomplete knowledge? One possible solution would be using probabilistic planning techniques that can generate discrete actions. Last but not least, how this solution can become more beneficial and usable? It can be done by transforming the plan into executable commands in accordance with the underlying system.

References

Barrere, M.; Badonnel, R.; and Festor, O. 2014. Vulnerability assessment in autonomic networks and services: a survey. *IEEE communications surveys & tutorials* 16(2):988–1004.

Benton, K.; Camp, L. J.; and Small, C. 2013. Openflow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 151–152. ACM.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *ICAPS*, 12–21.

Edelkamp, S., and Hoffmann, J. 2004. Pddl2. 2: The language for the classical part of the 4th international planning competition. *4th International Planning Competition (IPC04), at ICAPS04*.

Fox, M., and Long, D. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)* 20:61–124.

Gerevini, A.; Saetti, A.; Serina, I.; and Toninelli, P. 2004. Lpg-td: a fully automated planner for pddl2. 2 domains. In *In Proc. of the 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04) International Planning Competition abstracts*. Citeseer.

Hoffmann, J. 2003. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Hoffmann, J. 2015. Simulated penetration testing: From" dijkstra" to" turing test++". In *ICAPS*, 364–372.

Holm, H.; Sommestad, T.; Almroth, J.; and Persson, M. 2011. A quantitative evaluation of vulnerability scanning. *Information Management & Computer Security* 19(4):231–247.

Kamongi, P.; Kotikela, S.; Kavi, K.; Gomathisankaran, M.; and Singhal, A. 2013. Vulcan: Vulnerability assessment framework for cloud computing. In *Software Security and Reliability (SERE), 2013 IEEE 7th International Conference on*, 218–226. IEEE.

Khan, S., and Parkinson, S. 2016. Towards a multi-tiered knowledge-based system for autonomous cloud security auditing. In *Proceedings of the AAAI-17 Workshop on Artificial Intelligence for Cyber Security (AICS)*. AAAI.

Maynor, D. 2011. *Metasploit toolkit for penetration testing, exploit development, and vulnerability research*. Elsevier.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl-the planning domain definition language.

Obes, J. L.; Sarraute, C.; and Richarte, G. 2013. Attack planning in the real world. *arXiv preprint arXiv:1306.4044*.

Owsap. 2017. Vulnerability scanning tools.

Parkinson, S., and Crampton, A. 2016. Identification of irregularities and allocation suggestion of relative file system permissions. *Journal of Information Security and Applications* 30:27–39.

Rahman, A. F. A.; Ahmad, R.; and Ramli, S. N. 2014. Forensics readiness for wireless body area network (wban) system. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, 177–180. IEEE.

Riabov, A.; Sohrabi, S.; Udrea, O.; and Hassanzadeh, O. 2016. Efficient high quality plan exploration for network security. In *International Scheduling and Planning Applications woRKshop (SPARK)*.

Ristov, S.; Gusev, M.; and Donevski, A. 2014. Security vulnerability assessment of openstack cloud. In *Computational Intelligence, Communication Systems and Networks (CICSyN), 2014 Sixth International Conference on*, 95–100. IEEE.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2013a. Penetration testing== pomdp solving? *arXiv preprint arXiv:1306.4714*.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2013b. Pomdps make better hackers: Accounting for uncertainty in penetration testing. *arXiv preprint arXiv:1307.8182*.

Sarraute, C.; Richarte, G.; and Lucángeli Obes, J. 2011. An algorithm to find optimal attack paths in nondeterministic scenarios. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 71–80. ACM.

Shah, S., and Mehtre, B. M. 2015. An overview of vulnerability assessment and penetration testing techniques. *Journal of Computer Virology and Hacking Techniques* 11(1):27–49.

Shmaryahu, D. 2016a. Constructing plan trees for simulated penetration testing. In *The 26th International Conference on Automated Planning and Scheduling*, 121.

Shmaryahu, D. 2016b. Constructing plan trees for simulated penetration testing. In *The 26th International Conference on Automated Planning and Scheduling*, 121.

Sohrabi, S.; Riabov, A.; Udrea, O.; and Hassanzadeh, O. 2016. Finding diverse high-quality plans for hypothesis generation. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*.

Sohrabi, S.; Udrea, O.; and Riabov, A. V. 2013. Hypothesis exploration for malware detection using planning. *Edited By: Nicola Policella and Nilufer Onder* 29.

Umrao, S.; Kaur, M.; and Gupta, G. K. 2012. Vulnerability assessment and penetration testing. *International Journal of Computer & Communication Technology* 3(6–8):71–74.

Webb, E. M.; Boscolo, C. D.; and Gilde, R. G. 2016. Network appliance for vulnerability assessment auditing over multiple networks. US Patent App. 15/079,224.

Zhao, J., and Zhao, S. Y. 2015. Security and vulnerability assessment of social media sites: An exploratory study. *Journal of Education for Business* 90(8):458–466.

Coverage Planning for Earth Observation Constellations

Evridiki V. Ntagiou[1], Claudio Iacopino[2], Nicola Policella[3], Roberto Armellin[1], Alessandro Donati[3]

[1]Surrey Space Centre, University of Surrey, Guildford, UK
Email: {e.ntagiou, r.armellin}@surrey.ac.uk

[2]Surrey Satellite Technology Ltd, Guildford, UK
Email:c.iacopino@sstl.co.uk

[3]ESOC, European Space Agency, Darmstadt, Germany
Email:{nicola.policella,alessandro.donati}@esa.int

Abstract

Earth Observation market has been increasing in both size and complexity over the last years. EO missions are becoming more capable and more agile, carrying high-resolution sensors that need to frequently be steered at different directions depending on the mission goals. In this paper we discuss the Coverage Planning problem Disaster Monitoring Constellation (DMC3) mission deals with. It is an Earth Imaging mission from Surrey Satellite Technology Ltd (SSTL). The combinatorial optimization problem of determining a not only feasible but optimal sequence of the spacecraft attitude in order to image the total of a target area is *NP*-hard. We propose an automated planning system for DMC3, employing a self-organizing software architecture and a nature inspired optimization algorithm, Ant Colony Optimization. The advantages of the system are discussed and some key results are shown.

Introduction

Planning the operations of an Earth Observation satellite is the process of determining *which* available tasks the satellite will perform and *when* these will take place, as the available resources, samples' collection goals, weather conditions and user requirements evolve. Agile EO spacecraft orbit the Earth, and gather information by slewing their sensors towards areas of interest. Automating the process of finding feasible schedules for EO satellites has recently been the objective of several researchers in both academic and industrial environment. (Chien et al., 2012) The difficulty of the underlying combinatorial optimization problem initially depends on the satellite characteristics, constraints and the planning horizon. The complexity increases even further when accounting for the highly dynamic nature of the problem, due to goals that might be redefined, weather conditions that can change, emergency cases that might strike.

In this paper, we study the planning and scheduling of imaging requests submitted to SSTL's Disaster Monitoring Constellation 3 (DMC3) mission. It is an Earth Imaging mission of low cost small satellites, providing images for several applications, commercial or of public interest, on a daily basis. The platform consists of 3 high-resolution optical Earth imaging agile spacecraft flying in a 650km Sun Synchronous orbit, which can be steered up to 45° off-nadir pointing along the roll and pitch axes. Their slewing ability results in a very big Area of Regard in the surface of the Earth, but also increases the complexity of the planning process. The objective of this work is to find schedules for this mission, when the Areas of Interest considered cannot be entirely imaged with a single pass.

Previous research considers the selection of swaths for a fixed nadir pointing spacecraft. (Cordone R. et al., 2006) A Langrangian heuristic and a subgradient optimization method were used, producing very good solutions. When the assumption of fixed pointing is removed, the main approaches found in the literature involve a greedy (Muraoka et al., 1998) and a GRASP algorithm applied to an Integer Linear Programming model. (Galan-Vioque et al., 2011)

In this paper, we aim at applying a Swarm Intelligence (SI) algorithm to the design of an automated ground based Mission Planning System (MPS). Our work is based on a previous research in which a SI method, Ant Colony Optimization (ACO), was applied to Mission Planning for EO Constellations where the planning problems could be represented as *binary decision* problems (Iacopino et al., 2013). Generalising the approach, we study the application of ACO in planning problems of higher complexity, which

involve *multiple decision* problems – at each decision step multiple options exist rather than two.

Coverage Planning Problem

The Coverage planning problem consists of finding a way to *cover* all the parts of an area of arbitrary shape. In robotics, covering an area translates to visiting all of its points, thus a motion path has to be found for the robot. In Earth Imaging applications, the satellite is expected to image the total of the area. When agile spacecraft are considered, a plan of the attitude maneuvers has to be decided. Regardless of the field of research, coverage planning is an *NP*-hard combinatorial optimization problem. (Strimel G.P. et al., 2014).

More specifically, DMC3 constellation is expected to be able to image a ground area of around 1 million square km in the surface of the Earth, each day. The satellites are in a 650km sun-synchronous orbit and operate under three imaging modes, but the most commonly used is the *strip mode*. In this mode, the satellite captures long *strips* of orthogonal shape (*Figure 1*). This is due to the "push broom" form of imaging employed, i.e. the camera is lying across the orbit ground track. The satellites are agile, thus can be steered up to 45° off-nadir pointing along the roll and pitch axes. For this paper we consider only *roll* axis steering. Strips consist of smaller size images of square size 23 by 23 square kilometers, called *scenes*, which can be imaged within the roll capability range of the satellites. The width of a scene increases as the attitude of the spacecraft diverges from nadir, with a maximum width of about 44km. Power and thermal constraints limit the maximum strip length to 175 scenes. The spacecraft have 2 data recorder devices: a small one of a total memory consisting of 30 GB meant to be used for near real time imaging, and one large device of a total memory of 512 GB meant to be used in a stored and forward manner. The size of a single image i.e. a scene, can vary depending on the attitude of the spacecraft, the location and compression used, from about 250MB to 1300MB. The frequency of Ground Station passes is as high as once per orbit. Each strip imaged is firstly stored on the on board memory, limiting the rest of the images that can be acquired before a downlink. The satellites download data with a rate of 350 Mb/s, thus their memory can be emptied after a pass with duration of at least 616s. Ground Station passes' duration can vary between 2-10 minutes.

The planning problem we consider is: given an Area of Interest (AoI) of arbitrary shape, a planning horizon, a single spacecraft orbit and pointing availability, and Ground Station passes, the *objective* is to maximize the area that can be imaged, while respecting the following constraints:

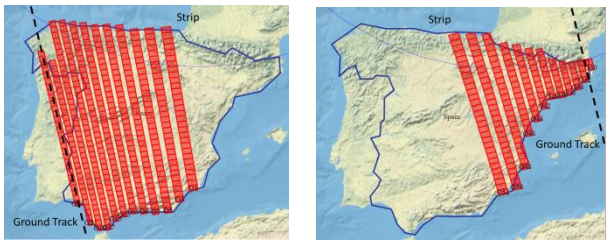


Figure 1. Two passes over the same AoI, with different ground tracks, and their imager slew angle options.

- *Temporal constraints*. They regard the sequence of the spacecraft' imaging opportunities of the AoI and the Ground Stations. The orbit of the spacecraft defines the order of the imaging opportunities, which dictates the order of the Nodes in the graph representation of the problem shown in *Figure 2*.
- *Resource constraints*. For the on-board storing, we consider the use of only the small data recorder device. The resource constraints are satisfied by updating the current schedule once a newly added imaging opportunity results in memory overflow, as described in the Algorithm section. Another part of the resource constraints is defined by the Ground Stations' availability. DMC3 can serve multiple Ground Stations but their number and position on Earth are not considered in this paper. We only regard the Ground Station passes as on board memory renewals. Also, Ground Stations are not guaranteed to be available during the time of a pass. Their availability is subject to changes due to the workload or emergency cases, thus they are regarded as unreliable resources from a planning point of view. This unreliability can lead to having fewer Ground Station passes than initially expected, resulting in increased severity of the resource constraint. In such cases, the system should adapt to this change.
- *Spatial constraints*. Given that the ground track of the spacecraft is not the same every time it passes over an AoI, it is certain that some acquisitions will overlap with others. This will not only hinder or delay the total coverage of an area, but will also result in unnecessary use of the spacecraft imaging sensor. Overlap cannot be considered as a hard constraint of the problem, since a solution will still be feasible even when the images overlap. It is though highly desirable to keep it at minimum levels, especially when the time horizon requested by the user for the completion of the task is not long.
- *Imaging constraints*. The *strip mode* of imaging if considered. The spacecraft image strips of

orthogonal size of the AoI in each imaging opportunity. (*Figure 1*)

The operational requirements form the objectives for a mission planning system to perform well, and cannot be overlooked. An automated MPS, especially, needs to cope with the following requirements:

- **Reliability**. A feasible solution must be provided *at all times*. The MPS has to be designed in a way that a high level of system responsiveness is preserved.

- **Scalability**. An increase in the number of users, AoIs or spacecraft results in a corresponding burst in the complexity of the planning problem. Hence, the MPS needs to be scalable to the input size, preserving its performance and usability.

- **Adaptability**. Given that the satellites' availability and the users' preferences might be redefined or an emergency situation might arise, we desire a system that will be able to adjust its behaviour, in response to environmental changes.

In the next section we discuss the potential of a Swarm Intelligence algorithm in dealing with these requirements.

Planning with Stigmergy

For an MPS that copes with all the aforementioned requirements we employ a nature inspired meta-heuristic method, Ant Colony Optimization. ACO is a probabilistic algorithm used to find the solution in Computer Science and Operations fields' problems that can be reduced to finding optimal paths in graphs. (Dorigo & Stutzle, 2004) Real world ant colonies are able to find the shortest paths between their nests and a food source. They do that using no direct communication with each other. All the individuals follow simple sets of rules, and none has universal knowledge of the colony's actions. Nevertheless, the colony does have a complex behavior, which is the result of interaction with the environment. This mechanism is called *stigmergy*, a means of indirect coordination of a number of individuals, through their environment.

The basic principle is that the traces left in the environment by an agent's actions stimulate the next agent's actions. Stigmergy is a form of *self-organization*. Complex, seemingly intelligent structures are produced, without need for any control, using only indirect communication. Thus, the agents usually are designed to be simple, without intelligence, memory, or awareness of one other.The same principles apply to ACO. The basis of ACO algorithms is summarized below:

When the ants are searching for food, in the natural world, they first wander randomly. After finding a source of food, they return to their colony laying down pheromone in the path that they follow. If other ants find such a trail they are less likely to continue their wandering, but follow the trail instead. In case it leads them to food, they will also reinforce it upon their return to the colony. The pheromone trails start to evaporate over time. Hence, the pheromone will eventually be gathered in the shorter paths, which get marched over more frequently.

The deposit mechanism helps the colony find a good solution whereas the pheromone evaporation is a means of avoiding convergence to a locally optimal solution. A typical ACO algorithm involves three main steps:

- *Path Construction*: The ants construct a path that includes all the vertices of the graph. They do that by iteratively adding an edge to the path, after they chose it based on its amount of pheromone. The probabilistic rule used favours the edge with the highest pheromone amount:

$$P_{i,j} = \frac{\tau_{i,j}^{\alpha}}{\sum_{j=1}^{M_i} \tau_{i,j}^{\alpha}} \quad (1)$$

  where $\tau_{i,j}$ is the amount of pheromone in the edge j of vertex i and M_i is the number of incoming edges in the vertex i. The parameter α defines the importance of the pheromone intensity in the ant's choice of edge. $P_{i,j}$ is the probability of edge *(i,j)* to be chosen. In the system, it is implemented as a roulette wheel selection process.

- *Path Evaluation*: Once at the end of the graph, each ant evaluates the path they created, based on an objective function, *f*.

- *Update of pheromone field*: The update takes place in two steps. First each ant deposits an amount of pheromone to the *path* it constructed, based on its evaluation. Then, the pheromone of the whole *graph* is evaporated by a certain rate, ρ.

Problem representation

ACO is a graph search optimization method; the graph gives structure to the search space. It is a critical part of the optimization process as it provides the environment. To that respect, it should be easy to build and traverse while also being representative of the problem. Our choice of representation is a *directed graph* that follows these guidelines.

During each pass over the AoI, the attitude of the spacecraft needs to be determined. Since there is a range of

roll angles that allow it to point to the AoI, there is a choice of attitude to be made, for each single pass. The problem is represented in the system by a directed graph *G(V,E)*, where the Nodes represent imaging opportunities of the AoI, and the edges the spacecraft attitude. More specifically:

- *V* is the set of vertices or *Nodes* of the graph. There are two types of Nodes:
  - *N* Nodes that represent a pass of the spacecraft over the AoI. In these Nodes, the memory resource is consumed.
  - *G* Nodes that represent a pass over a Ground Station, or downlink availability. In these Nodes, memory resource is renewed.
- *E* is the set of all the Edges of the graph, each one representing a spacecraft attitude. Each *N* Node has a set of incoming edges to it, each corresponding to a roll angle choice for this pass. For all the *N* Nodes there is an extra edge representing the option of no image capturing during this pass.

The order of the Nodes is the corresponding chronological order of the imaging opportunities defined by the orbit, as shown in *Figure 2*. Each edge is associated with two values: $\theta_{i,j}$ for the roll angle it represents, and $m_{i,j}$ for the memory the corresponding strip consumes. Given its form, we refer to this *directed graph* representation as the *N*-ary chain, due to the arbitrary number (*N*) of edges that are incoming to each Node.

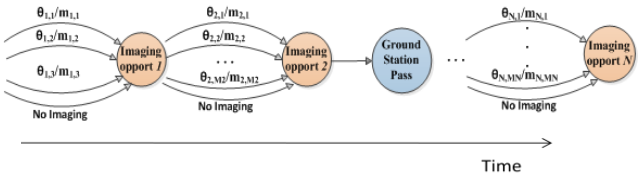


Figure 2. An N-ary chain representation of the coverage planning problem.

In the case of many independent AoIs requested to be imaged, the graph includes the passes and corresponding imaging opportunities of all of them in a chronological order.

Algorithm

All search algorithms involve making a choice at each step: to either *exploit* or *explore* the search space. The first suggests that the search will be directed towards already visited regions of the search space, whereas the second means that more information will be gathered regarding its unexplored regions. Balancing the two phases is important for the success of Evolutionary Computation and Swarm Intelligence algorithms. (Crepinsek, Liu, & Mernik, 2013). Several mechanisms have been proposed to this direction:

- The deposit and evaporation mechanisms are forms of exploitation and exploration respectively. (Dorigo & Stutzle, 2004)
- Pheromone deposit function.
  - Having a minimum and maximum amount of pheromone added. (Stutzle & Hoos, 2000)
  - Allowing only for "elite ants" to deposit pheromone (Bullnheimer, Hartl, & Strauss, 1999)
- Probabilistic edge selection function.
  - Random edge selection in parallel to the selection that is guided by the pheromone field and heuristic value. (Nakamichi & Takaya, 2004)
  - Tuning parameter alpha when a probabilistic rule of the form (1) is used. (Meyer, 2004)

In this system the alpha parameter control approach is used. In (Iacopino et al., 2013) it was proven that for probabilistic rules of the form (1) parameter *alpha* controls the balance between exploration and exploitation in ACO algorithms. The critical value around which the colony's behavior changes is alpha = 1. The algorithm workflow is summarized in the following:

```
1: PheromoneInitialization();
2: for all ant do
3: for all node do
4:        path+=TransitionRule();          //Path Construction
5:        feasibilityCheck(path);
6: end for
7: phDep=ObjectiveFuntion(path);           //Path Evaluation
8: update(phDep);                    //Pheromone field Update
9: update(phEvap);
9: updateAlpha();                                //Alpha Update
10: if convergence then
12:       savePath();
12:       restartAlpha();
13: end if
14: end for
```

Through the *feasibilityCheck* method, we make sure that each path that is evaluated is feasible i.e. consumes less memory than the one available on board. Each time a new edge is added, the total memory is computed. In case there is a constraint violation, some edges are deleted from the path, and replaced by the corresponding 'No imaging' edges. The selection is made by a weighted roulette wheel which includes all the edges on the path, based on the pheromone levels and memory they consume.

The evaluation of the path includes a dynamical scale that adjusts to the best current solution, in order to

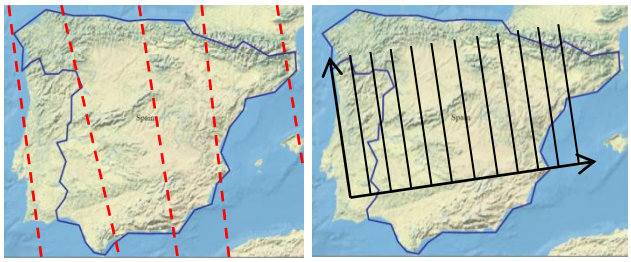
determine the amount of pheromone that will be deposited. The objective function we use in this optimization problem is:

$$f = \max\{\text{Coverage}\} \quad (2)$$

The key part of the algorithm is the *updateAlpha* method, or the way we balance between exploration and exploitation phases. Alpha takes values around 1 in [0.5, 2], and is changing based on a non-decreasing function. By increasing the value of alpha, we increase the effect of the pheromone field in the ants' choice. An exploration/ exploitation cycle thus happens by starting with a value below 1, and reaching a value above it, depending on when the convergence criteria is considered reached. Restarting the cycle multiple times, we give the system the ability to integrate any changes of the environment, like the addition or deletion of graph nodes and edges, or the change in the available resource. It is a way of increasing the system's *adaptability*.

Coverage calculation

Computing the size of the imaged area can become a bottleneck for the performance of the system. The underlying geometric problem involves the calculation of the size of each strip's intersection with the AoI, subtracting the areas covered by two or more strips (overlap). This calculation is computationally too expensive to take place many times during the planning. We estimate the coverage employing a simpler algorithm. The calculation is based in one assumption: given that DMC3 is imaging only in ascending node, for a sufficiently small Area of Interest the Earth's curvature is small enough to assume that the ground tracks of the spacecraft are almost parallel to one another (*Figure 3a*). Thus, we assume that the orientation of the all the strips in the search space will be roughly the same.



(a) *(b)*

Figure 3. (a) Ground Tracks of satellite passes over Spain for a week (b) Division of AoI in bands

In Figure 3a we show the Ground Tracks over the area of Spain for one spacecraft and a period of one week. The average inclination with respect to a Cartesian frame, for this example, is 6º. The average difference among the ground tracks inclinations is smaller than 1º. Our assumption, thus, is acceptable.

The calculation involves dividing the AoI in bands parallel to each other, as shown in Figure 3b, with a band-frame orientation equal to the average orientation of all the ground tracks considered, that is 6º with respect to the Cartesian frame. For sufficiently narrow bands, if two strips intersect with the same band, they will most likely overlap each other. We maintain and update two tables, each with size B, equal to the number of bands considered. In the first table we keep the number of strips intersecting each band. This process can be reduced to a binary search of the position of each strip's starting and ending longitude coordinates, with respect to the positions of the bands. The second table has the maximum and minimum latitude values of all the strips intersecting the band. Objective function (2) becomes:

$$f = \max\{latCov + longCov\} \quad (3)$$

where $latCov = \sum_{i=1}^{B}(lat[i]_{max} - lat[i]_{min})$ is the latitude coverage estimation, B is the total number of bands, $lat[i]$ is the latitude – maximum or minimum – per band and $longCov$ is the longitude coverage estimation, equal to the number of bands that intersect with at least one strip. Thus, the more the overlap among the strips, the more strips each single band intersects with and the smaller the value of $longCov$. The bands' width is chosen with respect to the swath width. We usually select a width equal to a quarter of the nadir pointing swath width, given the tradeoff between computation time and accuracy.

Considering priorities

The current system can also accommodate requirements that can be described through *priorities*. These can include weather conditions, user priority, etc. In such cases, the objective function will become:

$$f = \max\{\text{Coverage} + \sum p_{ij}\}$$

Priorities are integrated in the system by adding a priority value p_{ij} in the corresponding imaging opportunities, or adding a *weight* value on the edges of the graph. They can be *user* specific, and *area* specific. In the first type, the acquisitions associated with a specific user are assigned with the corresponding priority value. The area specific priorities, involve imaging a subarea of the AoI with higher priority. For example while imaging Spain, the capital of the country, Madrid, might need to be fully covered or in the case when we are aware that a lot of clouds will be gathered in some area, we choose to avoid imaging it during this time.

Simulation results

In our software prototype, a user submits their imaging request and preferred time of task completion, along with more specific requirements such as subareas with higher priority, or weather conditions that can render an image unacceptable. Their requests are asynchronous and can change or be filed anew at any time. SSTL's orbit propagator (Wu, Brewer, & Palmer, 2002) then produces the imaging opportunities of the spacecraft which is the input of the system, and the user's preferred time of completion constitutes the planning horizon. The available on board memory and set of Ground Station passes during this horizon are also imported to the system, and the graph environment is formed. The output is a feasible schedule of the imaging sensor tilts for each spacecraft.

We test the system with two problems of different difficulty, considering a single AoI imaging request. In this paper, we define the difficulty by the size of the AoI and the planning horizon, since these two factors determine the size and topology of the *N*-ary chain. The size of the AoI is defined by the ratio of its longitude range over the nadir pointing swath width of the spacecraft e.g. for a small AoI this ratio is less than 10, whereas an AoI with a ratio bigger than 30 is considered of big size. The first test regards a small AoI which can be seen in blue outline in Figure 5, and a planning horizon of up to 10 days. The optimal solution in this case does not take too long to calculate with exhaustive search, therefore a comparison between ACO solution and optimum can be made. In the exhaustive search algorithm, the search space is pruned by not considering a path which exceeds the memory resource constraint once one is found.

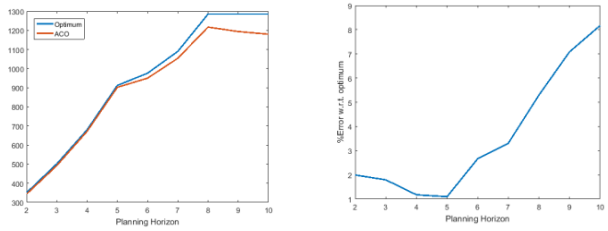
For the given AoI, we increase the planning horizon – thus increasing the problem difficulty – and note the performance of the system. Increasing the time horizon by one day is usually equal to an increase of the graph representation by one Node – if the AoI is in the field of view of the spacecraft during that day – but a multiplication of the search space by a factor equal to the number of incoming edges to the new Node. For example, adding a new Node to the graph with 9 edges translates to a search space that is 9 times bigger. In this example, the graph representation includes up to 10 Nodes with 2-8 incoming edges each.

Depending on which axis we are more interested in covering we can use weights in Objective function (3) and adjust them accordingly:

$$f = \max\{k_1 latCov + k_2 longCov\}$$

Increasing k_1 we include a bigger number of longer strips in the solution, which can result in the increase of overlap. Based on our assumption regarding the orientation of the strips, we can avoid overlap by increasing the importance of the longitude coverage, k_2, since $longCov$ table includes the overlap value in its calculation. Thus, the higher k_2, the smaller the tolerance to overlap. Depending on the AoI and the specific time horizon though, the system will choose the option of no imaging in some passes, in an effort to find a solution with no overlap. This can pose the risk of a very poorly imaged AoI. For the rest of the paper, we set k_1=k_2=1.

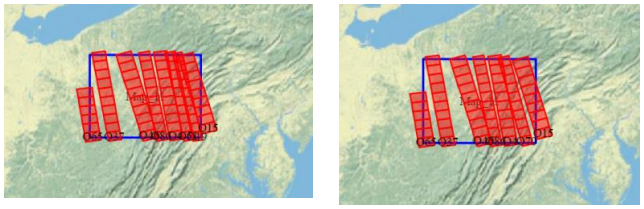
The number of ants used in the simulations is chosen empirically based on experience of the system performance for different input sizes. In general, the quality of the solutions does not improve linearly to the number of ants, but there exists a maximum ant population size after which the solution quality does not improve. For the sake of brevity, the corresponding tests are not shown. For this problem, *3000* ants are used, which corresponds to 3000 objective function evaluations. In Figure 4 we note the error between the solution of the system and the optimum, for an increasing input size.



(a) *(b)*

Figure 4. (a) Optimal solution and ACO system's output (b) Percentage of the error

The error is very small ranging between 1-8%. In order to visualize the quality and differences of the two solutions, we contrast them in Figure 5.



(a) *(b)*

Figure 5. (a) Optimal solution (b) Average ACO solution

We now discuss the *reliability* of the system. In *Figure 6* the evolution of the ACO solution error is shown after multiple *individual* runs, for the longest considered time horizon i.e. 10 days. The error value in both Figure 4 and Figure 6 is a percentage computed with respect to the optimum value:

$$error = \frac{optimum - value}{optimum}\ 100\%$$

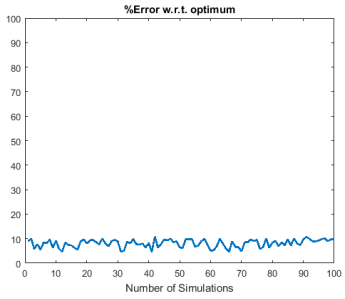
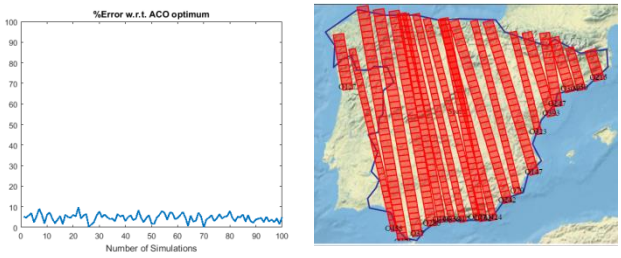


Figure 6. Evolution of the ACO solution error in independent runs.

We see that for 100 different independent runs, the error never increases more than *10%* of the optimum value. The simulations are independent of each other so as to point out the uniformity of the solutions produced.

Next we study a problem of big size in which Spain is the Area of Interest. In order to test the system in increased problem complexity and produce solutions that cover a large percentage of the AoI, we increase the planning horizon to 3 weeks, which translates to a graph of 21 Nodes with 5-45 incoming edges. In each simulation we now use *5000* ant agents. In *Figure 7* we note the percentage of the error between the ACO solutions and the ACO optimum for multiple runs, and visualize the average solution.



(a) *(b)*

Figure 7. (a) ACO error for individual simulations (b) Visualization of average ACO solution

Figure 7(a) description is twofold: first, the system converges to a feasible solution at every simulation. Secondly, it demonstrates that the system steadily provides solutions of similar quality. We now compare the average ACO solution with an algorithm of *greedy* nature, for the same AoI and planning horizon. The algorithm's criteria is the *strip size*; for every pass over the AoI the strip of biggest size is chosen. In Figure 8 we contrast the two results; the percentage of the imaged area when the greedy algorithm is used is 20% smaller than with ACO. This percentage is problem dependent and will vary for problem instances of different difficulty.

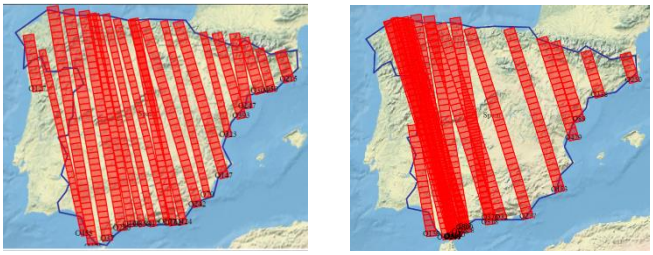


Figure 8. Comparison of ACO average solution with a greedy algorithm outcome for a given imaging request.

Future Work

Multiple objectives

In a MPS the user should be able to define the goals, thus there should be a level of freedom to the metric they can choose to measure the performance of the system.

To that respect, there are many aspects of the mission that need to be considered. In the coverage planning problem, for example, a solution of specific coverage and zero overlap, is considered worse than another solution of the same coverage, with some overlap, which can be completed in an *earlier time*. Another aspect to consider is the quality of the images provided. The bigger the distance from nadir pointing, the more distorted the image, but the wider the strip as well. Decreasing the distortion, might mean longer time to cover an AoI. Hence, we want to take into account multiple mission objectives, which can also be conflicting, and produce the Pareto front of the solution. This will allow for the trade-off between each of the objectives to appear.

Generalizing to diverse missions

The scope of our research is to not only produce a MPS architecture that is mission specific, but use a general enough approach to address planning problems that many missions deal with. To that respect, we have already applied our approach to *oversubscribed scheduling problems* (Ntagiou et al., 2017). This type of problem is often found in space missions e.g. data relay missions.

Scaling to multiple spacecraft

Currently, given that each DMC3 spacecraft has their own orbit and imaging opportunities, we employ 3 separate chains. Parallel implementation will be considered as the chains can be traversed in parallel by different ants, as long as in the objective function evaluation the paths of all the chains are taken into account. In this way, the constellation is regarded as a unity by the system and not 3 different spacecraft. Our future plan is to work on the load balancing among the three spacecraft.

Given the current trend of Earth Observation missions involving hundreds of spacecraft (Buchen, 2015), our focus is also in the scalability of our system, when the number of spacecraft increases that much. To that respect, we aim at producing a coordination mechanism that is irrelevant of the number of spacecraft considered.

Acknowledgements

This work is co-funded by the Surrey Space Centre (SSC) of the University of Surrey, the Surrey Satellite Technology Ltd and the Operations Center of the European Space Agency (ESA/ESOC).

References

Buchen, E. (2015). Small Satellite Market Observations. *29th Annual AIAA/USU Conference in Small Satellites.*

Chandra-Mohan, B., & Baskaran, R. (2012). A survey: Ant Colony Optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications*, 4618-4627.

Chien, S., Johnston, M., Frank, J., Giuliano, M., Kavelaars, A., Lenzen, C., & Policella, N. (2012). A generalized timeline representation, services, and interface for automating space mission operations. *12th International Conference on Space Operations, SpaceOps.*

Cordone, R., Gandellini, F., & Righini, G. (2006). Solving the swath segment selection problem though Langragean relaxation. *Computers and Operations Research*, 854-862.

Crepinsek, M., Liu, S., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR), Vol. 45 Issue 3*.

Dorigo, M., & Stutzle, T. (2004). Ant Colony Optimization. *MIT Press*.

Galan-Vioque, J., Vasquez, R., Carrizosa, E., Flores, C. V., Perea, F., & Crespo, F. M. (2011). Towards a visual tool for swath acquisition planning in multiple-mission EOSs. *International Workshop on Planning and Scheduling for Space (IWPSS).*

Iacopino, C., Palmer, P., Policella, N., Donati, A., & Brewer, A. (2013). Self-Organizing MPS for Dynamic EO Constellation Scenarios. *International Workshop on Planning & Scheduling for Space (IWPSS).* NASA Ames Research Center, California.

Muraoka, H., Cohen, R., Ohno, T., & Doi, N. (1998). Aster Observation scheduling algorithm. *International Conference on Space Operations (SpaceOps).* Tokyo, Japan.

Ntagiou, E., Policella , N., Iacopino, C., Armellin, R., & Donati, A. (2017). Ant Colony Optimization applied to the Planning of a Data Relay Space Mission. *International Workshop on Planning and Scheduling for Space.*

Strimel, G., & Veloso, M. (2014). Coverage Planning with Finite Resources. *International Conference on Intelligence Robots and Systems (IROS).* IEEE/RSJ.

Stützle, T., & Dorigo, M. (August 2002). A Short Convergence Proof for a Class of Ant Colony Optimization Algorithms. *IEEE Transactions on Evolutionary computation, Vol. 6, No. 4*.

Wu, S. -F., Brewer, A., & Palmer, P. L. (2002). ImPredict: a Fast Image Prediction Software and Its Application in the SSTL off-axis Image Scheduling System. *15th AIAA/USU Conference on Small Satellites.*

Domain-independent Metrics for Deciding When to Intervene

Sachini Weerawardhana[1] **Adele Howe**[2] **Mark Roberts**[3]
[1]Computer Science Department, Colorado State University, Fort Collins, CO, USA | sachini@cs.colostate.edu
[2]Computer Science Department, Colorado State University, Fort Collins, CO, USA
[3]Naval Research Laboratory, Code 5514; Washington, DC, USA | mark.roberts@nrl.navy.mil

Abstract

Deciding when to intervene can be as important as recognizing a goal or plan. Intervening too early may waste effort on false positives or provide clues for the next attack, while intervening too late may result in undesirable state (e.g., a stolen file). We build on prior work in plan and goal recognition to study intervention for cybersecurity and benchmark planning domains. Because each application may have distinct needs for selecting interventions, we formulate the problem as a multi-objective planning problem of three domain-independent metrics: timeliness, which captures how soon the undesirable state may occur; certainty, which captures how frequently the undesirable state may be seen; and desirability, which captures the user's preference for continuing the current action despite the increased risk. Against an ideal baseline, we examine trade-offs in choosing the "correct" intervention point by varying the weights associated with these metrics, the observability of actions, and the presence of extraneous actions. We find that timeliness is essential for identifying eminent consequences, certainty and desirability are less sensitive to extraneous actions, and all three metrics are very sensitive to worsened observability. Our study provides a foundation for further work in understanding intervention.

1 Introduction

A wealth of literature in plan and goal recognition has examined how to infer a single agent's plan (e.g., (Geib and Goldman 2009; Ramırez and Geffner 2009)), the agent's goal (e.g., (Ramírez and Geffner 2011; Yin, Chai, and Yang 2004)), or the goals or plans of multiple agents (e.g., (Banerjee, Kraemer, and Lyle 2010; Kaminka, Pynadath, and Tambe 2002)). Some recent work has even examined plan/goal recognition in the face of noisy observations (e.g., (Geib and Goldman 2005; Vattam and Aha 2015)) or extraneous actions (e.g., (Gal et al. 2012; Sohrabi, Riabov, and Udrea 2016)). Yet relatively little of this research considers the question of when to intervene if one wants to thwart the plan or goal if, for example, the agent we want to disrupt is attempting to produce an undesirable state. The decision of when to intervene must be made judicially. Intervening too early may lead to wasted effort chasing down false positives, helpful warnings being ignored as a nuisances, or leaking information for the next attack. On the other hand, intervening too late may result in undesirable consequences.

Our motivating application is a software agent that monitors the state of a personal computer to help home computer users avoid security and privacy vulnerabilities. Home computer users are viewed as the "weakest link" in computer security because they lack the time, knowledge and resources to defend against the increasing incidence of computer security and privacy threats (Sasse, Brostoff, and Weirich 2001). Moreover, some threats, such as software downloads and phishing, require user action or at least acquiescence (Howe et al. 2012). Security analysis for home computer users focuses on identifying threats on a personal computer by modeling attacker actions, the system state of the computer and computer user's actions, including both ordinary and risky actions. The *Personalized Attack Graph* (PAG) extends the attack graph model (Sheyner et al. 2002) to support individual computer/user level granularity and by representing the states and actions in PDDL (Urbanska et al. 2013) Attacker actions do not include actions that cannot be observed on the home computer. System states can include levels of partial compromise (e.g., access to the password key-chain), configuration information (e.g., operating system), and state changes achieved on specific system components (e.g., implanting a keystroke logger).

In this paper, we examine how well an algorithm can determine the best intervention point for the cybersecurity application, calling it a "critical trigger action" because it may lead to undesirable state. As each situation may have a unique definition of the ideal intervention point, we formulate intervention as a multi-objective optimization problem of three domain independent metrics: (1) timeliness, which quantifies how soon the undesirable state may occur, (2) certainty, which highlights frequently occurring actions as important, and (3) desirability, which measures the contribution of the action to user's own objective. The desirability metric helps separate common harmless actions that further the user's actual goal from harmful actions to be avoided. A critical trigger action is a user action that maximizes the linear combination of these three objective metrics.

Given a PDDL domain model, a set of undesirable states to avoid, and an observation trace of actions, our algorithm identifies critical trigger actions. An intervention is correct if, compared to a ground truth trace, the algorithm (1) ignores extraneous actions (i.e., as true-negatives) and, (2) identifies actions leading to undesirable state (i.e., as true-

positives). We begin with a study of traces taken from a human subject study for computer security. We then generalize our results to four benchmark planning domains and consider the impact of missing and extraneous observations of actions and test the algorithm. Across all benchmark domains, certainty and desirability metrics perform well in ignoring extraneous actions, while the timeliness metric and it's combinations with certainty and desirability perform well in identifying true positives. Thus, in this work we have identified two metrics that are sensitive to noise in action based observation traces and a metric that is sensitive to partial observability of actions.

1.1 Example

Before introducing a formal definition, we present an example to clarify the key concepts. Suppose a user wants to read email (e.g., the goal G) but there exists an undesirable state U_1 (e.g., `installation of malicious software`). Figure 1 illustrates the *possible* manifestation of U_1. The user has executed actions $a1, a2, \ldots$ (e.g., `start email application, log in to email account, ...`) from the initial state I to achieve G, resulting in $S_{a1}, S_{a2}, \ldots$.

Let us now consider possible plans that lead to U_1 through actions $b, c, d, f, g, h, j, k, l, m, n, p, q$. In the context of email, these actions might be clicking phishing links, downloading affected files, etc. We will call these *undesirable plans* and denote any such plan as π_{U_1}. Ideally, the user would continue toward achieving G. However, by mistake or undue influence, the user may deviate from pursuing G and unwittingly follow paths reaching U_1. The security agent has observed the action sequence O = $\{a1, a2, a3\}$ executed by the user. In state S_{a3}, the user might reach U_1 through a set of undesirable plans $\Pi_{U_1} = \{(bgn[u_1]), (bgmq[u_1]), (ch[u_1]), (cjp[u_1]), (cjdfl[u_1]), (cjdfkp[u_1]), (fl[u_1]), (fkp[u_1]), (fkdfl[u_1]), (fkdfkp[u_1])\}$ where a plan is a sequence of actions (letters) followed in brackets by the undesirable state that results and each letter $(b, c, d, f, g, h, j, k, l, m, n, p, q)$ denotes candidate trigger actions (an action that appears in at least one undesirable plan). Our question is: *given the current state S_{a3}, which action in the set of actions in Π_{U_1} will be the best choice for the security agent to issue an interrupt if observed?*

2 Problem Statement

We adapt the notation of Ramirez and Geffner (2010) to define the intervention problem. There, a planning domain $D = \langle F, A \rangle$ is a tuple of F, the set of fluents, and A, the set of actions with preconditions and effects that are fluents. A planning problem $P = \langle D, I, G \rangle$ is a tuple consisting of the domain D an initial state $I \subseteq F$ and goal state $G \subseteq F$. We assume that the planning domain and problems are represented in PDDL. Each action $a \in A, a = \langle Pre(a), Add(a), Del(a) \rangle$, consists of preconditions, add and delete effects, respectively. A solution to the planning problem is a sequence of actions where the final state of the plan entails G.

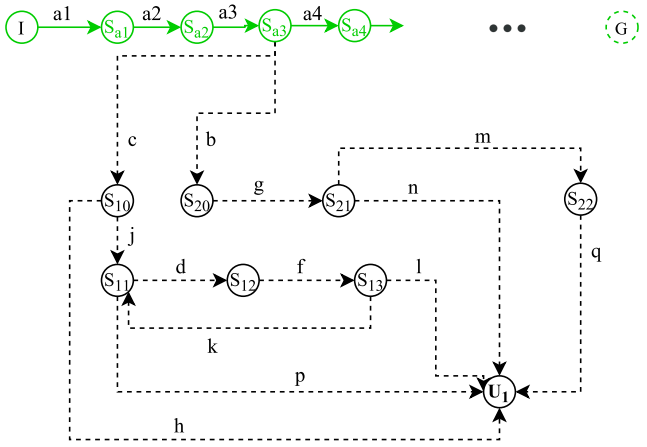


Figure 1: An application domain where a user executes actions $a1, a2, \ldots$ in succession to transform initial state I to a goal state G. An undesirable state U_1 may develop from state S_{a3} after $a3$ has been executed, following dotted paths towards U_1.

An **observation trace** O is a sequence of observed actions $O = \{o_1, o_2, ..., o_n\}$ *where* $o_i \in A$ *and* $i = 1, 2, \ldots, n$ (n = length of observation trace) and states resulting from execution of actions $o_i \in O$ are known.

When we are looking for possible paths leading to undesirable state, we search for an **undesirable plan** $\pi_U = \{a_1, \ldots, a_k\}$ of length k that entails the one or more undesirable states $U \subseteq F$ and $a_i \in A$. There may be more than one undesirable plan, in which case we consider a set of undesirable plans Π_U. In the example in section 1.1, the domain has only one undesirable state $U = U_1$.

An **intervention problem** $T = \langle D, O, I, U, \alpha \rangle$ consists of a planning domain D, a sequence of observed actions $O \subseteq A$, an initial state $I \subseteq F$, a set of undesirable states $U \subseteq F$, and a weight vector α for the objective function, where $\alpha = [\alpha_1, \ldots, \alpha_m]$, α_i is in the range $[0, 1]$, m is the number of objectives and $\sum_1^m \alpha_i = 1$. We will discuss the object function in more detail below.

Traces may contain extraneous or missing actions. An *extraneous action* is an observation $o \in O$ such that the state resulting from executing o is never added to the global state (i.e., set of fluents that are true) by any of the actions $a_i \in \pi_U$. A *missing observation* with respect to π_U occurs when the state resulting from executing o is added to the global state by an of the actions $a_i \in \pi_U$, and $o \notin O$. A **trigger action** is an action $t_i, t_i \in O$, that is found in at least one undesirable plan for a sequence of O observed actions. A **critical trigger action** c_i is a trigger action at o_i that maximizes $V(c_i)$, as defined in section 2.1.

A solution to the intervention problem is a vector of decision points corresponding to actions in the observation sequence indicating whether it was identified as a critical trigger action or not given the observations up to that point in the sequence.

	C	T	D	α = [0.33,0.33,0.33]	α = [1,0,0]	α = [0,1,0]	α = [0,0,1]
				$V(a)$	$V(a)$	$V(a)$	$V(a)$
b	2/10=0.2	max(3/3,4/4)=1.0	-2/39=0.05	0.38	0.2	**1.0**	-0.05
c	4/10=0.4	max(2/2,3/3,5/5,6/6)=1.0	-4/39=0.1	0.43	0.4	**1.0**	-0.1
d	4/10=0.4	max(3/5,4/6,3/5,4/6)=0.6	-4/39=0.1	0.30	0.4	0.6	-0.1
f	6/10=0.6	max(2/5,3/6,2/2,3/3,5/5,6/6)=1.0	-6/39=0.15	**0.48**	**0.6**	**1.0**	-0.15
g	2/10=0.2	max(2/3,3/4)=0.75	-2/39=0.05	0.30	0.2	0.75	-0.05
h	1/10=0.1	max(1/2)=0.5	-1/39=0.03	0.19	0.1	0.5	**-0.03**
j	3/10=0.3	max(2/3,4/5,5/6)=0.83	-3/39=0.08	0.35	0.3	0.83	-0.08
k	4/10=0.4	max(2/6,2/3,4/5,5/6)=0.83	-6/39=0.15	0.36	0.4	0.83	-0.15
l	3/10=0.3	max(1/5,1/2,1/5)=0.5	-3/39=0.08	0.24	0.3	0.5	-0.08
m	1/10=0.1	max(2/4)=0.5	-1/39=0.03	0.19	0.1	0.5	**-0.03**
n	1/10=0.1	max(1/3)=0.33	-1/39=0.03	0.13	0.1	0.33	**-0.03**
p	4/10=0.4	max(1/3,1/6,1/3,1/6)=0.33	-4/39=0.1	0.21	0.4	0.33	-0.1
q	1/10=0.1	max(1/4)=0.25	-1/39=0.03	0.11	0.1	0.25	**-0.03**

Table 1: Certainty (C), timeliness (T), desirability (D) computations for each candidate trigger action for the example in Figure 1. $V(a)$ is the value returned by the critical trigger multi-objective function assuming equal weighting ($\alpha = [0.33, 0.33, 0.33]$), only C ($\alpha = [1, 0, 0]$), only T ($\alpha = [0, 1, 0]$) and only D ($\alpha = [0, 0, 1]$) for each candidate trigger action.

2.1 Objective Function

To quantify when an intervention should occur we define a multi-objective function of three metrics: certainty, timeliness and desirability. These metrics are calculated based on a set Π_U, which supports a form of sampling.

Certainty measures the likelihood of action a occurring in Π_U.

$$Certainty(a|\Pi_U) = \frac{|\pi_U \in \Pi_U \text{ in which } a \text{ occurs}|}{|\Pi_U|} \quad (1)$$

Actions occurring frequently in Π_U indicate the importance of that action toward causing U. For example, if an action a occurs in all plans Π_U, the certainty metric will assign a high value to a, giving it a higher probability of being selected as a critical trigger compared to a less frequent action.

Timeliness requires knowing what actions could yet be observed, which might effect the causation of the undesirable state. For this study, timeliness is measured by the maximum normalized steps remaining in Π_U in which the action a occurs. Timeliness quantifies how soon an observation will trigger the undesirable state. Let n be the remaining number of steps in $\pi_U \in \Pi_U$ after some action a. Then,

$$Timeliness(a|\Pi_U) = \max_{\pi_U \in \Pi_U} \left(\frac{\max(n)}{|\pi_U|} \right) \quad (2)$$

Desirability measures the effect of an action on user's goals, which is ignored in the other two metrics. It separates common harmless actions (e.g., user opening web browser) from avoidable ones and connects the observations to knowledge of the user's goals. In this study, we use *Desirability* to downgrade the contribution of common actions to criticality. Hence, the negative metric:

$$Desirability(a|\Pi_U) = -\frac{|\text{appearance of } a \text{ in } \Pi_U|}{\sum_{i=1}^{|\Pi_U|} |\pi_i|} \quad (3)$$

Together, these define $V(a)$ for candidate trigger action a for a weighting provided by α:

$$V(a) = \alpha_1 * Certainty(a|\Pi_U) + \alpha_2 * Timeliness(a|\Pi_U) + \alpha_3 * Desirability(a|\Pi_U)$$

Table 1, shows how the critical trigger action is identified using the proposed objective function for the example in Figure 1 given the observation sequence of actions $O = \{a1, a2, a3\}$. In state S_{a3}, assuming equal weighting of metrics, the algorithm identifies f to be the action that maximizes the objective function, and returns it as a possible point of intervention. Table 1 also shows how this decision is affected by the choice of α. If only certainty metric is used ($\alpha = [1, 0, 0]$) action f gets selected as the intervention point. Using only timeliness metric ($\alpha = [0, 1, 0]$) yields three possible intervention points: f, b and c. Finally, using only desirability yields four possible intervention points: h, m, n and q.

3 Critical Trigger Recognition Algorithm

Intervention is different from the typical plan recognition problem because we assume the user pursues goals but wants to also avoid undesirable state. As such, the agent responsible for identifying the critical trigger actions (i.e., intervention point) has no knowledge about the user's end goal and knows only the set of states the user should avoid. Furthermore, identifying the goal or the plan of the user, as in the solution to goal or plan recognition problem, does not guarantee that the undesirable state will not be triggered. Therefore, the objective is to identify intervention points, based on their potential to cause an undesirable state.

Our problem assumes user's actual end goals are unknown and only possesses knowledge about a set of states user wants to avoid. Thus, we view the user's attempt to achieve a goal as a planning process. This entails that plan library based solutions are not feasible. Therefore, our approach borrows ideas from the generative plan recognition approaches. Our algorithm analyzes the Planning Graph and operates incrementally, similar to prior work by Sun et al. (2007) and Hong (2001). Following work by Geib and Goldman (2009), we adopt a model focused on execution: what could be done next. In contrast to the works of Ramirez and Geffner (2009; 2010) that assumed a fully observable state space, our approach takes into consideration the inher-

ent unreliable nature of observations (missing and irrelevant actions) towards identifying critical trigger actions. Further, we assume that the undesirable states (goals) are known and unintentional by the actor. Our solution employs PDDL and the Mosaic planner (Roberts, Howe, and Ray 2014) to sample possible plans from the current state. For each observation, the algorithm outputs whether or not it is a critical trigger action.

Figure 2 shows the seven-step decision cycle of the algorithm. As defined in the problem statement, the process takes as input: a PDDL domain, an initial state, a set of undesirable states and the objective function weights.

Step 1: We assume the full observation trace is known in advance, with actions made available one at a time to identify the intervention point.

Step 2: For each possible undesirable state U, generate PDDL problem instances. In the first cycle, the initial state is from the input; in subsequent cycles, the state is updated in step 7. To extract applicable states for the next cycle, we chose to search forward the planning graph starting from initial state. This is because the unobservable actions require us to update state during search to accommodate those changes that are presupposed by the actions being able to execute.

Step 3: Mosiac returns Π_U for U. Mosaic is built on top of the LAMA 2008 (Richter and Westphal 2010) with some patches applied to its parser from the current Fast Downward repository[1]. It included three extensions to improve the plan set and produce diverse plans faster: the use of a tabu mechanism to guide search away from already known solutions, the use of multiple queues to increase the diversity of solutions explored, and the use of parsimony to bias the search toward goal-oriented solutions. From the family of variants, iterated Tabu A*(ITA) and Multi-queue A* (MQA), we use MQA-ts to compute up to 10 possible plans, based both on the recommendation of the authors and because it showed the most promise in prior work.

Step 4: Extract unique actions in Π_U as candidate trigger actions. Actions include the action name and the instantiated parameter values.

Step 5: Compute C, T and D for each candidate trigger action A by iterating over the plans in Π_U, as defined in section 2.1.

Step 6: Rank candidate actions by V and flag the highest ranked actions are critical triggers corresponding to the current observation. Step 6 also evaluates the accuracy of the identified critical trigger by comparing against the observation. We use a ground-truth plan (UP) that achieves the undesirable state and assume that all actions in UP are critical triggers. If the algorithm returns the current observation as the critical trigger, then the decision is correct if the current observation is an action in UP (i.e., true-positive). Alternatively, if the current observation is not the critical trigger and it was also missing from UP (i.e., observation was an extraneous action and a true-negative), then that decision is also labeled as correct. All other cases are incorrect.

Step 7: The cycle begins over by merging the effects of the observed action as defined in the PDDL domain with the current state. Because we allow for partial observability, the observed action may appear to be inapplicable because actions that satisfied its preconditions were not observed and so captured by the current state. Our approach addresses this inconsistency in state by finding a plan that modifies current state to the state after adding the effects of the observed action. Then, each action in that plan are executed (i.e., adding add effects, removing delete effects) to update the current state.

[1]See `http://www.fast-downward.org/`

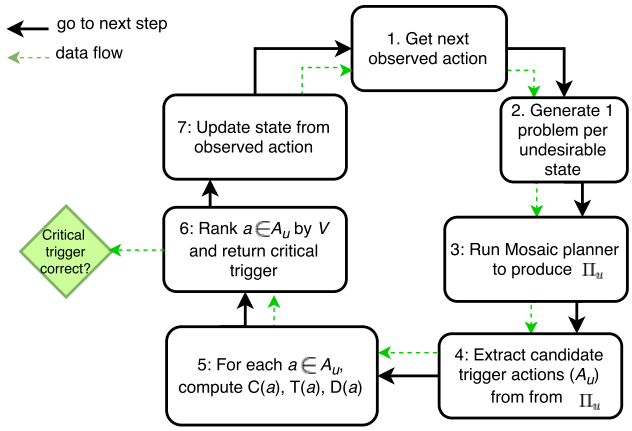


Figure 2: Seven-step process for determining whether an observed action is a critical trigger action.

4 Experimental Setup

We examine which factors impact the performance of detecting critical trigger actions. The *independent variables* of the study are summarized in Table 2, and include the weights for V, the problems, the percentage of extraneous actions, and the percentage of actions "removed" to emulated partial observability. Objective weights vary between a single metric, pairs of metrics and all three metrics with equal weights. We decided on these objective weight settings in order to identify which metrics (or their combinations) are sensitive to partial observability and extraneous actions separately.

We begin with results from a user study on cybersecurity that motivated our work. For traces from the user study, we examine the impact of the objective weights (OW) used for in the objective function. Noisy sensing can lead to extraneous actions or missed observations, which complicates intervention. So we generalize these results to a set of benchmark domains from (Ramırez and Geffner 2009), where greater experimental control allows us to evaluate the impact of extraneous actions (EA) and partial observability (PO). Thus, we expect some degradation in performance as the noise increases and observability decreases; the question is by how much?

The *dependent variables* of the study include accuracy and computational overhead as measured in CPU time. For the cybersecurity domain, accuracy is defined as the percent true-positives only. For the benchmarks, we report accuracy in terms of percent true-positives and true-negatives. Computation time is included because ultimately we envision this

Variable	Settings
Objective weights (OW) (C,T,D)	(1,0,0), (0,1,0), (0,0,1), (0.33,0.33,0.33), (0.5,0.5,0), (0.5,0,0.5), (0,0.5,0.5)
Planning domains (PD)	`PAG`, `blocks-words`, `navigator`, `ipc-grid+`, `logistics`
Extraneous actions (EA)	0%, 50%, 75%, 100%
Partial observability (PO)	25%, 50%, 75%, 100%

Table 2: Independent variables for evaluating impact of algorithm parameters and sensitivity to noisy data

being one component of a user supporting agent, which will require fast response. CPU time includes the time required to process each observation (one cycle of the process depicted in Figure 2) within a trial; this includes both generating the alternate plans and ranking the actions.

We treat the undesirable plan (UP) achieving the undesirable goal state as the 'ground truth' trace. This is a fair assumption because the trace generation algorithm produces traces leading to the undesirable state (with varying levels of noise and observability). We measure accuracy with two metrics: (1) success rate in ignoring extraneous actions (Ignored EA) and (2) success rate in flagging undesirable actions that appear in the ground truth plan (UP) (Flagged UP). Ignored EA for an observation trace is computed as the count of instances where the observation was an extraneous action and it was not flagged as a critical trigger (count EA). Thus, Ignored EA% = (count EA / number of extraneous observations in trace). Flagged UP for an observation trace is computed as the count of instances where the observation was an action from UP and was flagged as a critical trigger (count UP). Thus, Flagged UP% = (count UP / number of actions of UP that are in trace)

5 Cybersecurity Domain (PAG)

We conducted an human subject experiment in a sand-boxed simulation environment to determine how non-expert users behave when presented with questionable computer security situations and to compare their actual observed behavior to their self-reports obtained via pre- and post-hoc surveys. Participants were presented with a desktop which includes common applications such as emailing, Web-browsing, social networking etc. Subjects were provided with written instructions on how to perform normal home computer user activities such as reading/sending email, installing software etc. While the normal computer activities were being performed, different events were simulated, without the subject's knowledge that can trigger threats and vulnerabilities of interest. These events included enticing user to disclose sensitive information (login names, passwords) to phishing sites over email and social media communications, responding to pop-ups asking the user to download a software, and reacting to malicious activities detected in the computer by an anti-virus program. Sixty-three human subjects participated in the study; their actions were collected into observation traces.

We constructed a PDDL domain for the Personalized Attack Graph (`PAG`) to investigate what actions subjects might take amid the aforementioned threat scenarios. We considered four undesirable states for this domain, resulting in four problem definitions. The `PAG` domain has 45 actions, 47 types, 70 constants and 55 predicates.

We expected accuracy to be low because user logs offer unique challenges to the algorithm. Human users do not typically perform tasks in a ordered sequence. For example, some logs indicated that the subject was performing the same task repetitively, users had skipped a task to come back to it after an interval and more interestingly, some users were actively trying to trigger the vulnerabilities. These scenarios make it difficult to model a consistent state in a planning environment and affect candidate trigger actions being selected for the critical trigger ranking algorithm. To assess algorithm performance in a more controlled environment, we turned to benchmarks.

5.1 PAG Results

Since activity logs captured during the experiment could not be properly controlled for extraneous and missing actions, we limit the assessment to accuracy and the effect of objective metric weightings on accuracy and CPU usage. The subjects were only provided with instructions to perform normal computer tasks. No restrictions were imposed on how they should interact with desktop environment (e.g., undo/redo tasks, exploring application features). To this end, subject's goals while performing the tasks were unknown. Logs generated by two subjects were discarded because they did not complete the experiment. Considering 61 traces used in this evaluation, mean trace length was 39 (SD=21.13). The longest trace contained 120 observations, while the shortest was 10.

Mean accuracy was 59.53% (SD=30.79) for all OW, but the weights exerted a significant influence on accuracy (F=40866, $p \ll 0$). The highest accuracy (mean=95.59%,SD=2.13) occurred for two configurations of objective weight assignments: one that equally weighted all three metrics and one that equally weighted certainty and timeliness, ignoring desirability. This suggests, by tagging specific undesirable actions and monitoring progress of plans, onset of critical states can be identified accurately in cybersecurity scenario. Mean CPU time per cycle of the algorithm was 1.7 seconds (SD=0.45), suggesting that this algorithm could easily be integrated into an agent that mitigates cybersecurity issues.

6 Benchmark Domains

To generalize the results, we examine four domains from (Ramırez and Geffner 2009). `Block-Words` constructs one of 20 English words using 8 distinct letters. `Grid-Navigation` forms paths through a connected graph to specific locations (goals). `IPC-grid+` is a grid navigation with keys added at a restricted set of locations. `Logistics` moves packages between locations. To keep the scale similar to the user study, we randomly selected four problems from each domain distribution.

6.1 Trace Generation

To construct problem factors (EA, PO), we consider four problems from each benchmark domain. For each problem, we generate noisy traces by incrementally building the plan starting from the initial state and interleaving it with actions from a set of precomputed extraneous actions when the current state meets the preconditions of the extraneous action. Thus, the trace generation algorithm consists of two stages (1) computing the set of extraneous actions (EA) and (2) interleaving these extraneous actions. Actions from the original undesirable plan are randomly removed from the interleaved trace to account for partial observability (PO).

We use Metric-FF (Hoffmann 2003) planner to generate extraneous actions because the planner's implementation offers configuration options to extract the relaxed planning graph and add/delete effects of actions. The challenge in generating extraneous actions to ensure it (1) does not introduce new facts that interfere with the progression of the 'ground truth' plan or (2) does not delete the progress already made by the 'ground truth' plan. We incrementally build the set of extraneous action starting from selecting actions that can be executed immediately after the goal state has been reached. By adding new actions, object types, and predicates to the domain model, the size of the resulting extraneous action set can be significantly increased to handle longer plans and create lengthy traces. The advantage of iteratively forward-expanding the set of extraneous actions starting at the goal state that at this stage the only condition we need to check is whether or not the chosen action deletes the goal state or not. Additionally, alternative actions available at the current level of the plan graph do not guarantee that they will not violate conditions (1) and (2) later. In certain occasions this could lead to traces where the same action is being done and undone repeatedly without making progress through the plan. We designed trace generation algorithm that can overcome this challenge and builds the ground-truth plan incrementally from the initial state, while interleaving it with extraneous actions at relevant states.

We define percent extraneous actions (EA) in the observation trace (number of extraneous actions in trace / number of actions in plan) as a proxy for signal-to-noise ratio [0%, 50%, 75%, 100%]. Starting from the initial state, actions in undesirable plan are added to the observation trace sequentially, including extraneous actions as appropriate. This produces a final plan. Partial observability (PO) is computed by removing actions from the final plan are randomly selected and removed such that, PO% (number of original actions in trace after removal / number of original actions in trace before removal) is one of [25%, 50%, 75%, 100%].

6.2 Benchmark Results

We review the overall accuracy of our algorithm in benchmark domains by evaluating how well it ignores extraneous actions and flags undesirable actions.

Ignoring Extraneous Actions When encountered with extraneous actions in the trace, the algorithm must be able to avoid flagging it as critical. We define mean Ignored EA% percentage for a domain as: (sum of Ignored EA% per trial / number of trials with EA%>0). Table 3 shows how mean Ignored EA% varies with the noise level of the observation trace and objective weight assignments.

Results show that our algorithm consistently ignores extraneous actions for all benchmark domains. This high accuracy rate can be attributed to our process of selecting critical trigger actions. Since candidate trigger actions are extracted from a set of alternative plans leading to the same undesirable state, the likelihood of true extraneous actions to appear in the set of alternative plans is low. This reduces the likelihood of false-positives in the trace.

The OW combinations significantly influence Ignored EA% ($p < 0.05$ for the `blo`, `ipc` and `log` domains). Post-hoc analysis using TukeyHSD at $\alpha = 0.05$ shows that across domains, OW combinations that do not consider the timeliness metric better perform at ignoring extraneous actions than combinations that include timeliness. This is because timeliness metric is not sensitive to extraneous actions in the trace. As a result, the objective function can not sufficiently differentiate between extraneous actions and actions in ground truth plan, leading to false positives. In contrast, certainty and desirability metrics look for occurrences of an action in undesirable plans. As extraneous actions do not occur in undesirable plans, both metrics are capable of filtering extraneous actions from the candidate action pool by minimizing objective function value and preventing them from being selected as the critical trigger.

Flagging Undesirable Actions Flagged UP% captures how well the algorithm flags observations as critical triggers given that the observation appears in an undesirable plan treated as ground truth. We first look at Flagged UP% in traces with 0% noise and full observability (i.e., best-case scenario) to establish an upper bound to the metric.

Table 6 shows a very large range (Max-Min) for Flagged UP%. This indicates that although the only variable for this sample is OW, Flagged UP% is also sensitive to other external factors that have not been accounted for in our proposed objective function. The challenge lies in identifying these external factors and determining their relationship so that the objective function can be tuned to improve accuracy. This will be the main focus of our future work.

Table 4 shows that Flagged UP% also increases when observability increases in the trace. Factor analysis using one-way ANOVA for OW shows that this positive effect is significant ($p < 0.05, df = 6$) for all four benchmark domains. Interestingly, high Flagged UP% was reported for OW combinations that consider timeliness: specifically for configurations, certainty-timeliness-desirability with equal weighting, certainty-timeliness with equal weighting and desirability-timeliness with equal weighting. Post-hoc analysis using TukeyHSD at $\alpha = 0.05$ shows that this difference is significant. Thus, we conclude that the timeliness metric can improve the true-positive rate, yielding higher precision for the algorithm. The timeliness metric is sensitive to partial observability because it sufficiently captures distance to triggering an undesirable state, which is a consistent indicator of the progress is being made. Even with partial observability, the remaining steps of a plan change in such a way that it can

Domain	Mean Ignored EA%									
	EA% in trace			OW Assignments (C,T,D)						
	50%	75%	100%	(0,0,1)	(0,0.5,0.5)	(0,1,0)	(0.3,0.3,0.3)	(0.5,0,0.5)	(0.5,0.5,0)	(1,0,0)
blo	95.2 (8.0)	97.6 (3.6)	97.3 (3.2)	99.0 (1.6)	94.7 (6.7)	95.2 (6.2)	95.4 (7.0)	98.6 (1.7)	95.4 (7.0)	98.6 (1.7)
ipc	86.4 (14.4)	89.9 (11.6)	89.9 (10.7)	95.6 (3.7)	83.2 (13.0)	81.3 (12.9)	83.9 (15.1)	95.9 (3.4)	85.1 (14.8)	95.9 (3.4)
log	97.9 (4.4)	96.7 (5.7)	96.1 (5.4)	98.4 (2.8)	94.4 (7.2)	96.1 (3.6)	96.2 (6.9)	98.6 (2.6)	96.2 (6.9)	98.5 (2.6)
nav	100 (0)	100 (0)	100 (0)	100 (0)	100 (0)	100 (0)	100 (0)	100 (0)	100 (0)	100 (0)

Table 3: Mean Ignored EA% (and standard deviation) for benchmark domains for levels of EA% in observation trace and objective weight assignment classes. The left three columns combine mean Ignored EA% for OW assignments across all values of α and breaks mean Ignored EA% down by EA% in trace. The right OW columns breaks down mean Ignored EA% by specific α value for all levels of EA% in observation trace.

Domain	Mean Flagged UP%										
	PO% in trace				OW Assignments (C,T,D)						
	25%	50%	75%	100%	(0,0,1)	(0,0.5,0.5)	(0,1,0)	(0.3,0.3,0.3)	(0.5,0,0.5)	(0.5,0.5,0)	(1,0,0)
blo	20.1 (21.2)	23.6 (14.4)	31.8 (19.3)	40.2 (26.1)	9.2 (7.6)	24.8 (20.1)	37.0 (17.5)	46.5 (24.9)	19.0 (7.6)	46.8 (24.9)	19.0 (7.6)
ipc	6.9 (18.8)	25.6 (20.0)	42.3 (32.0)	50.8 (36.2)	9.7 (5.2)	46.5 (34.7)	47.9 (35.0)	47.5 (35.8)	10.5 (4.9)	47.1 (35.9)	10.6 (4.9)
log	18.4 (27.4)	16.4 (20.1)	20.4 (19.5)	31.5 (17.8)	15.2 (10.6)	20.9 (28.2)	24.6 (25.4)	27.2 (28.5)	18.3 (10.1)	27.2 (28.5)	18.3 (10.9)
nav	14.8 (19.1)	32.4 (22.6)	45.2 (27.4)	61.7 (39.8)	14.1 (3.4)	56.8 (33.7)	56.8 (33.7)	56.8 (33.7)	14.1 (3.43)	56.8 (33.7)	14.1 (3.4)

Table 4: Mean Flagged UP% (and standard deviation) for benchmark domains for levels of PO% in observation trace and objective weight assignment classes. The left four columns combine mean Flagged UP% for OW assignments across all values of α and breaks mean Flagged UP% down by PO% in trace. The right OW columns breaks down mean Flagged UP% by specific α value for all levels of PO% in observation trace.

Domain	EA% in trace				PO% in trace				Δ
	0%	50%	75%	100%	25%	50%	75%	100%	
blo	2.3	2.2	2.2	2.2	2.2	2.2	2.3	2.2	1.4
ipc	4.9	4.8	4.9	4.9	4.9	4.9	4.9	4.8	0.9
log	1.7	1.7	1.6	1.7	1.7	1.7	1.7	1.7	0.5
nav	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	0.5

Table 5: Mean CPU time in seconds for problem factors for each domain. Δ is the difference between the min and max times for each domain.

Domain	Flagged UP%			
	Mean	SD	Min	Max
blo	37.12	24.83	6.12	77.78
ipc	43.27	30.55	0.00	90.00
log	31.77	17.27	12.82	66.67
nav	61.68	40.31	14.10	100.00

Table 6: Flagged UP% for traces with 0% EA and 100% PO for Block-words (bw), IPC-grid+ (ipc), Logistics (log), and Grid-Navigation (nav) domains

be tracked and correctly reflected in the objective function. Accuracy of the certainty and desirability metrics lowers as partial observability increases because they consider occurrences of an action.

Computational Overhead For each benchmark domain, we calculated the average CPU time per cycle for problem factors (EA and PO). As shown in Table 5, we found differences due to observation trace factors; two-way ANOVA on EA and PO showed significant main effects and interaction effects (p <0.05) for ipc, log and nav domains.

7 Related Work

The two most closely related areas of literature are plan recognition and automated planning for cybersecurity. Plan recognition is the problem of inferring the course of action (i.e., plan) an actor may take towards achieving a goal from a sequence of observations (Schmidt, Sridharan, and Goodson 1978; Kautz and Allen 1986). The constructed plan, if followed to completion, is expected to result in states that correspond to goals of the actor, which in turn presupposes that the actor intends to achieve those goals. Many approaches to plan recognition use a plan library to infer a relationship between the observed actions and hypothetical plans/goals (Carberry 2001). For example, PHATT enumerated all plans, which included the observations, to compute the proportion consistent with each goal (Geib and Goldman 2009). Plan libraries use pre-compiled plans (with some approaches allowing for learning new plans (Lesh and Etzioni 1996; Bauer 1998)), which can limit the set of recognizable sequences.

Hong (2001) constructed a "Goal Graph" of observed actions, world state nodes and goal nodes; this graph was analyzed to extract the goals that explained the observed actions. Sun and Yin (2007) constructed valid plans given a partial set of observations extending the Planning Graph (Blum and Furst 1997) to allow for states to be unknown and embedded the analysis in a system that incrementally updated the graph while action transpired. Ramirez and Geffner (2009; 2010) used an existing planner to generate hypotheses from observations. Their approaches offer advantages of being more adaptive to input as well as exploiting existing planning systems and plan representations. Their first approach computed the set of goals that can be achieved by optimal plans that match the observations. Their second approach removed the optimality constraint and computed a probability distribution across possible plans that could be generated from existing planners. Vattam et al. (2015) allowed for missing or misclassified actions in the traces. Their SET-PR represented library plans as action sequences and computed a similarity metric between observations and known plans.

AI Planning has previously been used to reason about computer security. Boddy et al. (2005) built Behavioral Adversary Modeling System, which could identify potential vulnerabilities and countermeasures with a focus on insider subversion. Sohrabi et al. (2013) generated hypotheses of which nodes in a network might be infected with malware. The Core Security model captures the actions of attackers at the level of known network vulnerabilities(Obes, Sarraute, and Richarte 2010). Hoffmann has extended the Core Security model as a POMDP (Hoffmann 2015) for penetration testing: simulating what an attacker might reasonably try over time. Geib and Goldman (2009) defined a domain in which attackers try to achieve one of three goals: *Bragging*, which is the attacker boasting of success, *Theft*, which is the theft of information, or causing a *Denial of Service* to legitimate users by exploiting security vulnerabilities on servers.

The CIRCA architecture for adaptive real-time control systems (Goldman et al. 1997) presents a solution to the problem of avoiding undesirable states. They proposed the concept "temporal preemption", which builds event controllers (plans) that make potential failure states unreachable, and thereby keeping the system safe. The difference between their domain and ours is that in their domain, the actions to which the plans are generated have timing constraints/durations and ours do not. Further, our focus is on identifying the intervention point as early as *helpful* when a user may be triggering an undesirable state.

8 Summary and Future Work

We have described a variant of plan recognition that can help identify intervention points to help a user avoid undesirable states while interacting with a computer. Our approach views the decision of when to intervene as a multi-objective optimization problem that optimizes three domain-independent objective metrics: certainty, timeliness and desirability. We tested our algorithm on both benchmark domains and human subject data from a cybersecurity experiment. Results show that, across all benchmark domains, certainty and desirability metrics perform well in ignoring extraneous actions, while the timeliness metric and it's combinations with certainty and desirability perform well in identifying true positives. We identified two metrics that are sensitive to noise in action based observation traces and a metric that is sensitive to partial observability of actions.

However, the low percentage of true-positives in the best case scenario indicate that the metrics are not good enough in their raw form in dealing with partial observability. Therefore, metrics must be developed to evaluate the contribution of less frequent actions appearing in alternative plans. Evaluation of the effects of objective weight metrics, shows that desirability metric does not adequately downgrade the effect of certainty and timeliness. This indicates that objective weight assignments require an in depth look into multi-objective optimization techniques to find the optimal combination of weights over metrics. Furthermore, concepts behind certainty metric touches upon landmarks in planning. We will further explore these aspects in the future.

Intervention should be embedded in a system that acts like a smart executive assistant. As suggested in (Papadimitriou et al. 2015), adding reasoning about the goals of the actors can add personalization (e.g., tune when actions are flagged) and improve the effectiveness of the computer/user interaction.

Acknowledgments

We thank the anonymous reviewers for comments that helped improve the paper. Mark Roberts thanks NRL for funding this research.

References

Banerjee, B.; Kraemer, L.; and Lyle, J. 2010. Multi-agent plan recognition: Formalization and algorithms. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, AAAI'10, 1059–1064. Atlanta, Georgia: AAAI Press.

Bauer, M. 1998. Acquisition of abstract plan descriptions for plan recognition. In *Proceedings of the 15th Nat. Conf. on Artificial Intelligence (AAAI)*, 936–941.

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1):281–300.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, 12–21.

Carberry, S. 2001. Techniques for plan recognition. *User Modeling and User-Adapted Interaction* 11(1):31–48.

Gal, Y.; Reddy, S.; Shieber, S. M.; Rubin, A.; and Grosz, B. J. 2012. Plan recognition in exploratory domains. *Artificial Intelligence* 176(1):2270–2290.

Geib, C. W., and Goldman, R. P. 2005. Partial observability and probabilistic plan/goal recognition. In *Proceedings of the International workshop on modeling other agents from observations (MOO-05)*, volume 8, 1–6.

Geib, C., and Goldman, R. 2009. A probabilistic plan recognitino algorithm based on plan tree grammars. *Artificial Intelligence* 173:1101–1132.

Goldman, R. P.; Musliner, D. J.; Boddy, M. S.; and Krebsbach, K. D. 1997. The CIRCA model of planning and execution. In *Working Notes of the AAAI Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control*, volume 970. AAAI.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20(1):291–341.

Hoffmann, J. 2015. Simulated penetration testing: From "Dijkstra" to "Turing Test++". In *Proceedings of International Conference. on Automated Planning and Scheduling (ICAPS)*, 364–372. AAAI Press.

Hong, J. 2001. Goal recognition through goal graph analysis. *Journal of Artificial Intelligence Research* 15:1–30.

Howe, A. E.; Ray, I.; Roberts, M.; Urbanska, M.; and Byrne, Z. 2012. The psychology of security for the home computer user. In *Proceedings of the IEEE Symposium on Security and Privacy*, 209–223.

Kaminka, G. A.; Pynadath, D. V.; and Tambe, M. 2002. Monitoring teams by overhearing: A multi-agent plan-recognition approach. *Journal of Artificial Intelligence Research* 17(1):83–135.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of 5th National Conference on Artificial Intelligence (AAAI)*, 32–37.

Lesh, N., and Etzioni, O. 1996. Scaling up goal recognition. In *Proceedings of the 5th International Conference on Knowledge Representation and Reasoning (KR)*, 178–189.

Obes, J. L.; Sarraute, C.; and Richarte, G. 2010. Attack planning in the real world. In *Working Notes for the AAAI Workshop on Intelligent Security (SecArt)*, 10–17.

Papadimitriou, D.; Velegrakis, Y.; Koutrika, G.; and Mylopoulos, J. 2015. Goals in social media, information retrieval and intelligent agents. In *IEEE 31st International Conference on Data Engineering (ICDE),*, 1538–1540.

Ramırez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI)*, 1778–1783.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, 1121–1126.

Ramírez, M., and Geffner, H. 2011. Goal recognition over POMDPs: Inferring the intention of a POMDP agent. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 2009–2014. AAAI Press.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Roberts, M.; Howe, A.; and Ray, I. 2014. Evaluating diversity for classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 253–261. NH, USA: AAAI Press.

Sasse, M. A.; Brostoff, S.; and Weirich, D. 2001. Transforming the 'weakest link' – a human/computer interaction approach to usable and effective security. *BT Technology Journal* 19(3):122–131.

Schmidt, C. F.; Sridharan, N.; and Goodson, J. L. 1978. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence* 11(1-2):45–83.

Sheyner, O.; Haines, J.; Jha, S.; Lippmann, R.; and Wing, J. 2002. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, 273 – 284. Washington, DC, USA: IEEE Computer Society.

Sohrabi, S.; Riabov, A.; and Udrea, O. 2016. Plan recognition as planning revisited. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 3258–3264. New York, NY, USA: AAAI Press.

Sohrabi, S.; Udrea, O.; and Riabov, A. 2013. Hypothesis exploration for malware detection using planning. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, 883–889. Bellevue, Washington: AAAI Press.

Sun, J., and Yin, M. 2007. Recognizing the agents goals incrementally: planning graph as a basis. *Frontiers of Computer Science in China* 1(1):26–36.

Urbanska, M.; Roberts, M.; Ray, I.; Howe, A.; and Byrne, Z. 2013. Accepting the inevitable: Factoring the user into home computer security. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*.

Vattam, S.; Aha, D.; and Floyd, M. W. 2015. Error tolerant plan recognition: An empirical investigation. In *Proceedings of the 28th Florida Artificial Intelligence Research Society Conference FLAIRS*, 397–403. Hollywood, FL, USA: AAAI Press.

Vattam, S. S., and Aha, D. W. 2015. Case-based plan recognition under imperfect observability. In *Proceedings of the 23rd International Conference on Case Based Reasoning (ICCBR)*, 381–395. Cham, Germany: Springer.

Yin, J.; Chai, X.; and Yang, Q. 2004. High-level goal recognition in a wireless LAN. In *Proceedings of the National Conference on Artificial Intelligence*, 578–584. AAAI Press.

Temporal Planning for Compilation of Quantum Approximate Optimization Circuits

Davide Venturelli[1,3], **Minh Do**[2,4], **Eleanor Rieffel**[1], **and Jeremy Frank**[2]
[1] Quantum Artificial Intelligence Laboratory, NASA Ames Research Center
[2] Planning and Scheduling Group, NASA Ames Research Center
[3] USRA Research Institute for Advanced Computer Science (RIACS)
[4] Stinger Ghaffarian Technologies (SGT Inc.)

Abstract

We investigate the application of temporal planners to the problem of compiling quantum circuits to newly emerging quantum hardware. While our approach is general, we focus our initial experiments on Quantum Approximate Optimization Algorithm (QAOA) circuits that have few ordering constraints and thus allow highly parallel plans. We report on experiments using several temporal planners to compile circuits of various sizes to a realistic hardware architecture. This early empirical evaluation suggests that temporal planning is a viable approach to quantum circuit compilation.

1 Introduction

We explore the use of temporal planners to optimize compilation of quantum circuits to newly emerging quantum hardware. Previously, only special purpose quantum hardware was available, namely, quantum annealers that could run one type of quantum optimization algorithm. The emerging gate-model processors are universal in that once scaled up, they can run any quantum algorithm and thus expanding the empirical exploration of quantum algorithms beyond optimization, as well as enabling the exploration of a broader array of quantum approaches to optimization. IBM recently provided public access to a 5-qubit gate-model processor through the cloud (IBM 2017), recently updated to 17 qubits, and scalable gate-model quantum computing architectures are being manufactured by other groups, such as TU Delft (Versluis et al. 2016), UC Berkeley (Ramasesh et al. 2017), Rigetti Computing (Sete, Zeng, and Rigetti 2016), and Google (Boxio 2016). All cited groups have announced plans to build gate-model quantum processors with 40 or more qubits in the near term.

Quantum algorithms process information stored in qubits, the basic memory unit of quantum processors, and quantum operations (called *gates*) are the building blocks of quantum algorithms, just as instructions on registers are the building blocks of classical algorithms. Specifically, quantum algorithms must be compiled into a set of elementary machine instructions (the gates), which are applied at specific times in order to run them on quantum computing hardware. For a review of quantum computing, see (Rieffel and Polak 2011).

Quantum algorithms are often specified as quantum circuits on idealized hardware since physical hardware has constraints varying from architecture to architecture. For example, the emerging gate-model quantum hardware mentioned above all use superconducting qubits in a planar architectures with nearest-neighbor restrictions on the locations (qubits) to which the gates can be applied. Idealized circuits generally do not consider those nearest neighbor constraints. For this reason, compiling idealized quantum circuits to specific hardware requires adding supplementary gates that move qubit states to locations where the desired gate can act on them.

Quantum computational hardware suffers from *decoherence*, which degrades the performance of quantum algorithms over time. Especially for near-term hardware, which will not be able to mitigate decoherence, it is important to *minimize* the *duration of the circuit* that carries out the quantum computation, so as to minimize the decoherence experienced by the computation. Optimizing the duration of compiled circuits is a challenging problem due to the parallel execution of gates with different durations. Further, for quantum circuits with flexibility in when the gates can be applied, or when some gates can be applied in a different order while still achieving the same computation, the search space for feasible compilations is often very large. That freedom makes it more challenging to find optimal compilations, but also means there is a greater potential win from improved compilation optimization than for less flexible circuits.

While there has been active development of software libraries to synthesize and compile quantum circuits from algorithm specifications (Wecker and Svore 2014; Smith, Curtis, and Zeng 2016a; Steiger, Häner, and Troyer 2016a; Devitt 2016; Barends et al. 2016), few approaches have been explored for compiling idealized quantum circuits to realistic quantum hardware (Beals et al. 2013; Brierly 2015; Bremner, Montanaro, and Shepherd. 2016), leaving the problem open for innovation. An analogous issue arising when compiling classical programs is the *register allocation* problem, in which program variables are assigned to machine registers to improve execution time; this problem reduces to graph coloring (Fu, Wilken, and Goodwin 2005). Very recently, computer scientists have started looking at heuristic approaches with off-the-shelf mixed-integer-linear-programming MILP solvers such as Gurobi to solve similar general benchmarking problems (Bhattacharjee and Chattopadhyay 2017).

In this paper, we apply temporal planning techniques to

the problem of compiling quantum circuits to realistic gate-model quantum hardware. Specifically, we model machine instructions as PDDL2.1 durative actions, enabling domain-independent temporal planners to find a parallel sequence of conflict-free instructions that when executed can achieve what the high-level quantum algorithm intends to achieve. While our approach is general, we focus our initial experiments on circuits that have few ordering constraints and thus allow highly parallel plans. We report on experiments using several temporal planners to compile circuits of various sizes to an architecture inspired by those currently being built. This early empirical evaluation suggests that temporal planning is a viable approach to quantum circuit compilation. A more elaborate discussions of the techniques and results obtained can be found in the extended version of this paper at (Venturelli et al. 2017).

2 Architecture-specific compilation problem

Quantum circuits for general quantum algorithms are often described in an idealized architecture in which any 2-qubit gate can act on any pair of qubits. In an actual architecture, physical constraints impose restrictions on which pairs of qubits support gate interactions. For superconducting qubit architectures, qubits in a quantum processor can be thought of as nodes in a planar graph, and 2-qubit quantum gates are associated to edges. Gates that operate on distinct sets of qubits may be able to operate concurrently though there may be additional restrictions, such as requiring the sets to be non-adjacent, as in Google's proposed architecture (Boxio 2016)). Furthermore, there are different types of quantum gates, each taking different durations, with the duration depending on the specific physical implementation.

In order for the computation specified by the idealized circuit to be carried out, we require a particular type of 2-qubit gate, the *swap* gate, which exchanges the state of two qubits. A sequence of swap gates moves the contents of two distant qubits to a location where a desired gate can be applied. Swap gates may be available only on a subset of edges in the hardware graph, and swap duration may depend on where they are located. For the purposes of this study, we will consider the case in which swap gates are available between any two adjacent qubits on the chip and all swap gates have the same duration, but our approach can handle the more general cases.

Problem definition: Given an idealized circuit consisting only of the non-swap gates, used to define a general quantum algorithm, the circuit compilation problem is to find a new architecture-specific circuit that implements the idealized quantum circuit by adding swap gates when required. The main objective is to minimize the overall duration to execute all gates in a new circuit.

Compilation example: Figure 1 shows a hypothetical chip design that we will use for our experiments on circuit compilation. It is inspired by the architecture of the machine envisioned by Rigetti Computing Inc. (Sete, Zeng, and Rigetti 2016). Qubits are labeled with n_i and the colored edges indicate the types of 2-qubit gates available, in this case swap

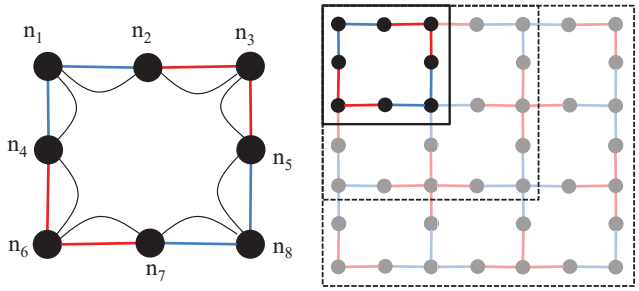


Figure 1: *Left*: A schematic for the hypothetical chip design used in our numerical experiments, with available 2-qubit gates represented by colored arcs in a weighted multigraph. Each color is associated to a specified, distinct gate-type and duration: SWAP gates (black) and two other types of 2-qubits gates (red and blue). The 1-qubit gates are present at each qubit (black dot). *Right*: Dashed boxes indicate the 3 different chip sizes used in our empirical evaluation (see Sec. 5). For visual clarity, only the label locations and the SWAP-gates for the smaller chip size, corresponding to the top-left sector of the largest chip, are shown.

gates and two other types of 2-qubit gate (further described in Section 4).

To illustrate the challenges of finding effective compilation, we present some concrete examples, with reference to the 8-qubit section in the top left of Fig. 1. Suppose that at the beginning of the compilation, each qubit location n_i is associated to the qubit state q_i. Let us also assume that the idealized circuit requires the application of a red gate to the states q_2 and q_4, initially located on qubits n_2 and n_4. One way to achieve this task would be to swap the state in n_4 with n_1, while at the same time swapping n_2 with n_3. Another swap, between n_1 and n_2, positions q_4 in n_2 where a red-gate connects it to q_2 (which is now in n_3).

The sequence of gates to achieve the stated goal are:

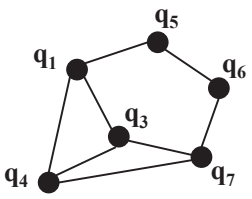
$$\{\text{SWAP}_{n_4,n_1}, \text{SWAP}_{n_2,n_3}\} \rightarrow \text{SWAP}_{n_1,n_2} \rightarrow \text{RED}_{n_2,n_3} \equiv \text{RED}(q_2, q_4) \quad (1)$$

The first line refers to the sequence of gate applications, while the second corresponds to the algorithm objective specification (a task defined over the qubit states). The sequence in Eq. (1) takes $2\tau_{swap} + \tau_{red}$ clock cycles where $\tau_\star$ represents the duration of the $\star$-gate.

As the second example, the idealized circuit requires $\text{BLUE}(q_1, q_2) \wedge \text{RED}(q_4, q_2)$, in no particular order. If $\tau_{blue} > 3 \times \tau_{swap}$, the compiler might want to execute BLUE_{n_1,n_2} while the qubit state q_4 is swapped all the way clockwise in five SWAPs from n_4 to n_3 where RED_{n_2,n_3} can be executed. However, if $\tau_{swap} < 3 \times \tau_{blue}$, it is preferable to wait until the end of BLUE_{n_1,n_2} and then start to execute the instruction sequence in Eq. (1).

3 Compiling QAOA for the MaxCut problem

While our approach can be used to compile arbitrary quantum circuits to a wide range of architectures, in this paper we concentrate on one particular case: compiling



PS1	*MX*	*PS2*
P-S(q_1, q_4)	MIX(q_1)	P-S(q_1, q_4)
P-S(q_1, q_3)	MIX(q_3)	P-S(q_1, q_3)
P-S(q_3, q_4)	MIX(q_4)	P-S(q_3, q_4)
P-S(q_3, q_7)	MIX(q_5)	P-S(q_3, q_7)
P-S(q_4, q_7)	MIX(q_6)	P-S(q_4, q_7)
P-S(q_6, q_7)	MIX(q_7)	P-S(q_6, q_7)
P-S(q_5, q_6)		P-S(q_5, q_6)
P-S(q_1, q_5)		P-S(q_1, q_5)

Figure 2: An example 6-vertex MaxCut problem on a randomly generated graph (qstates q_2 and q_8 are not appearing in this instance) and the p-s and mix gates for $p = 2$.

QAOA circuits for MaxCut to an architecture shown in Figure 1. We choose to work with QAOA circuits because they have many gates that "commute" with each other (i.e., no ordering enforced). Such flexibility means that the compilation search space is larger than for other less flexible circuits. Thus, compared to other classes of circuits, finding the optimal compilation is more challenging, but there is greater potential from improved compilation optimization. MaxCut was selected as the target problem since it has been becoming one of the de-facto benchmark standards for quantum optimization of all types and it is considered a primary target for experimentations in the architecture of (Sete, Zeng, and Rigetti 2016)[1].

MaxCut Problem: Given a graph $G(V, E)$ with $n = |V|$ vertices and $m = |E|$ edges. The objective is to partition the graph vertices into two sets such that the number of edges connecting vertices in different sets is maximized.

From an optimization standpoint, the quadratic boolean objective function whose maximization solves MaxCut is:

$$U_{MaxCut} = \frac{1}{2} \sum_{(i,j)\in E} (1 - s_i s_j) \qquad (2)$$

where s_i and s_j are binary variables associated to the vertices in V which assume value +1 or -1 depending on which of the two partitions defined by the cut are assigned. From this formulation, an idealized QAOA circuit requires a 2-qubit gate for each quadratic term in Eq. (2), as well as an 1-qubit gate for each vertice (Farhi, Goldstone, and Gutmann. 2014a).

Idealized QAOA circuits for MaxCut alternate between a *phase separation* step (PS) and a *mixing* step. The phase-separation step for QAOA for MaxCut is simpler than for other optimization problems and consists of a set of identical 2-qubit gates that must be applied between certain pairs of qubits, depending on the graph of the MaxCut instance under consideration. We will refer to these as *p-s* gates, and the main goal of the compilation is to plan out those gates. All p-s gates can be carried out in any order (subject to constraints on the chip). In the mixing phase, a set of 1-qubit operations are applied, one to each qubit. All p-s gates that involve a specific qubit q must be carried out before the mixing operator on q can be applied. These two steps are repeated p times. We consider p = 1 and p = 2 in our experiments (detailed in Section 5). Fig. 2 shows a concrete 6-vertex MaxCut example with the set of available p-s and mix gates for $p = 2$.

With reference to Fig. 1, the constraints on the compilation problem are:

- SWAP gates are located at every edge with τ_{swap} = 2.
- there are two kind of non-swap gates: P-S gates are 2-qubit gates and MIX gates are 1-qubit gates.
- P-S gates are located at every edge of the grid, but their duration τ_{p-s} can be 3 or 4 depending on their location (respectively blue or red edges in Fig.1).
- MIX gates are located at every vertex with τ_{mix}=1.
- In the initialization stage, which is not considered as part of the compilation problem, a quantum state is assigned to each qubit.

4 Compilation of a Quantum Circuit as Temporal Planning Problem

Planning is the problem of finding a conflict-free set of actions and their respective execution times that connects the *initial-state* I and the desired *goal state* G. We now introduce some key background concepts for the problem of compiling quantum circuits as a temporal planning problem.

Planners: a planner is a software that takes as input a specification of domain and problem descriptions and returns a valid plan if one exists. Many different approaches have been implemented to find a viable plan, among them: (i) heuristically search over the possible valid plan trajectories or over the library of partial plans or (ii) compile the planning problem into another combinatorial substrate (e.g., SAT, MILP, CSP) and feed the problem to off-the-shelf solvers. At the abstract level, the planner needs to solve the QAOA compilation problem exemplified in Figure 2: it identifies the required P-S or MIX gates and builds a conflict-free schedule for all those gates.

Planning Domain Description Language (PDDL): PDDL is a modeling language that was originally created to standardize the input for planners competing in the International Planning Competition (IPC). Over time, it has become the de-factor standard modeling languages used by many domain-independent planners. We use PDDL 2.1, which allows the modeling of temporal planning formulation in which every action a has duration d_a, starting time s_a, and end time $e_a = s_a + d_a$. Action conditions *cond*(a) are required to be satisfied either (i) instantaneously at s_a or e_a or (ii) required to be true starting at s_a and remain true until e_a. Action effects *eff*(a) may instantaneously occur at either s_a or e_a. Actions can execute when their

[1]Recently Google Inc. has published a modified QAOA procedure for MaxCutin fixed nearest-neighbor architectures (Farhi et al. 2017) that explicitly tries to avoid the compilation problem that is the main subject of our work. However, their approach does not work for all classes of MaxCut instances, and the introduced modifications also led to overall performance reduction.

temporally-constrained conditions are satisfied; and when executed will cause state-change effects. The most common objective function in temporal planning is to minimize the plan *makespan*, i.e. the shortest total plan execution time. This objective matches well with the objective of our targeted quantum circuit compilation problem. To enable reuse of key problem features present in an ensemble of similar instances, the PDDL model of a planning problem is separated into two major parts: (i) the *domain* description that captures the common objects and behaviors shared by all problem instances of this planning domain and (ii) the *problem instance* description that captures the problem-specific objects, initial state, and goal setting for each particular problem.

Modeling Quantum Gate Compilation in PDDL 2.1: PDDL is a flexible language that offers multiple alternative ways to model a planning problems. These modeling choices can greatly affect the performance of existing PDDL planners. For instance, many planners pre-process the original domain description before building plans; this is time-consuming, and may produce large 'ground' models depending on how action templates were written. Also, not all planners can handle all PDDL language features effectively (or even at all). We have iterated through different modeling choices with the objective of constructing a PDDL model that: (i) contains a small number of objects and predicates for compact model size; (ii) uses action templates with few parameters to reduce preprocessing effort; while (iii) ensuring that the model can be handled by a wide range of existing PDDL temporal planners.

At the high-level, we need to model: (i) conceptually how actions representing P-S, SWAP, and MIX gates affect qubits and qubit states (qstate); (ii) the actual qubits and qstates involved with a particular compilation problem, their initial locations and final goal requirements, (iii) the underlying qubit-connecting graph structure. We follow the conventional practice of modeling (i) in the *domain description* while (ii) is captured in the *problem description*. One common practice is to model (iii) within the problem file. However, given that we target a rather sparse underlying qubit-connecting graph structure (see Figure 1), we decide to capture it within the domain file to ease the burden of the potentially time-consuming step of "grounding" and pre-processing step for existing planners. Specifically:

Objects: We need to model three types of object: qubits, qstates, and the location of the P-S and SWAP gates (i.e., edges connecting different qubits). Since qstates are associated to specific qubits (by means of the predicate $located\_at$, see Figure 3 for concrete example), they have been modeled explicitly as planning objects, while the qubits and the gate locations (i.e., edges) are modeled implicitly. It is clear from the action definitions in Figure 3 that qubit locations are embedded explicitly within the action declaration. This approach avoids declaring qubits as part of the action parameters, significantly reducing the number of ground actions to be generated. For 2-qubit actions, the potential number of ground actions reduce from

```
(:constants q1 q2 q3 q4 q5 q6 q7 q8 - qstate)

(:durative-action mix_q1_at_1
    :parameters ( )
    :duration (= ?duration 1)
    :condition (and (at start (located_at_1 q1))
                    (at start (GOAL_PS1 q1 q5))
                    (at start (GOAL_PS1 q1 q7))
                    (over all (not (mixed q1))))
    :effect (and (at start (not (located_at_1 q1)))
                 (at end (located_at_1 q1))
                 (at end (mixed q1))))
```

Figure 3: PDDL model of an example MIX gate.

N^4 to $N^2 \times |E|$, with N the number of qubits in the chip (up to 40) and E the set of connections between qubits. While it's true that many modern planners will be able to filter out invalid ground actions during the grounding/preprocessing step, our empirical evaluation shows that capturing the graph structure explicitly in the domain file speeds up the preprocessing time of all tested planners, sometime as significantly as 40x.

Actions: temporal planning actions are created to model: (i) 2-qubit SWAP gates, (ii) 2-qubit P-S gates, and (iii) 1-qubit MIX gates. The most complex constraint to model is the conditions to mix a qstate q given the requirement that *all* P-S gates involving q in the previous phase separation step have been executed. We explored several other choices to model this requirement such as: (i) use a metric variable $PScount(q)$ to model how many P-S gates involving q have been achieved at a given moment; or (ii) use ADL quantification and conditional effect constructs supported in PDDL. Ultimately, we decided to explicitly model all P-S gates that need to be achieved as conditions of the MIX(q) action. This is due to the fact that alternative options require using more expressive features of PDDL2.1 which are not supported by many effective temporal planners[2]. Figure 3 shows an example of the mix gate modeled in PDDL[3].

Alternative model: given that non-temporal planners can perform much better than temporal planners on problems of the same size, we have also created the non-temporal version of the domain by discretizing action durations into consecutive "time-steps" t_i, introducing additional predicates $next(t_i, t_{i+1})$ enforcing a link between consecutive time-steps. However, initial evaluation of this approach with the

[2]Only one of six planners in the Temporal track of the latest IPC (2014) supports numeric variables and also only one of six supports quantified conditions. Preliminary tests with our PDDL model using metric variables to track satisfied goals involving qstate q using several planners shows that they perform much worse than on non-metric version. This is to be expected as currently, state-of-the-art PDDL planners still do not handle metric quantities as well as logical variables.

[3]The full set of PDDL model for all our tested problems is available at: https://ti.arc.nasa.gov/m/groups/asr/planning-and-scheduling/VentCirComp17_data.zip.

	P1						P2			
	N8		N21		N40		N8		N21	
Util	0.9	1.0	0.9	1.0	0.9	1.0	0.9	1.0	0.9	1.0
SGPlan	50	50	50	50	50	50	50	50	-	-
TFD	50	50	50	50	-	-	50	50	50	50
LPG	50	50	50	50	10	14	50	50	-	6

Table 1: Summary of the solving capability of selected planners. Numbers indicate how many random problems out of 50 have been solved.

	p=1, N8		p=1, N21		p=2, N8	
Utilization	0.9	1.0	0.9	1.0	0.9	1.0
SGPlan	0.74	0.76	0.68	0.68	0.76	0.80
TFD	**0.96**	**0.98**	**0.96**	**0.95**	**1.0**	**0.99**
LPG	0.82	0.83	0.83	0.81	0.53	0.51

Table 2: Plan quality comparison between different planners using IPC formula (higher value indicates better plan quality).

M/Mp SAT-based planner (Rintanen 2012) (which optimize parallel planning steps) indicated that the performance of non-temporal planners on this discretized (larger) model is much worse than the performance of existing temporal planners on the original model. Another option is to totally ignore the temporal aspect and encode it as a "classical" planning problem where actions are instantaneous. A post-processing step is then introduced to inject back the temporal constraints and schedule actions in the found classical plans. While we do not believe this approach would produce good quality plans, it's another promising option to scale up to larger problems in this domain.

5 Empirical Evaluation

We have modeled the QAOA circuit compilation problem as described in the previous sections and tested them using various off-the-shelf PDDL 2.1 Level 4 temporal planners. The results were collected on a RedHat Linux 2.4Ghz machine with 8GB RAM.

Problem generation: three grid sizes based with N = 8, 21 and 40 qubits (dashed boxes in Figure 1) were used. The design in Figure 1 is representative of devices to come in the next 2 years; a gate-model 8-qubit chip with the grid we used will be available shortly from Rigetti.

For each grid size, we generated two problem classes: (i) p = 1 (only one PS-mixing step) and (ii) p = 2 (two PS-mixing steps). To generate the graphs G for which a MaxCut needs to be found, for each grid size, we randomly generate 100 Erdös-Rényi graphs G (Erdös and Rényi 1960). Half (50 problems) are generated by choosing N of $N(N-1)/2$ edges over respectively 7, 18, 36 qstates randomly located on the circuit of size 8, 21, and 40 qubits (referred to herafter as 'Utilization' u=90%). The other half are generated by choosing N edges over 8, 21, and 40 qstates, respectively (referred to herafter as 'Utilization' u=100%). In total, we report tests on 600 random planning problems with size ranging from 1024 - 232,000 ground actions and 192 - 8,080 predicates.

Planner setup: Since larger N and/or p lead to more complex settings with more predicates, ground actions, and thus require planners to find longer plans, the allocated cutoff time for different setting are as follow: (i) 10 minutes for $N = 8$, (ii) 30 minutes for $P = 1, N = 21$; (iii) 60 minutes for other cases. We select planners that performed well in the temporal planning track of previous IPCs, while at the same time representing a diverse set of planning technologies: (i) *LPG*: which is based on local search with restarts over action graphs (Gerevini, Saetti, and Serina 2003); (ii) *Temporal FastDownward (TFD):* a heuristic forward state-space (FSS) search planner with post-processing to reduce makespan (Eyerich, Mattmüller, and Röger 2009); and (iii) *SPGlan:* partition the planning problem into subproblems that can be solved separately, while resolving the inconsistencies between partial plans using extended saddle-point condition (Wah and Chen 2004; Chen and Wah 2006).

We ran SGPlan (Ver 5.22) and TFD (Ver IPC2014) with their default parameters while for LPG (Ver TD 1.0) we ran all three available options: (i) *-speed* that uses heuristic geared toward finding a valid plan quickly, (ii) *-quality* that uses heuristic balancing plan quality and search steps, and (iii) *-n 10* that will try to find within the time limit up to *10* plans of gradually better quality by using the makespan of previously found plan as upper-bound when searching for a new plan. Since LPG ($n = 10$) option always dominates both LPG-quality and LPG-speed by solving more problems with better overall quality for all setting, we will exclude results for LPG-quality and LPG-speed from our evaluation discussion. For the rest of this section, LPG result is represented by LPG ($n = 10$).

Evaluation Result Summary: Table 1 shows the overall performance on the ability to find a plan of different planners. SGPlan stops after finding one valid plan while TFD and LPG exhaust the allocated time limit and try to find gradually improving quality plans. Since no planner was able to find a single solution for N = 40 and p = 2, we omit the result for this case from Table 1. Overall, SGPlan and TFD were able to solve the highest number of problems, followed by LPG. SGPlan can find a solution very quickly, compared to the time it takes other two planners to find the first solution. It is the only planner that can scale up to $N = 40$ for p = 1 (finding plans with 150-220 actions). Unfortunately, SGPlan stopped with an internal error for N = 21 and p = 2. TFD generally spent a lot of time on preprocessing for $p = 1, N = 21$ (around 15 minutes) and $p = 2, N = 21$ (around 30 minutes) but when it's done with the pre-processing phase [4] it can find a solution very quickly and also can improve the solution quality very quickly. TFD

[4]The two most time-consuming parts in TFD's pre-processing routine are "processing axioms" and "invariant analysis". While "processing axioms" are always consistently time-consuming, "invariant analysis" is heuristically done and sometime can be quick while some other times can be very time consuming.

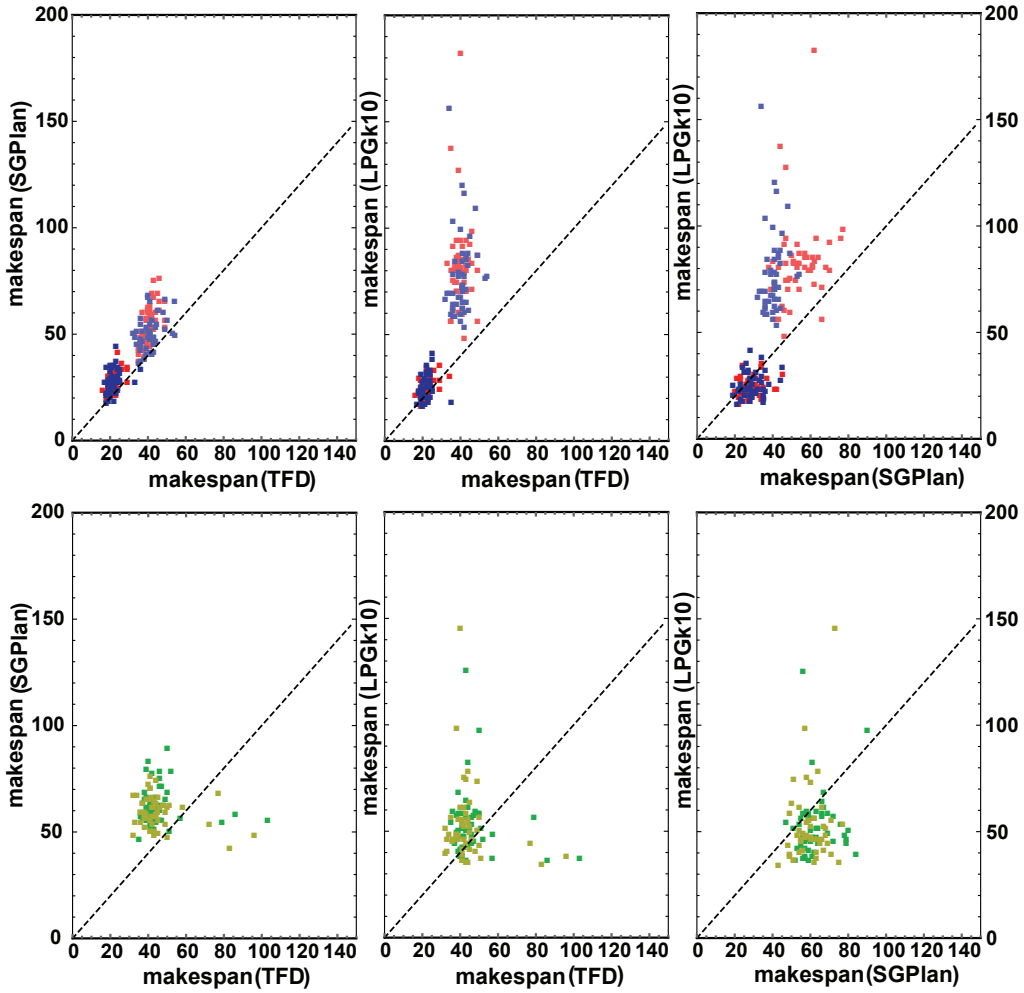


Figure 4: Instance-by-instance comparison of *SGPlan*, *TFD* and *LPG*. Top panel shows results for N=8: red dots indicate instances with u=90% while blue dots are for u=100%. Lower makespan data points refer to p=1 while higher makespans refer to p=2 (see Table 1). Bottom panel shows results for $N = 21$: Green indicates u=90% and yellow u=100%.

spent all of the 60 minutes time limit on pre-processing for $N = 40$ problems. LPG can generally find the first solution quicker than TFD (still much slower than SGPlan) but does not improve the solution quality as quickly as TFD over the allocated timelimit. We also tested YAHSP3-mt (Vidal 2014), another recent award winning temporal planner, but it did not return any solution.

Plan quality comparison: to compare the plan quality across planners, we use the formula employed by the IPCs to grade planners in the temporal planning track since IPC6 (Helmert, Do, and Refanidis 2008): for each planning instance i, if the best-known makespan is produced by a plan P_i, then for a given planner X that returns a plan P^i_X for i, the score of P^i_X is calculated as: $makespan(P_i)$ divided by $makespan(P^i_X)$. A comparative value closer to 1.0 indicates that planner X produces better quality plan for instance i. We use this formula and average the score for our three tested planners over the instance ensembles that are completely solved by the time cutoff. Table 2 compares different planners with regard to plan quality. For N = 8 and p = 1, TFD found the best or close to the best quality plans. LPG is about 15% worse while SGPlan, which unlike TFD and LPG only find a single solution, produces lower quality plans. The comparison results for $N = 21$ and $p = 1$ is similar. For N = 8 and p = 2, TFD again nearly always produces the best quality plan. However, for this more complex case, SGPlan produces overall better quality plans compared to LPG, even though LPG returns multiple plans for each instance.

Figure 4 shows in further detail the head-to-head makespan comparison between different pairs of planners, specifically pairwise comparisons between TFD, SGPLan, and LPG: TFD always dominates SGPlan, TFD dominates LPG majority of the times, and SGPlan dominates LPG on bigger problems, but is slightly worse on smaller problems.

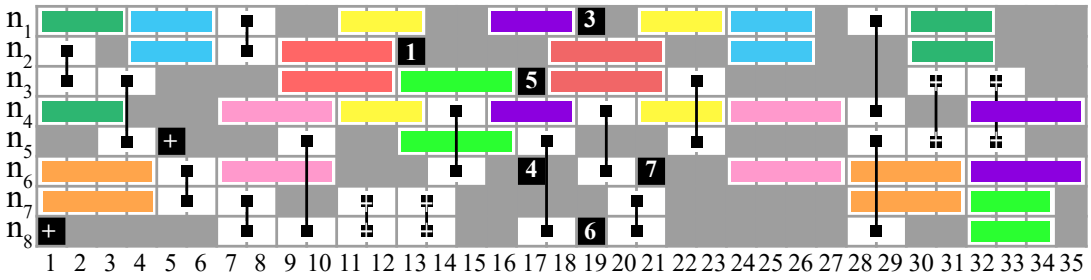


Figure 5: Compilation of $p = 2$ QAOA performed by TFD for the MaxCut problem depicted in Fig. 2 on the $N = 8$ grid in Fig. 1; with time-step on the x-axis and qubit locations on the y-axis. Each row indicates what gate operates on each qubit at a given time-step during the plan. Colored blocks represents p-s gates (of duration 3 or 4 depending on whether they are RED or BLUE). White blocks are swap gates. Gates synchronized in pair, since they involve 2-qubit. Black blocks with numbers are mix gates acting on the corresponding state. Gates marked with a + indicate superfluous gates that could be detected and eliminated in post-processing.

Planning time comparison: Both TFD and LPG use "anytime" search algorithms and use all of their allocated time to try finding gradually better quality plans. In contrast, SGPlan returns a single solution and thus generally take a very short amount of time with the median solving time for SGPlan in p=1$|N_8$, p=1$|N_{21}$, P=1$|N_{40}$ and P=2$|N_8$ are 0.02, 1, 25, and 0.05 seconds[5].

Other planners: we also conducted tests on: *VHPOP*, *HSP**, *CPT*, and POPF. While LPG, SGPlan, and TFD were selected for their ability to solve large planning problems, we hoped that HSP*, CPT, and VHPOP would return optimal plans to provide a baseline for plan quality estimation. Unfortunately, HSP*, CPT, VHPOP (and also POPF) failed to find a single plan even for our smallest problems for various reason: CPT underwent internal errors after a quick search time, VHPOP ran out of memory quickly, while HSP* couldn't find any plan for a cutoff time of 2 hours. POPF, which does not guarantee finding optimal plans, but produced good quality plans for other temporal planning domain, also does not find any solution.

Discussion: Our preliminary empirical evaluation shows that the test planners provide a range of tradeoffs between scalability and plan quality. At one end, we have SGPlan that can scale up to large problems and solve them in a short amount of time while providing reasonably good quality plans (compared to the best known solutions). At the other end, we have TFD, which utilizes all of the allocated time to find the best quality solutions but in general is the slowest by far to come up with some valid solution. LPG balances between the two: it can either find one solution quickly like SGPlan or can utilize the whole cutoff time to find better quality solutions. Since planning is exponentially hard with regard to the problem size (i.e., number of state variables and actions), being able to partition it into sub-problems of smaller sizes definitely help SGPlan to find a valid solution quickly. However, there are several reasons that TFD and LPG can find overall better quality solutions: (i) their anytime algorithms allows them to gradually find better quality plans using the previously found plans as baseline for pruning unpromissing search directions; (ii) SGP's partitioning algorithm is based on logical relationship between state variables and actions and ignores all temporal aspects. Thus, combining plans for sub-problems using logical global constraints can lead to plans of lower quality for time-sensitive objective function such as minimizing the plan makespan.

What's missing from our analysis is the assessment on how good the quality of the best found plans compared to the optimal solutions. At the moment, there is no published work on finding optimal solution for this problem and as outlined above, our current effort in getting the existing optimal-makespan planners to find solutions have not been fruitful. This is one important future research direction. Based on the "eye-test" and manual analysis, the best plans returned are usually of good quality but not without defects. Figure 5 shows a visualization, in a 'Gantt chart' format, of a plan found by TFD for the problem instance depicted in Figure 2. In this plan, qstate q_1 initially located at n_1, is undergoing the following sequence of actions:

$$\begin{aligned}\text{P-S}_3(q_1,q_4) &\rightarrow \text{P-S}_3(q_1,q_3) \rightarrow \text{P-S}_4(q_1,q_5) \rightarrow \text{MIX}(q_1)\\ &\rightarrow \text{WAIT}(4) \rightarrow \text{P-S}_4(q_1,q_5) \rightarrow \text{WAIT}(2)\\ &\rightarrow \text{P-S}_3(q_1,q3) \rightarrow \text{WAIT}(3) \rightarrow \text{P-S}_3(q_1,q_4)\end{aligned}$$

where we denote the duration of the P-S gates in subscript and we introduced an WAIT gate to indicate inaction times. The second mixing phase is trivially scheduled at the end of the last tasks for each qstate. The example plan shown in Figure 5, pictures the compilation of the problem instance in Fig. 2. This plan has a very short makespan, but contains some unnecessary gates. Examples are the repeated swaps at time 11 and 30, and the mixing of the un-utilized logical states q_2 and q_8 at times 1,5. These spurious gates/actions do not affect the makespan, and they can be identified and eliminated by known plan post-processing techniques (Do and Kambhampati 2003). We also believe a tighter PDDL

[5]For comparison purpose, LPG-quality, which also tries to return a single good quality solution, produces the median solving time for P=1$|N_8$ and P=2$|N_8$ are 0.9 and 70 seconds respectively.

model will help eliminate extra gates.

6 Conclusion and Future Work

In this paper we presented a novel approach to the problem of compiling idealized quantum circuits to specific quantum hardware, focusing our experiments on QAOA circuits. Our presentation and tests have been focused on the pedagogical and practically relevant example of MaxCut, but the approach is sufficiently general to be applied to any discrete optimization problem, such as max E3LIN2 (Farhi, Goldstone, and Gutmann. 2014b). Three well-established temporal planners were able to compile the QAOA circuits with reasonable efficiency, demonstrating the viability of this approach.

This work paves the way for potentially impactful future work on the use of artificial intelligence methods for quantum computing. In future work, we plan to further tune the performance of the planners, including choosing an initial assignment of qstates to qubits favorable for compilation. In order to scale reliably to larger plan sizes we will develop decomposition approaches in which $p > 1$ could be divided into multiple $p = 1$ problems to be solved independently and matched in a post-processing phase. We will also compare with other approaches to this compilation problem such as sorting networks (Beals et al. 2013; Brierly 2015; Bremner, Montanaro, and Shepherd. 2016), and we will look at parameters values for the durations that match existing hardware, in collaboration with experimental groups. A virtue of the planning approach is that the framework is very flexible with respect to features of the hardware graph, including irregular structures. Moreover, we will include in the PDDL modeling additional features that are characteristics of quantum computer architectures, such as the crosstalk effects of 2-qubit gates and the ability to *quantum teleport* quantum states across the chip (Copsey et al. 2003). We will also consider other families of quantum circuits and more sophisticated measures against which to optimize the compilation beyond simply the duration of the cirucit. We believe that this approach could be of great interest for the leading community that is developing low-level quantum compilers for generic architectures (Steiger, Häner, and Troyer 2016b; Häner et al. 2016) and for the designers of machine-instructions languages for quantum computing (Smith, Curtis, and Zeng 2016b; Bishop 2017).

7 Acknowledgements

The authors acknowledge useful discussions with Will Zheng, Robert Smith and Bryan O'Gorman. The authors appreciate support from the NASA Advanced Exploration Systems program and NASA Ames Research Center (Sponsor Award No. NNX12AK33A). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purpose notwithstanding any copyright annotation thereon.

References

[Barends et al. 2016] Barends, R.; Shabani, A.; Lamata, L.; Kelly, J.; Mezzacapo, A.; Heras, U. L.; Babbush, R.; Fowler, A. G.; Campbell, B.; Chen, Y.; et al. 2016. Digitized adiabatic quantum computing with a superconducting circuit. *Nature* 534(7606):222–226.

[Beals et al. 2013] Beals, R.; Brierley, S.; Gray, O.; Harrow, A. W.; Kutin, S.; Linden, N.; Shepherd, D.; and Stather, M. 2013. Efficient distributed quantum computing. *Proceedings of the Royal Society A.* 469(2153):767 – 790.

[Bhattacharjee and Chattopadhyay 2017] Bhattacharjee, D., and Chattopadhyay, A. 2017. Depth-optimal quantum circuit placement for arbitrary topologies. *arXiv preprint arXiv:1703.08540*.

[Bishop 2017] Bishop, L. S. 2017. Qasm 2.0: A quantum circuit intermediate representation. *Bulletin of the American Physical Society* 62.

[Boxio 2016] Boxio, S. 2016. Characterizing quantum supremacy in near-term devices. In *arXiv preprint arXiv:1608.0026*.

[Bremner, Montanaro, and Shepherd. 2016] Bremner, M. J.; Montanaro, A.; and Shepherd., D. J. 2016. Achieving quantum supremacy with sparse and noisy commuting quantum computations. In *arXiv preprint arXiv:1610.01808*.

[Brierly 2015] Brierly, S. 2015. Efficient implementation of quantum circuits with limited qubit interactions.". In *arXiv preprint arXiv:1507.04263*.

[Chen and Wah 2006] Chen, Y., and Wah, B. 2006. Temporal planning using subgoal partitioning and resolution in sg-plan. *Journal of Artificial Intelligence Research* 26:323 – 369.

[Copsey et al. 2003] Copsey, D.; Oskin, M.; Impens, F.; Metodiev, T.; Cross, A.; Chong, F. T.; Chuang, I. L.; and Kubiatowicz, J. 2003. Toward a scalable, silicon-based quantum computing architecture. *IEEE Journal of selected topics in quantum electronics* 9(6):1552–1569.

[Devitt 2016] Devitt, S. J. 2016. Performing quantum computing experiments in the cloud. *Physical Review A* 94(3):222–226.

[Do and Kambhampati 2003] Do, M. B., and Kambhampati, S. 2003. Improving the temporal flexibility of position constrained metric temporal plans. In *Proceedings of the* 13^{th} *International Conference on Artificial Intelligence Planning and Scheduling (ICAPS)*.

[Erdös and Rényi 1960] Erdös, P., and Rényi, A. 1960. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences* 5:569–573.

[Eyerich, Mattmüller, and Röger 2009] Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of the* 19^{th} *International Conference on Automated Planning and Scheduling*, 318 – 325.

[Farhi et al. 2017] Farhi, E.; Goldstone, J.; Gutmann, S.; and

Neven, H. 2017. Quantum algorithms for fixed qubit architectures. *arXiv preprint arXiv:1703.06199*.

[Farhi, Goldstone, and Gutmann. 2014a] Farhi, E.; Goldstone, J.; and Gutmann., S. 2014a. A quantum approximate optimization algorithm. In *arXiv preprint arXiv:1411.4028*.

[Farhi, Goldstone, and Gutmann. 2014b] Farhi, E.; Goldstone, J.; and Gutmann., S. 2014b. A Quantum Approximate Optimization Algorithm Applied to a Bounded Occurrence Constraint Problem. In *arXiv preprint axXiv:1412.6062*.

[Fu, Wilken, and Goodwin 2005] Fu, C.; Wilken, K.; and Goodwin, D. 2005. A faster optimal register allocator. *The Journal of Instruction-Level Parallelism* 7:1 – 31.

[Gerevini, Saetti, and Serina 2003] Gerevini, A.; Saetti, L.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research* 20:239 – 290.

[Häner et al. 2016] Häner, T.; Steiger, D. S.; Svore, K.; and Troyer, M. 2016. A software methodology for compiling quantum programs. *arXiv preprint arXiv:1604.01401*.

[Helmert, Do, and Refanidis 2008] Helmert, M.; Do, M.; and Refanidis, I. 2008. The 2008 international planning competition: Deterministic track. http://icaps-conference.org/ipc2008/deterministic/. 2008-02-19.

[IBM 2017] IBM. 2017. The ibm quantum experience. http://www.research.ibm.com/quantum/. 2017-02-19.

[Ramasesh et al. 2017] Ramasesh, V.; O'Brien, K.; Dove, A.; Kreikebaum, J. M.; Colless, J.; and Siddiqi, I. 2017. Design and characterization of a multi-qubit circuit for quantum simulations. In *March Meeting 2017*. American Physical Society.

[Rieffel and Polak 2011] Rieffel, E. G., and Polak, W. H. 2011. *Quantum computing: A gentle introduction*. MIT Press.

[Rintanen 2012] Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193:45 – 86.

[Sete, Zeng, and Rigetti 2016] Sete, E. A.; Zeng, W. J.; and Rigetti, C. T. 2016. A functional architecture for scalable quantum computing. In , *IEEE International Conference on Rebooting Computing (ICRC)*.

[Smith, Curtis, and Zeng 2016a] Smith, R. S.; Curtis, M. J.; and Zeng, W. J. 2016a. A practical quantum instruction set architecture. In *arXiv preprint arXiv:1608.03355*.

[Smith, Curtis, and Zeng 2016b] Smith, R. S.; Curtis, M. J.; and Zeng, W. J. 2016b. A practical quantum instruction set architecture. *arXiv preprint arXiv:1608.03355*.

[Steiger, Häner, and Troyer 2016a] Steiger, D. S.; Häner, T.; and Troyer, M. 2016a. Projectq: An open source software framework for quantum computing. In *arXiv preprint arXiv:1612.08091*.

[Steiger, Häner, and Troyer 2016b] Steiger, D. S.; Häner, T.; and Troyer, M. 2016b. Projectq: An open source software framework for quantum computing. *arXiv preprint arXiv:1612.08091*.

[Venturelli et al. 2017] Venturelli, D.; Do, M.; Rieffel, E.; and Frank, J. 2017. Compiling quantum circuits to realistic hardware architectures using temporal planners. In *arXiv preprint (https://ti.arc.nasa.gov/m/groups/asr/planning-and-scheduling/VentCirComp17_ArXiv.pdf)*.

[Versluis et al. 2016] Versluis, R.; Poletto, S.; Khammassi, N.; Haider, N.; Michalak, D.; Bruno, A.; Bertels, K.; and DiCarlo, L. 2016. Scalable quantum circuit and control for a superconducting surface code. *arXiv preprint arXiv:1612.08208*.

[Vidal 2014] Vidal, V. 2014. Yahsp3 and yahsp3-mt in the 8th international planning competition. In *Proceedings of the 8th International Planning Competition (IPC14)*.

[Wah and Chen 2004] Wah, B., and Chen, Y. 2004. Subgoal partitioning and global search for solving temporal planning problems in mixed space. *International Journal on Artificial Intelligence Tools* 13(4):767 – 790.

[Wecker and Svore 2014] Wecker, D., and Svore, K. M. 2014. Liqui $|>$: A software design architecture and domain-specific language for quantum computing. In *arXiv preprint arXiv:1402.4467*.