The Thirty-First International Florida
Artificial Intelligence Research Society Conference (FLAIRS-31)

Alert Generation in Execution Monitoring Using Resource Envelopes

T. K. Satish Kumar
tkskwork@gmail.com

Hong Xu
hongx@usc.edu

Zheng Tang, Anoop Kumar, Craig Milo Rogers, Craig A. Knoblock
{zhengtan, anoopk, rogers, knoblock}@isi.edu
Information Sciences Institute, University of Southern California,
Marina del Rey, California 90292, USA

Abstract

A *simple temporal network* (STN) can often be used to represent the flexibility in the execution of a plan. Nodes represent the execution times of actions and directed edges represent constraints between them. An *STN with resources* (STNR) is an STN in which each node is associated with production or consumption levels of resources. The upper (lower) *resource envelope* of an STNR is the maximum (minimum) accumulated *resource levels* at every time instant over all possible executions. In this paper, we discuss the usefulness of resource envelopes in the context of execution monitoring. We show that they can be used in a tractable framework for forward projection of world states during plan execution. This allows an execution monitor to recognize depletion of resources early and to generate alerts with large look-aheads. It also supports retasking by enabling "what-if reasoning" during plan execution.

Introduction

An agent that intends to achieve certain goal conditions in an interactive environment is really involved in a game against nature and other agents constituting that environment. In this game-theoretical setting, the agent is required to take an action in response to every situation it finds itself in. In other words, the agent needs to design a *policy* for achieving its goal conditions (Russell and Norvig 2009, Chap. 17). However, policy generation, in general, is computationally very hard; and therefore, the agent often benefits from making certain assumptions about the environment.

Automated planning techniques are used to focus on the agent's ability to compose individual actions into a goal-achieving sequence of actions (in a relatively simplified model of the environment). Compared to policy generation, plan generation is easier but is still PSPACE-hard (Russell and Norvig 2009, Chap. 11). The state-of-the-art automated planners in fact use a plethora of different heuristics and other algorithmic techniques (Ghallab, Nau, and Traverso 2004). Although research in automated planning has addressed richer models of the environment, the focus is still on the capability of the planning agent but not on the game-theoretical strategies of nature or other agents.

Because automated planning focuses on plan generation using certain assumptions on the environment, it is imperative for us to remove these assumptions in the concomitant problem of execution monitoring. In particular, a plan generated by the planning algorithm may not lead to the intended outcome when executed. For example, if there are exogenous factors that were not considered by the planning algorithm, the plan execution could fail or not lead to the intended outcome. This emphasizes the importance of execution monitoring despite the fact that it is not as well studied as automated planning (Pettersson 2005). In execution monitoring, the agent gathers partial observations from the environment while it executes the plan. The intention is to make sure that the observations made during plan execution do not invalidate the feasibility of the remaining plan.

At any point of time during execution, if the remaining plan is close to becoming infeasible, an alert has to be raised to invoke *retasking* or *replanning*. Retasking adjusts the current plan within the space of the same causal interactions between its actions. If retasking options are not available, or the remaining plan is conclusively infeasible, replanning is invoked. Replanning is significantly more expensive than retasking, and therefore, intelligent alert generation is important for minimizing the number of times that replanning is invoked. This in turn contributes to the successful integration of automated planning and execution monitoring.

Intelligent alert generation requires a forward projection of the current world state into all possible future world states. This look-ahead is important in the same sense that forward checking is important in constraint satisfaction (Dechter 2003). If all future world states are invalidated by the observations made so far, a *strong alert* is generated to invoke replanning. If a subset of the world states is invalidated by the observations made so far, a *weak alert* is generated to invoke retasking. In either case, computing all future world states for a certain look-ahead is computationally expensive since the number of such future world states can be exponentially large in the length of the look-ahead. This typically limits the look-ahead that can be used for alert generation in execution monitoring. Therefore, the infeasibility of the plans is not recognized early enough to take remedial measures.

The flexibility in the execution of a plan can often be represented by a *simple temporal network* (STN), where nodes

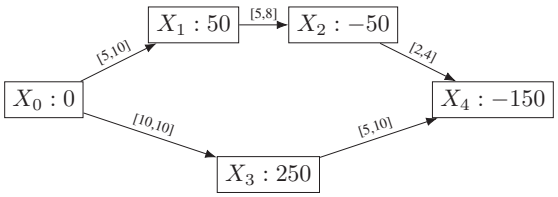


Figure 1: Shows an example of an STNR. Each square is a node. The labels X_0, X_1, X_2, X_3 and X_4 represent events, and the numbers after them indicate their resource levels. Each directed edge represents a simple temporal constraint specified using a lower and an upper bound. For example, X_1 and X_2 are connected by a directed edge that encodes the simple temporal constraint $5 \leq \tau(X_2) - \tau(X_1) \leq 8$.

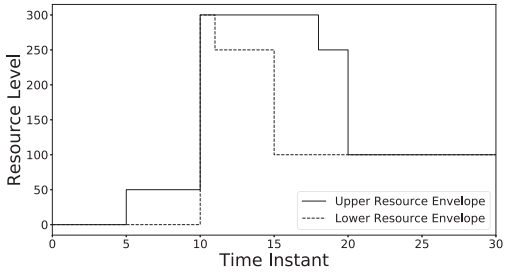


Figure 2: Shows the resource envelope of the STNR in Figure 1. The solid line shows the upper resource envelope and the dashed line shows the lower resource envelope. (At some time instants, the upper and lower bounds are equal, and are thus indicated together by the solid lines.)

represent the execution times of actions and directed edges represent constraints between them. In such a framework, we introduce the idea of *resource envelopes* to do forward projection of world states in polynomial time using a reduction to maxflow (Kumar 2003). This allows an execution monitor to recognize depletion of resources early and to generate alerts with large look-aheads. It also supports retasking by enabling "what-if reasoning" during plan execution.

Informally, given a set of events that produce or consume resources and the time constraints between these events (i.e., a plan), the upper and lower resource envelopes describe the highest and lowest possible total resource levels at each time instant over all possible executions of the plan. In this paper, we show how to apply resource envelopes for alert generation in various application domains, such as task scheduling in smart homes, food delivery systems, and system and service management.

Background and Formalization

The STN, first proposed in (Dechter, Meiri, and Pearl 1991), is a graphical representation of a collection of simple temporal constraints between the execution times of various events. It is popularly used in temporal reasoning for expressing fairly rich quantitative constraints while also being able to solve them in polynomial time. Autonomous space exploration (Knight et al. 2001), domestic activity management (Pecora and Cirillo 2009) and job scheduling on servers (Ji, He, and Cheng 2007) are just a few applications of STNs.

Formally, an STN S is defined on a directed graph $G = \langle \mathcal{X}, \mathcal{E} \rangle$, where $\mathcal{X} = \{X_0, X_1, \ldots, X_n\}$ is the set of nodes representing events and $\mathcal{E}$ is the set of directed edges between them representing simple temporal constraints. A *schedule* τ is a function that maps each node to a time instant at which the corresponding event should be executed. For any schedule τ, $\tau(X_0)$ is set to 0 by convention to establish a frame of reference. Each directed edge $e_{ij} = (X_i, X_j) \in \mathcal{E}$ is associated with a pair of non-negative real numbers $[LB(e_{ij}), UB(e_{ij})]$, representing the simple temporal constraint $LB(e_{ij}) \leq \tau(X_j) - \tau(X_i) \leq UB(e_{ij})$. A schedule is said to be *consistent* iff it satisfies all constraints given by the edges in $\mathcal{E}$. For a given STN S, the corresponding *simple temporal problem* (STP) is to find a consistent schedule for S if it exists. Throughout this paper, let $T(S)$ be the set of all consistent schedules of S.

An *STN with resources* (STNR) is defined as an STN in which each node $X \in \mathcal{X}$ is associated with a *resource level* $r(X)$. $r(X)$ is a real number that represents "how much X affects the resource": If $r(X)$ is positive, X is a *producer* in $\mathcal{P}$ and generates $r(X)$ amount of the resource upon execution; if $r(X)$ is negative, X is a *consumer* in $\mathcal{C}$ and depletes $-r(X)$ amount of the resource upon execution; if $r(X)$ is zero, it does not change the amount of the resource. $r(X_0)$ is always assumed to be zero. Figure 1 shows an example of the STNR. The *total resource level* $R(t, \tau)$ of a consistent schedule τ at a specific time instant t is defined as the sum of all resource levels of all events that have been executed no later than t as specified by τ, i.e, $R(t, \tau) = \sum_{X \in \{X' | \tau(X') \leq t\}} r(X)$.

The upper (lower) resource envelope of an STNR is the maximum (minimum) accumulated resource levels at every time instant considering all possible executions. Formally, it is defined as follows. We first define the upper bound $R_U(t)$ (and lower bound $R_L(t)$) of the total resource level at a certain time instant t of a given STNR as the highest (and lowest) total resource levels over all consistent schedules, i.e., $R_U(t) = \max_{\tau \in T(S)} R(t, \tau)$ (and $R_L(t) = \min_{\tau \in T(S)} R(t, \tau)$). We call the functions $R_U(\cdot)$ and $R_L(\cdot)$ of an STNR as its upper and lower resource envelopes, respectively. The resource envelope problem on an STNR is to compute its upper and lower resource envelopes (Kumar 2003). Figure 2 shows a simple example.

Computing Resource Envelopes

In this section, we review how to construct the upper and lower resource envelopes R_U and R_L of an STNR (Kumar 2003). The lower resource envelope R_L can be constructed using a procedure for constructing the upper resource envelope R_U by switching the roles of producers and consumers. Therefore, we only show the construction of the upper resource envelope R_U. Algorithm 1 shows how to compute the upper bound $R_U(t)$ of the total resource level at time instant t. This algorithm runs in polynomial time since it can

Algorithm 1: Compute the upper bound for a given STNR at a specific time instant t.

Function UPPER-ENVELOPE-AT-T(S, t)
- **Input:** An instance of the resource envelope problem on an STNR S.
- **Input:** A time instant t.
- **Output:** The upper bound $R_U(t)$ at time instant t.
- • *Construct the distance graph $\mathcal{D}(S)$ on the nodes of S as follows:*
  - **for** *each edge* $e = \langle X_i, X_j \rangle \in E$ **do**
    - Add $\langle X_i, X_j \rangle$ annotated with $UB(e)$;
    - Add $\langle X_j, X_i \rangle$ annotated with $-LB(e)$;
  - Let $dist(X_i, X_j)$ denote the length of the shortest path from X_i to X_j in $\mathcal{D}(S)$;
- • *Build a vertex-weighted directed graph $E(S)$ as follows:*
  - The vertices of $E(S)$ correspond to events in $\mathcal{P} \cup \mathcal{C}$;
  - The weight on a vertex X_i is set to $|r(X_i)|$;
  - **for** *each* $X_p \in \mathcal{P}$ *and* $X_c \in \mathcal{C}$ **do**
    - **if** $dist(X_p, X_c) \leq 0$ **then**
      - Add a directed edge $\langle X_p, X_c \rangle$;
- • *Construct a graph $M(S)$ from $E(S)$ as follows:*
  - Remove a production node $X_p \in \mathcal{P}$ and all its incident edges iff $t + dist(X_p, X_0) < 0$;
  - Remove a consumption node $X_c \in \mathcal{C}$ and all its incident edges iff $dist(X_0, X_c) - t < 0$;
- Compute $Q = \{u_1, u_2, \ldots, u_k\}$ as the largest weighted independent set in $M(S)$;
- **return** $R_U(t) := \sum_{y_i \in \mathcal{P} \cap Q} |r(y_i)| - \sum_{y_i \in \mathcal{C} \setminus Q} |r(y_i)|$;

Algorithm 2: Compute the upper resource envelope for a given STNR.

Function UPPER-ENVELOPE-ALL-T(S)
- **Input:** An instance of the resource envelope problem on an STNR S.
- **Output:** R_U of S.
- **for** *each* $X_p \in \mathcal{P}$ **do**
  - Insert $-dist(X_p, X_0)$ into list L;
- **for** *each* $X_c \in \mathcal{C}$ **do**
  - Insert $+dist(X_0, X_c)$ into list L;
- Sort L in ascending order $\langle d_1, d_2, \ldots, d_{|L|} \rangle$;
- **for** $i = 1, 2, \ldots, |L| - 1$ **do**
  - $R_U(d_i) :=$ UPPER-ENVELOPE-AT-T(S, d_i);
  - $R_U(t) := R_U(d_i)$ for all $t \in [d_i, d_{i+1})$;
- $R_U(t) := R(d_{|L|})$ for all $t \in [d_{|L|}, +\infty)$;
- $R_U(t) := 0$ for all $t \in [0, d_1)$;
- **return** R_U;

be reduced to computing maxflow on bipartite graphs. Algorithm 2 shows how to construct the entire upper resource envelope R_U. This algorithm also terminates in polynomial time since it only makes a linear number of calls to Algorithm 1. The upper resource envelope R_U is a piecewise constant function with a linear number of discontinuities. The proof for the correctness of these two algorithms has been shown in (Kumar 2003).

Application in the Smart Home Domain

In this section, we present a running example from the smart home domain (van den Briel, Scott, and Thibaux 2013; Kumar et al. 2018). Here, an intelligent scheduler first elicits user-specified constraints for when to run various appliances. It then provides a schedule that tries to satisfy these constraints and resolve various resource contentions. Figure 3 shows an example. Here, the washer can run anytime between 30 and 35 minutes, after which the dryer runs between 45 and 50 minutes. The dryer starts running between 5 and 30 minutes after the washer stops. The refrigerator should be running for the entire day, except for a break of less than 40 minutes. The rice cooker runs between 20 and 30 minutes and must finish cooking before the dinner time, e.g., 6:00pm, and after a certain time to keep the cooked food fresh, e.g., 5:00pm. The dishwasher runs between 50 and 70 minutes and should start after dinner time, e.g., 8:00pm, but before the end of the day. The water heater boils water between 40 and 45 minutes and must finish boiling (thus producing a certain amount of hot water) before any of the washer, the dishwasher or the rice cooker starts running. All appliances use electricity when they are running and the total electricity power is limited. The washer, the dishwasher, and the rice cooker also use hot water as a resource.

For a planner or scheduler that generates the flexible schedule in Figure 3, the execution monitor can construct the upper and lower resource envelopes for it as shown in Figure 4a. If certain exogenous information such as an electrical power limit of 1350 watts becomes available only at execution time, the execution monitor can use the lower resource envelope and generate an alert. In particular, it can inform the planner or scheduler that there exists a schedule with a total demand of electrical power greater than the electrical power limit within the time interval [1200,1440]. The planner or scheduler can then "retask" the schedule and constrain it further using the temporal constraint indicated by the dashed line in Figure 3. The upper and lower resource envelopes are recomputed by the execution monitor as shown in Figure 4b. Since the lower resource envelope is above the electrical power limit, no alert is generated until new exogenous information becomes available at execution time.

Application in the Food Delivery Domain

One potential application of resource envelopes is in the food delivery domain. In recent years, thanks to the emergence of Internet-based food delivery services, the food delivery industry has grown fast and is expected to continue to grow (Pigatto et al. 2017; Bajaj and Mehendale 2016). Most food delivery companies, such as GrubHub and delivery.com, follow the traditional strategy for delivering, i.e., they first take orders from customers and then deliver food to them. However, this traditional strategy often necessitates customers to wait for a long period of time after they place their orders.

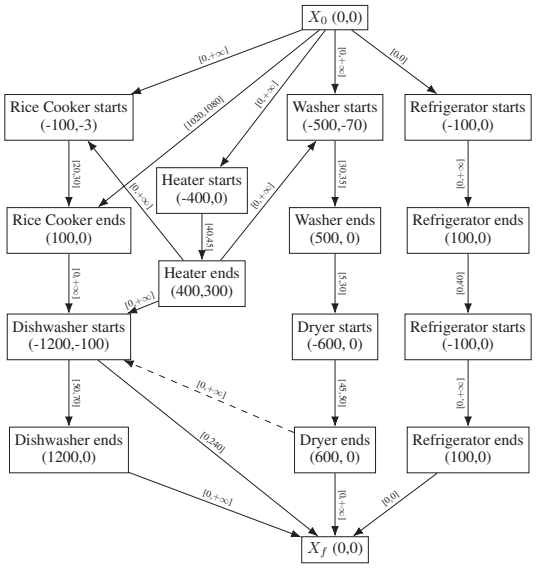
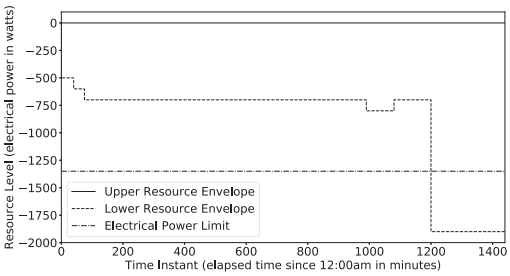
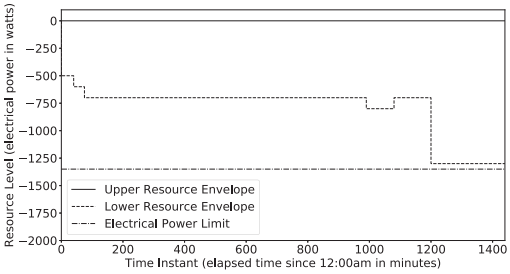


Figure 3: Shows the STNR representing an example application instance in the smart home domain. The meanings of the symbols are similar to those in Figure 1, except that each node is associated with two different types of resources (electrical power in watts and hot water in liters) as shown inside the braces in it. The bounds in the temporal constraints are in minutes. X_0 represents the beginning of the day, i.e., 12:00am. X_f represents the end of the day, i.e., the very moment before 12:00am of the next day. An implicit temporal constraint between X_0 and X_f (with both upper and lower bounds being 1440, the number of minutes in a day) is omitted in the figure. The temporal constraint represented by the dashed line indicates how the schedule should be further constrained in recognition of an alert generated by resource envelopes as shown in Figure 4.



(a) The upper and lower resource envelopes of electrical power consumption without the dashed line temporal constraint in Figure 3.



(b) The upper and lower resource envelopes of electrical power consumption with the dashed line temporal constraint in Figure 3.

Figure 4: Shows the resource envelopes for the example in Figure 3. The meanings of the symbols are similar to those in Figure 2. We assume that the total electrical power limit is 1350 watts.

To avoid long waiting times, a new real-time food delivery strategy has recently emerged. In this new strategy, food delivery cars roam the neighborhoods of a city, carrying popular menu items in anticipation of nearby customers placing orders. When an order is placed, a nearby driver carrying this menu item is assigned to deliver it. This proactive strategy achieves shorter waiting times. A real-world example is Uber's "instant delivery" service (Dave 2016; van Grove 2016).

Despite the arguments in favor of this new strategy, practical issues have rendered it unviable. For example, Uber's "instant delivery" service did not last long (Said 2016; Sidman 2016). In this section, we propose the use of resource envelopes to be used in a hybrid strategy that combines the traditional and proactive real-time food delivery strategies.

We assume that a fraction of the customers order food before the food delivery cars are en route. We refer to these orders as pre-orders. Orders placed by the rest of the customers when the food delivery cars are en route are referred to as post-orders. For pre-orders, a central planner makes an individual plan for each car. These plans are susceptible to exogenous factors such as traffic conditions, but are also required to adhere to customer satisfaction requirements. Due to such exogenous factors, there are simple temporal constraints between pickup and delivery events in each individual plan. To satisfy post-orders, nearby cars should be able to deviate from their current individual plans. This food delivery strategy therefore calls for a forward projection of the current world state to identify imminent shortages of popular menu items in individual cars. For this reason, we propose the use of resource envelopes to monitor the amount of food in each car in the context of its current plan and nearby post-orders.

Specifically, we model the individual plan of each car as an STNR. In such an STNR, each pickup and delivery event is a node. The change in the amount of popular menu items in the car during a pickup or delivery event is the resource level associated with the corresponding node in the STNR. In this model, monitoring the amount of popular menu items in each car can be done by computing the resource envelopes.

Figure 5 shows an example in the food delivery domain. r_1 and r_2 represent restaurants' locations where pickup events occur and $c_1, c_2, \ldots, c_6$ represent customers' locations where delivery events occur. If the food delivery car does not have to cater to any post-orders, the best path for it is $[r_1, c_1, c_2, c_3, r_2, c_4, c_5, c_6]$. The corresponding resource

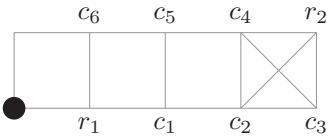


Figure 5: Shows a food delivery example. The grid represents a street network. Each horizontal and vertical edge has a unit length. The car is represented by the dark circle and is initially located at the bottom-left corner. We assume that the car takes 1 to 2 time units to traverse 1 unit length. r_* and c_* are the locations of pickup events (restaurants) and delivery events (customers), respectively. r_1 has a production of 4 and r_2 has a production of 5. All c_*'s have a production of -1.

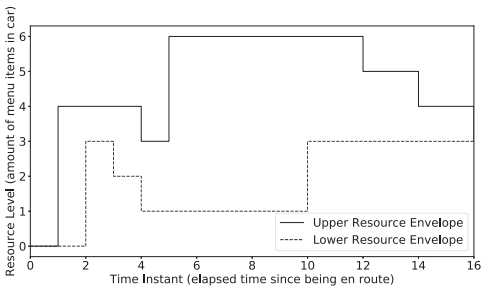
envelopes are shown in Figure 6a. However, in anticipation of post-orders, if we require the car to hold at least 2 units of menu items after its first pickup event, this path would exhibit the risk of being low on the amount of menu items in the time interval [4,10]. Nonetheless, if the planner adjusts the path to be $[r_1, c_1, c_2, r_2, c_3, c_4, c_5, c_6]$, the corresponding resource envelopes, as shown in Figure 6b, indicate that this new plan would not exhibit this risk.

Application in Service Management Domain

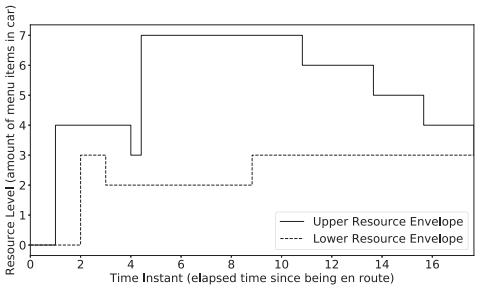
Another potential application of resource envelopes is in scheduling and managing services during the startup of operating systems. Typically, when an operating system boots, it starts a set of programs, called services, that persistently wait for and process incoming requests, but mostly remain idle in the background. During system boot, starting a service uses some resources such as CPU cores, RAM and external equipment, and may also depend on the availability of other services. For example, let us suppose that starting service A depends on the availability of service B: A could be a web server and B could be a database management system that allows SQL queries via a socket connection; or, A could depend on the availability of files in a partition of the disk, which is made available by a filesystem mounting service B.

In modern system and service managers, such as systemd (Poettering, Sievers, and Others 2017), to achieve a fast system boot, this dependency is often not required to be strictly chronological. In the first previous example, when A is a web server and B is a database management system, both A and B can start simultaneously. But this can lead to a situation where A needs to send an SQL query to B while B is still not ready to process the SQL query. In such a situation, the system and service manager can buffer the SQL query and forward it to B after B is ready. In the second previous example, when A depends on B for a disk partition, once again, A and B can start simultaneously. But, here too, A may need to access files before B finishes mounting that disk partition. In such a situation, the system and service manager can suspend A temporarily until B mounts the disk partition.

Although dependencies are not strictly chronological, they need to respect some temporal constraints. Typically



(a) The upper and lower resource envelopes of the amount of menu items in the car from the example in Figure 5 for the path $[r_1, c_1, c_2, c_3, r_2, c_4, c_5, c_6]$.



(b) The upper and lower resource envelopes of the amount of menu items in the car from the example in Figure 5 for the path $[r_1, c_1, c_2, r_2, c_3, c_4, c_5, c_6]$.

Figure 6: Shows the resource envelopes for two different paths of the car from the example in Figure 5. The meanings of the symbols are similar to those in Figure 2.

these constraints are simple temporal in nature. Continuing the first previous example, A itself may report a timeout error if its waiting time for a response to the SQL query from B exceeds a limit. Continuing the second previous example, A may report a file IO error if its waiting time to access the file exceeds a limit. These timeout constraints can be modeled as simple temporal constraints. Violating these constraints leads to failure of starting services.

In this application domain, there are two types of constraints: (a) the simple temporal timeout constraints and (b) the resource capacity constraints such as CPU cores, RAM and access to external equipment. Both temporal and resource constraints must be satisfied for a successful system boot. Resource envelopes can be used to analyze the interactions between temporal and resource constraints. The implication of tightening or relaxing a temporal constraint on the resource requirement can be quickly quantified. This allows informed strategies of scheduling. For example, in Figure 7, an examination of the lower envelope allows us to recognize the risk of not having enough CPU cores in the time interval [15,45]. An intelligent way to resolve this risk is to postpone the starting time of A and B to some time after C begins.

Conclusions and Future Work

In this paper, we formalized the STNR, a combinatorial structure that allows us to jointly reason about the temporal

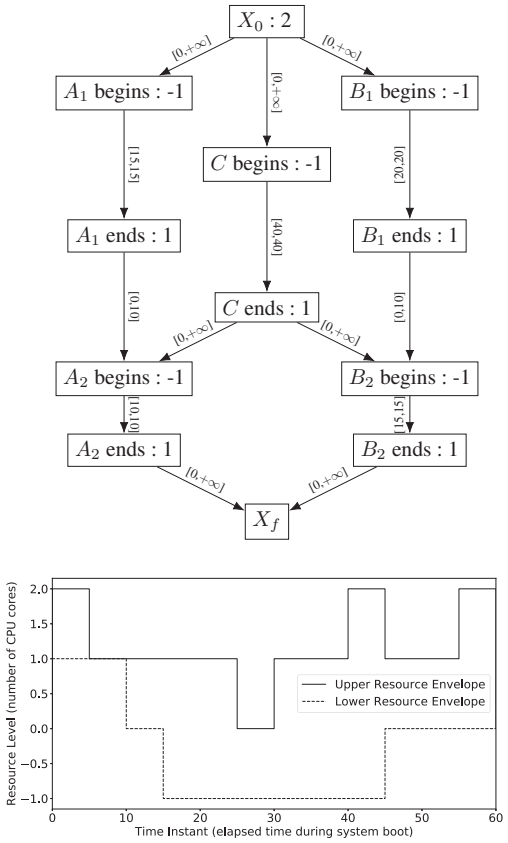


Figure 7: Illustrates an STNR modeling a system boot scenario. In this scenario, services A and B depend on service C. Both A and B require connection to C at some time instants while starting. We refer to the segments of A/B before and after the connection as A_1/B_1 and A_2/B_2, respectively. The beginning of A_2 and B_2 would depend on C. A_1, B_1 and C terminate within fixed time spans. The whole system boot must finish in 60 time units (this temporal constraint between X_0 and X_f is not shown in the figure). The resource is the two CPU cores available at the beginning. The upper panel shows the STNR and the bottom panel shows the corresponding resource envelopes.

aspects of a plan as well as the resource profiles of its individual actions. We then defined the problem of computing the upper and lower resource envelopes for a given STNR. We reviewed the algorithms that compute upper and lower resource envelopes for STNRs. We used the polynomial-time algorithms for computing these resource envelopes for alert generation with look-aheads. This enables early detection of failures in execution monitoring and therefore provides more time for retasking or replanning. We demonstrated the usefulness of resource envelopes in many application domains, including the smart home domain, the food delivery domain, and the service management domain. In future work, we intend to use resource envelopes in interleaved planning and scheduling and in an integrated framework for planning and execution monitoring.

Acknowledgment This material is based upon work supported by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-15-C-0138. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the official views or policies of the Department of Defense or the U.S. Government.

References

Bajaj, K., and Mehendale, S. 2016. Food-delivery start-ups : In search of the core. *Prabandhan: Indian Journal of Management* 9(10):42–53.

Dave, P. 2016. UberEats launches in Los Angeles, aiming to succeed where others have failed. *Los Angeles Times* March 15.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1–3):61–95.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann, 1st edition.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 1st edition.

Ji, M.; He, Y.; and Cheng, T. 2007. Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & Operations Research* 34(6):1764–1770.

Knight, S.; Rabideau, G.; Chien, S.; Engelhardt, B.; and Sherwood, R. 2001. Casper: space exploration through continuous planning. *IEEE Intelligent Systems* 16(5):70–75.

Kumar, T. K. S.; Wang, Z.; Kumar, A.; Rogers, C. M.; and Knoblock, C. A. 2018. Load scheduling of simple temporal networks under dynamic resource pricing. In *the AAAI Conference on Artificial Intelligence*.

Kumar, T. K. S. 2003. Incremental computation of resource-envelopes in producer-consumer models. In *the International Conference on Principles and Practice of Constraint Programming*, 664–678.

Pecora, F., and Cirillo, M. 2009. A constraint-based approach for plan management in intelligent environments. In *the Workshop on Scheduling and Planning Applications at ICAPS*.

Pettersson, O. 2005. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems* 53(2):73–88.

Pigatto, G.; de Camargo Ferraz Machado, J. G.; dos Santos Negreti, A.; and Machado, L. M. 2017. Have you chosen your request? Analysis of online food delivery companies in Brazil. *British Food Journal* 119(3):639–657.

Poettering, L.; Sievers, K.; and Others. 2017. systemd, version 235. https://www.freedesktop.org/wiki/Software/systemd/.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition.

Said, C. 2016. Uber kills ultrafast instant option for UberEats. *San Fransisco Chronicle* October 3.

Sidman, J. 2016. UberEats ditches instant food delivery service. *Washingtonian* September 28.

van den Briel, M.; Scott, P.; and Thibaux, S. 2013. Randomized load control: A simple distributed approach for scheduling smart appliances. In *the International Joint Conference on Artificial Intelligence*, 2915–2922.

van Grove, J. 2016. Uber now delivering food in San Diego. *The San Diego Union-Tribune* June 14.