Real-time Planning and Execution for Industrial Operations

Filip Dvorak
Schlumberger

Abstract

Real-time planning systems in industrial environments are faced with challenges of reasoning about how to find the optimal plan of actions to achieve a goal, dispatch and monitor the execution of actions, act on state information, diagnose failures and learn to adapt to unexpected events. On this poster, we formalize planning and execution architecture that encapsulates hierarchical deliberative planning, reactive planning and failure management within behavior trees, discussing experience from practical deployments of the architecture in the oil and gas industry.

Autonomous Control of Industrial Operations

Includes both long-term (deliberative) and short-term (reactive) planning:
- End-to-end high-level plan optimizes the global cost
- Short-term reactive planning recovers from unexpected states of the system

End-to-end problems are too complex to be captured within a single model
- Decomposition into a hierarchy of models helps to address the complexity

Requirements can often change
- How quickly the system needs to respond at different levels of abstractions of decision making
- Which problems need to be solved at which level of hierarchy

"Happy-path" is only a small portion of development
- Failure modeling and management consumes most of the time when deploying into production
- Models need to be easy to extend, debug and maintain

Representation

Definition 1. We define
- a finite set of objects C,
- a finite set of types $T = \{T_i \subseteq C\}$
- and a set of m n-ary variables $V = \{v(c_i \in T_1, .., c_n \in T_n) | dom(v) = \{1, 0\} \vee dom(v) = \mathbb{Z} \vee dom(v) = \mathbb{R}\}$
- an assignment $a_V = (u_1, .., u_m)$, where $u_i \in dom(v_i)$, s_{a_V} denotes the state of the world and S_V denotes the state space as a set of states produced by all possible assignments to variables V

Definition 2. For a state s_{a_V}, F is a formula grammar $F \rightarrow$
1. $\forall x(F) | \exists x(F)|$, where $x \in T_i$
2. $(F) \odot (F)|$, where $\odot \in \{\vee, \wedge, \Rightarrow, \iff\}$
3. $v_i \odot c|$, where $v_i \in V, c \in \mathbb{R}, \odot \in \{<, >, =\}$
4. $v_i|\neg v_i$, where $v_i \in V, dom(v_i) = \{0, 1\}$

For a formula f accepted by grammar F and a state s_{a_V}, $f(s_{a_V})$ and $f(s)$ denote a formula whose truth value is its evaluation.

Automated Diagnosis

Real-world systems naturally include scenarios where an unexpected event leads to a failure that requires either automated or human-assisted recovery. While planning is mainly concerned with finding a set of actions from the current state of the world to the goal state, we can use model-based diagnosis for understanding and explanation of why we cannot find a plan from the current state. A common approach to model the consistency of a state of a system is to define a set of rules R that encode the atomic pieces of knowledge about the system, then we say that R is consistent with regard to state S if all the rules R are satisfied. If R is inconsistent with regard to state S, we use conflicts and diagnosis to explain why:
- Minimal conflict C ⊆ R is a set of rules such that C is inconsistent with regard to S and any strict subset of C is consistent with regard to S.
- Minimal diagnosis D ⊆ R is a set of rules such that R \ D is consistent with regard to S and for any strict subset D′ ⊂ D, R \ D′ is inconsistent with regard to S.

As an example, we can think about evaluating if Covid-19 restrictions are being followed for an observed patient. Figure 1 demonstrates that we find a minimal conflict, showing us the chain of necessary implications that has been broken, and one of the minimal diagnosis shows that if Covid-19 infected patients did not need to be quarantined, the system would become consistent again.

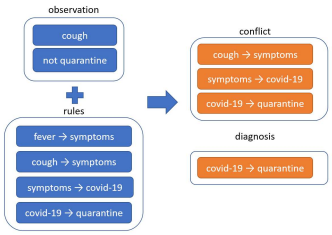

Figure 1: Illustrative example of deducting minimal conflict and an example minimal diagnosis for given observation and a set of rules.

Capturing the Tail of Unexpected Failures

Execution of real-world deployments naturally manifests many failures, which we divide into three categories.
- **Known** failures have been previously identified and we can recover from them automatically,
- **Diagnosable** failures are caught by the model-based diagnosis (hundreds of atomic formulas), human-assisted resolution of the conflicts then leads to either eliminating the failure or it becomes a known failure,
- **Unexpected** failures may require human contribution to understand the failure.

Figure 2 illustrates the continual flow from unexpected to known failures during development.

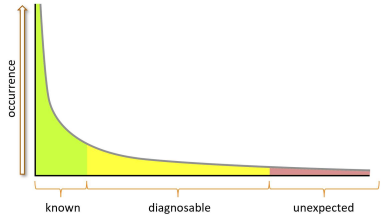

Figure 2

Behavior Trees

Behavior trees are an execution control paradigm, which has been initially used in computer games and became popular in a wide range of robotic applications. Behavior tree is a decision-making tree that always returns either success, failure or running status. A node is a behavior tree if it has a finite number of children that are nodes. We define several nodes:
→, sequence node evaluates to
- success if all its children evaluate to success,
- failure if at least on node evaluates to failure, and
- running if at least one node evaluates to running.

?, evaluates to:
- success if at least one child evaluates to success,
- failures if all children evaluate to failure, and
- running if at least at least one child evaluates to running

Condition(f)
- Succeeds if condition given by formula f holds

Diagnose(f)
- Keeps running until the underlying diagnosis problem is computed
- Succeeds if there are no conflicts

Action(a)
- Keeps running until the action a completes.
- Fails if persistent conditions are violated

Plan(p) & **Execute**(p)
- Keeps running until a plan is found and executed.
- Fails if there is no plan, or execution fails.

Figure 3 shows an example of a construction of a simple behavior tree which captures a model where we configure the system, then diagnose for any failures, if a failure is discovered then it is resolved using a custom recovery strategy, and finally we find a plan and execute it.

Figure 3

Automated Planning

Previous definitions gave us the state of the world, formulas to find structures in the world, capability to identify if the world is inconsistent and how to explain the inconsistency. Now we can start interacting with the world using a deliberative process of choosing and organizing actions by their expected outcomes – planning.

Definition 4. For a state space S we define a temporal numeric planning problem $P = (S, O, s_i, s_g)$, where:
- O is a set of operators $o = (f_c, f_p, \alpha_e, d, p)$, where f_c and f_p are formulas representing preconditions and persistent conditions, α_e represents effects as a partial parameterized assignment, $d : S \rightarrow \mathbb{R}$ is a parameterized duration function and p is a set of type assignments to all free variables (parameters) in f_c, f_p, α_e and d.
- s_i is an initial state of the world.
- s_g is a partially assigned goal state.

For a planning problem P, $\pi = \{a_o(b, p_o)\}$ is a set of actions, where $o \in O$ is an operator, $b \in \mathbb{R}$ is the action start and p_o is an assignment of objects to free variables in f_c, f_p, α_e and d.

π is a plan for the planning problem P iff all the actions can be started at their start times running for their duration, all conditions and persistent conditions are satisfied, all action effects are applied and after applying effects of the latest actions, and we reach a state that satisfies the goal s_g.

When a planner produces a plan of actions to take from the current state, these actions form a behavior tree where each action is represented by a single node. The node's success, failure, or running evaluation, directly reflects success, failure, or current execution of the action it represents—similarly to the given example when finding a key can fail, in which case the whole behavior tree fails.

Deployments – Composition of Control Models

Control Model consists of:
- Projection function that takes snapshot of the world into representation
- Behavior Tree, where:
- Planning nodes are parametrized by PDDL models using representation given by the projection function

Deployment can consist of multiple control models
- Projection functions are providing representation for each model independently
- Behavior Trees of two composed models can compose by:
- Running in parallel
- One completely alternating the other
- One running within the reactive part of the other

Figure 4 shows an example of a behavior tree, where the white blank nodes represent insertion first for alternating behavior of the whole tree and secondly for reactive behavior engaged just before deliberative planning & execution.

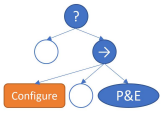

Figure 4

Conclusions

This poster formalizes a hierarchical scalable autonomous control architecture for real-time problems with heterogeneously spread domain knowledge, practically applied in an increasing number of unique field deployments in the oil and gas industry. The architecture interleaves well-known reasoning techniques, discussing the intricacies of working within hard (undecidable) problem spaces, and provides support for independent parallel development of different domains (PDDL modeling, reactive planning, or other custom types of behaviors).

In Schlumberger, AI planning has become an integral part of automation in the DrillOps on-target well delivery solution, providing and executing weeks-long plans for drilling operations and helping to drive new collaborations in the industry. In Wireline, AI planning, automated diagnosis, and behavior trees have been the main building blocks to answer the automation challenges of reasoning and control.

Contact

Filip Dvorak
Schlumberger
Email: fdvorak@slb.com