Advances in Cognitive Systems X (20XX) 1-6 Submitted X/20XX; published X/20XX

Computing Numeric Expectations for Cognitive Agents

Noah Reifsnyder NDR217@LEHIGH.EDU
Hector Munoz-Avila MUNOZ@CSE.LEHIGH.EDU
Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA 18015 USA

Abstract

One of the crucial capabilities for robust agency is self-assessment, namely the capability of the agent to compute its own boundaries. Goal reasoning agents do this by computing so-called expectations: constructs defining the boundaries of their courses of action as a function of the plan, the goals achieved by that plan, the initial state, the action model and the last action executed. In this paper we redefine four forms of expectations from the goal reasoning literature but this time the agent reasons with numeric fluents and we present a comparative study highlighting their trade offs.

1. Introduction

Over the past years there has been an increasing interest in goal reasoning agents; agents that may change their goals over time as a result of changes in the environment and/or changes in the user's requirements (Aha, 2018). One of the main motivations for goal reasoning is the robust intelligence problem, where agents exhibit **self-assessment** capabilities, namely, they are aware of their own boundaries. In this paper we focus on the self-assessment problem of goal reasoning agents. Our work is motivated by the premise that it is infeasible for the agent to plan ahead for every possible contingency that it may encounter when executing a course of action and it is also unfeasible to replan to every possible contingency (Ghallab et al., 2014).

In domains with numeric fluents, it has been observed that it is contrived to assume after performing each action, each fluent will be expected to have an exact value (Scala et al., 2016). For instance when traveling between two locations assuming that a precise amount of gasoline will be consumed by the vehicle. Even under "usual" conditions, factors such as traffic accidents may cause consumption changes. Authors have observed that a robust way to handle numeric fluents is to use intervals or margins of error (Moore et al., 2009). This reduces replanning since the agent can plan accounting for variations. Nevertheless, the agent may encounter conditions under which fluents may take values outside predefined ranges.

To address the necessity for failure detection, agents compute expectations, $X(\pi, s, G)$, as a function of the current state s, the policy π, and the goals G. Expectations are conditions that are checked against the observed environment, $O(s)$, during execution. When the conditions are not met, we consider this a discrepancy that must be addressed. Expectations are needed, because the environments are dynamic and the conditions under which a plan was generated may change at execution time. Numeric fluents can be altered by actions through functions instead of by simple

value assignment (Hoffmann, 2003; Coles et al., 2010). For example, for a *navigate* action, if a variable *fuel(r1)*, has a value of the fuel level of rover *r1*, then the variable could take a new value *fuel(r1) - (travel-time(?y,?z) * rate(r1))*, where $?y$ and $?z$ are locations. The starting state, s_0 includes numeric fluents such as the fuel level of *r1*, travel time between locations, and symbolic fluents such as the starting location of *r1*. Frequently, for planning purposes these functions are assumed to be monotonic (Edelkamp, 2003; Hoffmann, 2003; Bajada et al., 2015) although some paradigms drop this assumption (Scala et al., 2016). Scala et al. (2016) shows an algorithm for heuristic planning with numeric fluents using intervals. The general problem of planning with numeric constraints is undecidable (Helmert, 2002) although under some conditions it is polynomial (Aldinger et al., 2015). Our work doesn't make any assumptions of how the plan π was generated.

The following are the main contributions of the paper: (1) we re-examine a taxonomy that encompasses expectations as computed by goal reasoning systems for symbolic fluents reported in Dannenhauer & Munoz-Avila (2015); Munoz-Avila et al. (2019) and extend it to a taxonomy for numeric fluents and their margins of error. Up until now, the bulk of the research on goal reasoning has focused on domains where actions have symbolic fluents. The few exceptions will be discussed in the related work section. (2) We also analyze the tradeoffs between those expectations. (3) We report experiments on a goal reasoning system with the five forms of expectations and discuss their trade-offs.

2. Preliminary definitions

This section provides intuition into the definitions of the basic terminology we use. For the formal definitions please refer to Appendix A1. A state is a collection of variables that takes interval values.

In this paper we will focus on actions with numeric fluents for simplicity of the exposition. To represent states we use a collection of variables V; a state s indicates the binding of the value $s(v)$ of each variable $v \in V$. For instance, the variable *at-y(r1)* returns the y-coordinate as a confidence interval $[\underline{1.9}, \overline{2.3}]$ for rover *r1*. We use $\underline{x}$ for the lower bound of an interval x and $\overline{x}$ for its upper bound.

As exemplified in Table 2, an operator is a 4-tuple *o=(name parameters precondition effect)*. The parameters are a collection of free variables, which are used to facilitate writing operators.[1] We denote free variables by using "?". For instance *?r* denotes the free variable r (i.e., a rover).

The preconditions are a list of interval constraints. Interval constraints are of the form either *within (v, i)* or *not-within (v, i)*, where v is a state variable and i is an interval. The former checks if in the current state s, the state variable $s(v)$ is within interval i whereas the latter checks if $s(v)$ is not within i (i.e., $s(v)$ may partially overlap or maybe totally disjoint from i). For instance, in the operator shown in Table 2 we are checking if the interval for the variable $fuel(?r) = [\underline{fuel(?r)}, \overline{fuel(?r)}]$ is within the interval left bounded by $\overline{rate(?r)}$ and right bounded by ∞. That is, even if the highest fuel consumption, $\overline{rate(?r)}$, occurs, there is enough fuel to execute this action. If all the constraints within the preconditions of an action a are satisfied by a state s, then we say that a is applicable in s.

1. We call these free variables to distinguish them from the state variables.

Table 1. Planning problem with solution plan

(:Initial State
{fuel : {r1: [10,10]}}
{at-y : {r1: [2, 2], Beacon1: [0,0]}}
{at-x : {r1: [0,0], Beacon1: [2,2]}}
{lit : {Beacon1: [0, 0]}}
{rate : {r1: [.9, 1.1]}}
:Actions
move_north, move_south, move_east,
move_west, light_beacon
:Goals
{lit : {Beacon1: [1, 1]}}
:Plan π
move_north, move_north, move_east,
move_east, light_beacon

Table 2. example operator with numeric fluent (fuel)

(:operator move_north
:parameters ?r
:condition within(fuel(?r), [$\overline{rate(?r)}$, ∞])
:effect
at-y(?r) = [f_1(?r), f_2(?r)],
fuel(?r) = [f_3(?r), f_4(?r)],
$f_1(x) = \underline{\underline{at-y(x)}} - 1$,
$f_2(x) = \overline{\overline{at-y(x)}} - 1$,
$f_3(x) = \underline{fuel(x)} - \overline{rate(x)}$,
$f_4(x) = \overline{\overline{fuel(x)}} - \underline{\underline{rate(x)}}$

The effects indicate changes in value to the state variables. For example, Table 2 shows an example of an operator. It alters the variable fuel$(?r) = [\underline{\text{fuel}(?r)}, \overline{\text{fuel}(?r)}]$ as follows: fuel$(?r) = [\underline{\text{fuel}(?r)} - \overline{\text{rate}(?r)}, \overline{\text{fuel}(?r)} - \underline{\text{rate}(?r)}]$. $move\_north$ also alters value of the variable as follows: at-y$(?r) = (\underline{\text{at-y}(?r)} - 1, \overline{\text{at-y}(?r)} - 1)$. This denotes a change in y-position of the rover of -1 (north). So all together, the action $move\_north$ consumes the interval of rate(?r) amount of fuel while moving -1 in the y-coordinate plane.

The set of goals $\mathcal{G}$ is indicates desired values for variables in V. For example, we may state for the domain in Table 1 that the goals are to turn on *Beacon1* (as represented by the fluent *lit Beacon1*). A plan achieves the goals in $\mathcal{G}$. For example, the plan in Table 1, describes a plan that navigates an agent from its initial position of (0,2) to the position of the beacon at (2,0) and then turns on the beacon.

3. Immediate Expectations with Numeric Values

Immediate Expectations takes ideas from plan monitoring execution literature (see related work discussion). Informally, agents using immediate expectations check that the preconditions of the next action to execute a_{i+1} hold in the observed state s_i (Cox, 2007).

4. Informed Expectations with Numeric Values

Informed Expectations accumulate the effects from all previous actions executed so far in π. We present them informally here. The formal definition is in Appendix A3, which depends on the basic operator $\oplus_S$, defined formally in Appendix A2.

At any point in time, we have executed a prefix of the plan $\pi_{prefix} = a_1 \ldots a_{i-1}$, this has resulted in a sequence of states $s_0\ s_1 \ldots\ s_{i-1}$, where s_0 is the starting state and s_{i-1} is the current

state. At each time, we compute the informed expectations $inf_0, inf_1\ inf_2 \ldots\ inf_{i-1}$. Informed expectations carries out the accumulated effects for each variable $v = [\underline{v}, \overline{v}]$ as it has been transformed over time v, starting from its value in the initial state $[\underline{v^0}, \overline{v^0}]$:

$$[\underline{v^0}, \overline{v^0}], [f_1'(\underline{v^0}), f_2'(\overline{v^0})], \ldots [f_1{}'^{\cdots\prime}(f_1'(\underline{v^0})), f_2'^{\cdots\prime}(f_2'(\overline{v^0}))]$$

where (f_1', f_2') are the effects for v of a_1 and $(f_1{}'^{\cdots\prime}, f_2'^{\cdots\prime})$ are the effects for v for a_{i-1}. Informally, informed expectations compounds functions tuples, so that $inf_{i-1}(v) = [\underline{v^{i-1}}, \overline{v^{i-1}}]$ with $\underline{v^{i-1}} = f_1{}'^{\cdots\prime}(f_1'(\underline{v^0}))$ and $\overline{v^{i-1}} = f_2'^{\cdots\prime}(f_2'(\overline{v^0}))$.

If v is not changed in some action a in π_{prefix}, then we assume that its effects are $f_1(x) = f_2(x) = x$ (i.e., the identity function) and therefore when applying composite functions the values do not change.

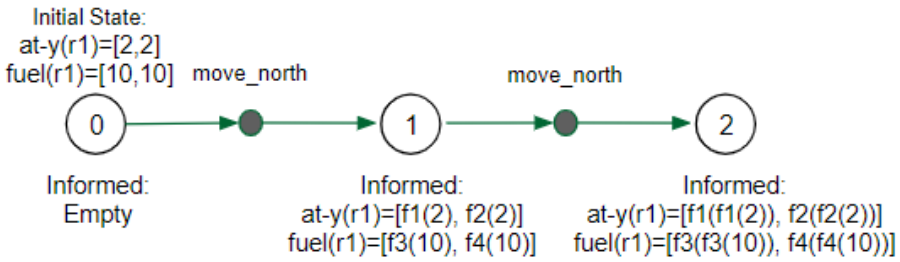

Figure 1. An Example of Informed expectations from the initial state in Table 1 using the operator in Table 2

Example. Figure 1 shows an example. The starting value of the variables at-y$(?r1) = [2, 2]$ and of fuel$(?r1) = [10, 10]$ is propagated forwards by the suffix plan π_{prefix} = move_north move_north resulting in:

- at-y$(?r1) = [f1((f1(2)), f2(f2(2))] = [0, 0]$
- fuel$(?r1) = [f3((f3(10)), f4(f4(10))] = [7.8, 8.2]$.

5. Regression Expectations with Numeric Values

Regression Expectations are an extension of goal regression. Goal regression determines the weakest preconditions needed to execute a plan (Veloso & Carbonell, 1993). Our extension accounts for our representation of the variables as intervals and the manipulation of numeric fluents. We present them informally here. The formal definition is in Appendix A4, which depends on the basic operator $\ominus_P$, defined formally in Appendix A2.

The goals $\mathcal{G}$ are a conjunction of numeric constraints on the variables. Therefore, in regression we carry back conditions on each of these variables v over a suffix of the plan $\pi_{suffix} = a_i \ldots a_n$, where a_n is the last action of the solution plan π and a_i is the next action to execute. Regression over π_{suffix} results in a sequence of regressed conditions $reg_i^v\ reg_{i+1}^v \ldots\ reg_n^v$. The value of reg_n^v

is the condition: $v = [\underline{v_n}, \overline{v_n}]$, where $[\underline{v_n}, \overline{v_n}]$ is the value of v in the final state for all v in $\mathcal{G}$. To compute the value of reg^v_{n-1}, we first copy reg^v_n into reg^v_{n-1}, and then modify it according to the following three cases:

- if v is not mentioned in a_n (that is, its value is not changed in the effects nor it is mentioned in the preconditions), then reg^v_{n-1} remains unchanged (i.e., $reg^v_{n-1} = reg^v_n$).
- if the value of v is changed in the effects of a_n, say $v = [\underline{v_n}, \overline{v_n}] = [f_1(\underline{v_{n-1}}), f_2(\overline{v_{n-1}})]$, then $v = [f_1^{-1}(\underline{v_n}), f_2^{-1}(\overline{v_n})]$ is added as a condition reg^v_{n-1} replacing the previous condition $v = [\underline{v_n}, \overline{v_n}]$.[2]
- Finally, all constraints mentioned within the preconditions of a_n are added to reg^v_{n-1}. If any of the preconditions has a condition requiring a value of v already in reg^v_{n-1}, this replaces any previously assigned value for v.

Each of the previous values reg^v_{n-2}, ..., reg^v_i are computed using the same three cases.

Example. Figure 2 shows an example, the regressed conditions for each state as obtained as follows:

- **State 5**. The goal is for $lit(Beacon1)$ to have a value $[1, 1]$ (i.e., the Beacon 1 should be "on"), hence $reg_5^{lit(Beacon1)}$ has as single condition: $lit(Beacon1) = [1, 1]$.
- **State 4**. The *light_beacon* action requires that $lit(Beacon1) = [0, 0]$ and that the vehicle $r1$ to be in the same location as *Beacon1*, hence, $reg_4^{lit(Beacon1)}$ consists of 3 conditions: *at-x(r1)=[0,0]*, *at-y(r1)=[2,2]* and *lit(Beacon1)=[0,0]*.
- **State 3**. The *move_east* action changes the value of *at-x(r1)* when applied in state 3 using functions f and g. Therefore, at-x$(r1) = [f^{-1}(2), g^{-1}(2))]$ is added as a condition of $reg_3^{lit(Beacon1)}$, replacing its previous value *at-x(r1)=[2,2]* in state 4. The fuel level of $r1$ is added to $reg_3^{lit(Beacon1)}$ because it is a precondition from the *move_east* action. The conditions on the variables *at-y(r1)* and $lit(Beacon1)$ are retained from $reg_4^{lit(Beacon1)}$.

6. Goldilocks Expectations with Numeric Values

Goldilocks Expectations, defined for the symbolic case in Reifsnyder & Munoz-Avila (2018), combine Informed and Regression Expectations. Extrapolating from the symbolic case, they can be computed by calculating informed expectations from the starting state (i.e., in the numeric case by compounding functions $v'^{\cdots\prime} = f'^{\cdots\prime}(\ldots(f'(v))\ldots)$), then regressing off from the final states' informed expectations (i.e., in the numeric case by compounding inverse functions $v = f_{\prime}^{-1}(\ldots(f_{\prime\ldots\prime}^{-1}(v'^{\cdots\prime}))\ldots)$). This would not work in the numeric case since it will regress to the exact same value.

2. f^{-1} denotes the inverse function of f.

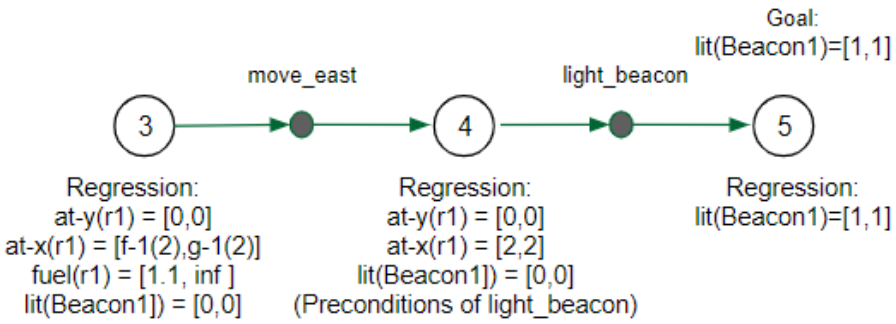

Figure 2. An Example of regression expectations from the goal state in Table 1. $f(x) = left(\text{at-x(x)}) - 1$ and $g(x) = right(\text{at-x(x)}) - 1$

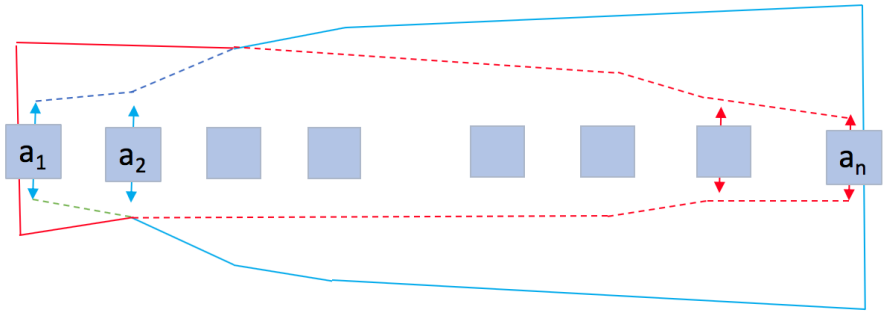

Figure 3. Illustration of Goldilocks expectations. The projected intervals from informed expectations are in blue and those of regression are shown in red. The dashed lines correspond to the Goldilocks expectations

We present a slightly different definition for Goldilocks Expectations tailored towards numeric interval variables; it "intersects" Informed and Regression Expectations by checking that both are met. Figure 3 provides an illustration for a plan $\pi = a_1\ a_2 \ldots a_n$. Function composition tends to enlarge the intervals are they are projected forwards from a_1 and backwards from a_n.

Our motivation for checking both, informed and regression simultaneously, is that Regression can detect when the agent will not achieve its goals, while Informed can detect that something is wrong in the execution of the plan. By looking at them simultaneously, the agent can make decisions over trade-offs between achieving the goals and how the agent is achieving them. For example, using the navigation domain from Table 1, an agent might have a goal to end the plan trace π with a range of fuel left, e.g., $\{fuel : \{r1 : [0, \infty)\}\}$. In this scenario, Regression Expectations will make sure the agent has enough fuel to finish π. While this is important, it is also important to realize that Informed Expectations keep track of accumulated effects of the actions executed so far. Informed is monitoring here the fuel consumption of the agent, and making sure it remains within the bounds as inferred from the action model. If the agent drifts out of those bounds, there may be a flaw with

the agent causing it to consume more fuel than projected. So the agent might still achieve the goals but consume more fuel than expected. Recognizing this expectation failure can allow the agent to trigger a discrepancy and avoid needlessly wasting fuel. By considering both of these expectations, the agent can detect a variety of possible failures at their onset beyond "just" achieving the goals.

Example. When the agent completes action a_3 of the plan trace π in Table 1 (the first instance of $move\_east$), it calculates the Goldilocks Expectations $gold_3$ as follows. $gold_3$ = (inf_3, reg_3), both of which have the following values (for details on how these values were computed see the examples in Appendix A3 and A4):

- $inf_3 = (\{at{-}y : \{r1 : [0,0]\}, at{-}x : \{r1 : [1,1]\}, fuel : \{r1 : [6.7, 7.3]\}\}$
- $reg_3 = \{at{-}y : \{r1 : [0,0]\}, at{-}x : \{r1 : [1,1]\}, fuel : \{r1 : [1.1, \infty)\}, lit : \{Beacon1 : [0,0]\}\})$

There is a difference in the two expectations over the expectations computed for variable $fuel(r1)$. On the Informed side we expect between 6.7 and 7.3 fuel units, but on the Regression side, we just expect to have more than 1.1 fuel units. If we violate the Informed side but not the Regression side, we know we can likely finish the plan, but it will indicate a larger than expected fuel consumption.

7. Empirical Evaluation

In our experiments, we did a comparative study of the 5 different types of expectations across numeric extensions of 2 domains used in the goal reasoning literature. For planning purposes, we use the Pyhop HTN planner Nau (2013), which handles numeric fluents and the HTN methods configured to generate correct plans. Other than the expectation type, the agent uses the same planning and discrepancy handling processes. Whenever a discrepancy is observed from the expectations, we use a simple goal-reasoning process to generate a goal to re-plan from the current state. Thus, any performance changes is attributable to the expectations.

Marsworld Definition. The first domain we used is a numeric variant on the domain Marsworld (Dannenhauer et al., 2016; Dannenhauer & Munoz-Avila, 2015), itself inspired by Mudsworld (Molineaux & Aha, 2014). The agent has to navigate a 10x10 grid to turn on 3 randomly placed beacons. Each movement action drains some amount of the agents fuel. Lighting each beacon also requires fuel, and consumes fuel from a different fuel reserve than from where the agent draws from for movement.

While executing its actions, the agent may unexpectedly have damage caused to it, forcing it to use more fuel per action until repaired (this can occur with a 5% probability after each action is taken). It can also lose some of its beacon fuel with a 5% probability after each action as well. During our testing, we ran 200 trials, each trial placed the rover and beacons randomly on the grid. During the trials we measured total fuel consumption as well as whether or not an execution failed. A failure means the preconditions of some action were not met when it was to be executed.

Results for Marsworld. In Figure 4, we can see that Regression and Goal Regression Expectations consumed the most fuel, with the 3 other expectation types performing basically equally.

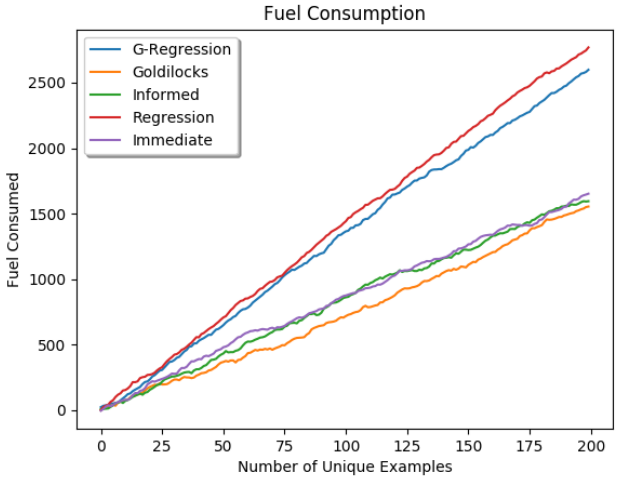

Figure 4. Accumulated Fuel Consumption in Marsworld across different types of expectations. Only Informed failed trails, which occurred 27% of the time

This occurs because Regression and Goal Regression Expectations are the only ones not noticing when the agent is damaged, causing increased fuel consumption. They only look at future preconditions, so they only realize the damage once it drains enough fuel so that the plan can no longer be completed. The other 3 expectation types identify increase consumption after 1 action, since they monitor effects of the actions.

Immediate, Regression, Goal Regression and Goldilocks are able to ensure that the plan will be completed without failures, while 27% of trials failed for Informed Expectations. Informed fails because it will attempt to execute an action without it's preconditions being met. All other expectation types check preconditions. Specifically, in this scenario, agents using Informed expectations will attempt to light a beacon after having lost some beacon fuel, thus failing the action.

Blockscraft Definition. The second domain we tested in is a numeric variant of the Blockscraft domain (Dannenhauer et al., 2016). In our variant, there are three central towers of blocks, colored red, blue and yellow. Each block has a random mass. The mass of each block can only be guessed at planning time but a range for the mass from all blocks is known at planning time. The agent is tasked with collecting all the blocks from a particular color (chosen arbitrarily) and putting them in its own depot, which is limited by the amount of mass it can store. The total mass of the blocks in each individual tower is guaranteed to be less than the maximum capability of the agent's depot. The agent can only access the top block in each tower.

There is another actor in the environment who can also take blocks from the 3 central towers. There is a 8% chance after each action that the other actor will take a block out of the 3 central towers and move it to its own depot, and a 2% chance that the actor will take a block from the 3 central towers and place it in the agent's depot. The agent can also: (1) scan the other actor's depot for blocks and collect them into its own inventory, and (2) scan its own inventory and dispose of any

block. The plans repeatedly pick the top block of the tower with the selected color and store that block into its own depot. So without the actor, the plans are guaranteed to achieve its task.

We ran 200 trials, where the mass of the blocks were randomized each trial. We measured the total mass obtained by the agent during each trial, as well as if the trial failed or not. A failure happens if the agent finishes and the total mass in its depot exceeds its capability, or an action's preconditions are not met when it is to be executed (the only possible occurrence of this is when the agent attempts to take a block from the central towers that is no longer there because the other actor removed it) and the agent doesn't check the preconditions.

Results for Blockscraft. Figure 5 shows the total mass accumulated across 200 runs for an agent using each type of Expectations. Informed and Goldilocks both accumulated roughly the same amount of mass. The reason is because agents using these expectations recognize when the actor adds to the agents collection and thus able to remove the extra block from its depot. By monitoring the effects of actions, the agent knows how much mass is has collected so far. Regression and Informed collected the most, as they are not monitoring the collected mass. Goal Regression monitors the collected mass ensuring that it will not exceed the maximum limit. This is why it resulted in more mass than Goldilocks and Informed while also collecting less than Regression and Immediate. Collecting less total mass is desirable because an increased amount of mass collected would mean an excess amount of un-needed blocks have been collected.

Only an agent using either Goldilocks or Goal Regression Expectations were able to complete the plan and have the goals fulfilled. Regression and Immediate both failed roughly 70% of their trials. These failures occurred when the other actor placed blocks into the agent's collection, eventually exceeding the maximum carry weight. Effects of actions are not monitored for those expectations, so the agent is not monitoring total mass of its collection. Informed failed 29% of its trials. This occurred due to the other actor removing blocks from the central tower that the agent had planned to obtain. The action then to collect that block failed because the block didn't exist in the tower any more. Goldilocks and Goal Regression had a 0% failure rate because they checked preconditions while assuring the goals would be achieved.

8. Trade-offs between Expectations

For the complexity analysis we use n, the length of the trace, and m, the maximum number of preconditions or effects.

Immediate expectations can be computed in $O(1 * m)$ because it only checks the preconditions of one action. It only recognizes failures as they happen, and only of the form where an action can no longer be executed. Still, in simple environments where the agent can expect little to no interference, immediate expectations can be sufficient.

Informed expectations can be computed in $O(n * m)$ time because they traverse the whole trace in the worst case while accumulating the effects. It allows the agent to ensure all accumulated changes the agent made to the environment are tracked. This in turn means that as long as the goals are satisfied due to the agents actions, than they will be satisfied at the end of execution. Secondly, as shown by the experiments above, informed expectations also allows the agent to detect discrepancies in its actions that may not affect its ability to achieve the goals, but may affect other factors such

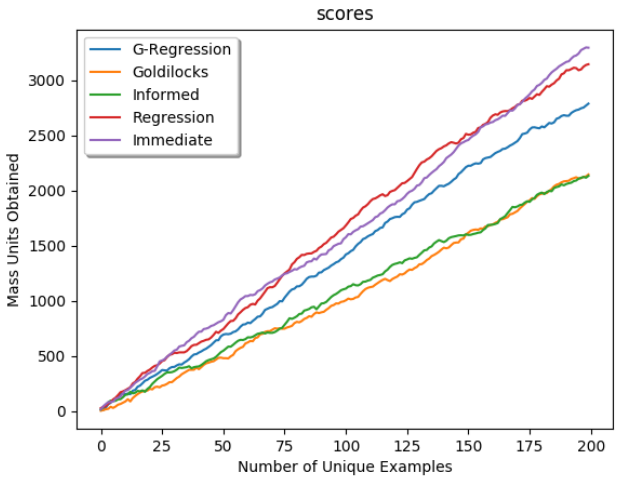

Figure 5. Accumulated Mass Obtained in the Blockscraft Domain across different types of expectations. Regression and Immediate failed 70% of their trials, while Informed failed 29% of the trials. Goldilocks and Goal Regression completed all trials without failing.

as efficiency. For example, as shown in the Marsworld experiments, the agent was able to conserve fuel. However, Informed lacks any ability to reason about the next actions to execute, and this can be shown by the agent frequently collecting more than the allowed mass in the Blockscraft domain.

Regression can be computed in $O(n * m)$ time because it may traverse the whole trace while collecting the preconditions. It allows the agent to certify the rest of the plan is able to be executed. Without a set of goals to regress off of, it cannot certify that the goals will be satisfied upon the completion of the plan's execution. As a result it frequently collects more than the allowed mass.

Goal Regression shares the pros and cons of Regression, with one additional helpful aspect. By starting the regression calculation with a set of goals, we can assure that the rest of the plan is applicable and the goals will be achieved at the termination of execution. This allows it to be sufficient to ensure that the agent finishes it's plan and achieves its goals in all scenarios.

Goldilocks can be computed in $O(n * m)$ because it will perform exactly one pass through the trace, namely, informed expectations up to the action and regression from the end to that action. It also provides the most use to the agent. Goldilocks allows the agent to both know that the rest of the plan can be executed (and that the goals will be achieved upon the termination of execution when using Goal Regression) and identify small discrepancies in the effects of actions that may not impede the ability to achieve the goals but can allow the agent to optimize other metrics such as cost or fuel consumption.

9. Related Work

Scala (2013) proposed using kernel methods to compute the necessary numerical conditions K^i needed to complete the rest of the plan $a_i \ldots a_n$, akin to our regression expectations. While not de-

fined that way, $K^i(v) = f_i^{-1}(\dots(f_n^{-1}(v))\dots)$, where v is a goal condition and f_j^{-1} is the inverse function for v in a_j (with $i \leq j \leq n$). Scala & Torasso (2014) expands this to distances around the values of v playing a similar role as our error intervals; Fritz & McIlraith (2009) assumes Gaussian distribution for this same purpose. Our work differs in some important aspects: by explicitly using inverse functions we provide a concrete way to compute these kernels. More importantly, we introduce two forms of expectations for numeric fluents: informed and Goldilocks. As shown in our experimental evaluation, Goldilocks allows the detection of deviations in the values of numerical values that are pivotal in finishing the provided plan with the expected outcomes, even when the goals are unknown.

We know of two systems using numeric fluents for goals reasoning. Weber et al. (2012) represents quantities as numeric fluents for a goal reasoning agent playing an adversarial real-time computer game. For instance, an action to *produce 10 archers*, will have as an expectation that 10 archers are produced. After executing the action, if the number of archers is num with $num < 10$, then a discrepancy is detected. The agent will formulate a new goal to produce $10 - num$ archers. In the context of the taxonomy we presented, this agent maintains immediate expectations. Furthermore, no margins of error are maintained as the agents expectations are exact natural numbers. Wilson et al. (2014) uses what we call informed expectations. They project forward the expected numerical values within intervals and detect discrepancies when the values are outside of these projected intervals. Our experiments show how informed expectations can incur into the highest number of errors because they don't regress conditions on the goals. This leads to a high volume of failures during execution.

Plan monitoring execution systems annotate the plan with conditions necessary for the plan's execution to be valid (Fikes et al., 1972). While not using "expectations" as a term, Ambros-Ingerson & Steel (1988) checks for the causal links, triples *(effect, fluent, precondition)* are met when the action having the precondition is to be executed. From our definitions, this is subsumed by immediate expectations. Plan monitoring execution has also been used to monitor optimality. For instance, Fritz & McIlraith (2007) uses goal regression to define necessary conditions to guarantee the optimal execution of the plan. These works use symbolic fluents.

10. Conclusions

We re-introduce 5 forms of Expectations over interval numerical fluents: Immediate, Informed, Regression, Goal regression, and Goldilocks. In our empirical evaluation, only agents using either Goldilocks or Goal Regression solved problems without failures across all domains. Goldilocks projects forward all changes the agent makes to the state, as well as making sure the agent is on track to meet the goals. The agent using Goal Regression was shown to be inefficient over the fuel consumption in one of the domains. The reason is that Goal Regression only checks if the agent is on track to satisfy the goals but makes no consideration of any other deviation from the action model.

Time series have been used to build statistical models of "normal" or expected readings for numerical values and in doing so detect outliers to predict malfunctions (Tsay, 1988). In this work, we assume the "normal" ranges has been given in the effects of the actions. However, in future

work, such models could be used in situations when we expect the effects of the actions to change over time.

Acknowledgements

This research was supported by ONR under grants N00014-18-1-2009 and N68335-18-C-4027 and NSF grant 1909879.

References

Aha, D. W. (2018). Goal reasoning: foundations emerging applications and prospects. *AI Magazine*.

Aldinger, J., Mattmüller, R., & Göbelbecker, M. (2015). Complexity of interval relaxed numeric planning. *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)* (pp. 19–31). Springer.

Ambros-Ingerson, J. A., & Steel, S. (1988). Integrating planning, execution and monitoring. *AAAI* (pp. 21–26).

Bajada, J., Fox, M., & Long, D. (2015). Temporal planning with semantic attachment of non-linear monotonic continuous behaviours. *IJCAI* (pp. 1523–1529).

Coles, A. J., Coles, A. I., Fox, M., & Long, D. (2010). Forward-chaining partial-order planning. *Twentieth International Conference on Automated Planning and Scheduling*.

Cox, M. T. (2007). Perpetual self-aware cognitive agents. *AI magazine*, *28*, 32.

Dannenhauer, D., & Munoz-Avila, H. (2015). Raising expectations in gda agents acting in dynamic environments. *IJCAI* (pp. 2241–2247).

Dannenhauer, D., Munoz-Avila, H., & Cox, M. T. (2016). Informed expectations to guide gda agents in partially observable environments. *IJCAI* (pp. 2493–2499).

Edelkamp, S. (2003). Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research*, *20*, 195–238.

Erol, K., Hendler, J., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. *National Conference on Artificial Intelligence (AAAI)* (pp. 1123–1128).

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Some new directions in robot problem solving. *Machine Intelligence*, *7*, 405–430.

Fritz, C., & McIlraith, S. (2009). Computing robust plans in continuous domains. *Nineteenth International Conference on Automated Planning and Scheduling*.

Fritz, C., & McIlraith, S. A. (2007). Monitoring plan optimality during execution. *ICAPS* (pp. 144–151).

Ghallab, M., Nau, D., & Traverso, P. (2014). The actor's view of automated planning and acting: A position paper. *Artificial Intelligence*, *208*, 1–17.

Helmert, M. (2002). Decidability and undecidability results for planning with numerical state variables. *AIPS* (pp. 44–53).

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

Molineaux, M., & Aha, D. W. (2014). Learning unknown event models. *AAAI* (pp. 395–401).

Moore, R. E., Kearfott, R. B., & Cloud, M. J. (2009). *Introduction to interval analysis*, volume 110. Siam.

Munoz-Avila, H., Dannenhauer, D., & Reifsnyder, N. (2019). Is everything going according to plan? - expectations in goal reasoning agents. *Proceedings of AAAI-19*.

Nau, D. (2013). Pyhop, version 1.2.2 a simple htn planning system written in python. `https://bitbucket.org/dananau/pyhop`. Accessed: 2019-01-30.

Reifsnyder, N., & Munoz-Avila, H. (2018). Goal reasoning with goldilocks and regression expectations in nondeterministic domains. *6th Goal Reasoning Workshop at IJCAI/FAIM-2018*.

Scala, E. (2013). Numeric kernel for reasoning about plans involving numeric fluents. *Congress of the Italian Association for Artificial Intelligence* (pp. 263–275). Springer.

Scala, E., Haslum, P., Thiébaux, S., & Ramirez, M. (2016). Interval-based relaxation for general numeric planning. *Proceedings of the Twenty-second European Conference on Artificial Intelligence* (pp. 655–663). IOS Press.

Scala, E., & Torasso, P. (2014). Proactive and reactive reconfiguration for the robust execution of multi modality plans. *ECAI 2014* (pp. 783–788). IOS Press.

Tsay, R. S. (1988). Outliers, level shifts, and variance changes in time series. *Journal of forecasting*, *7*, 1–20.

Veloso, M. M., & Carbonell, J. G. (1993). Toward scaling up machine learning: A case study with derivational analogy in PRODIGY. In S. Minton (Ed.), *Machine learning methods for planning*. San Francisco: Morgan Kaufmann.

Weber, B. G., Mateas, M., & Jhala, A. (2012). Learning from demonstration for goal-driven autonomy. *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Wilson, M. A., McMahon, J., & Aha, D. W. (2014). Bounded expectations for discrepancy detection in goal-driven autonomy. *AI and Robotics: Papers from the AAAI Workshop*.

Appendix

A1. Preliminary Definitions

A state is a mapping $s : V \rightarrow \mathbb{I}$ from a collection of variables V to a collection of intervals $\mathbb{I}$. A state is a mapping $s : V \rightarrow \mathbb{I}$ from a collection of variables V to a collection of intervals $\mathbb{I}$. As in Scala et al. (2016), we represent an interval as $x = [\underline{x}, \overline{x}]$ where $\underline{x}$ is the lower bound on x and $\overline{x}$ is the upper bound (similarly for open and mixed intervals), or if either side is unbounded, that bound is represented as an open interval bound on ∞ (or $-\infty$).

An operator is a 4-tuple *o=(name parameters precondition effect)*, where

- The parameters are a collection of free variables.

- The preconditions are a list of interval constraints. Interval constraints are of the form either *within (v, i)* or *not-within (v, i)*, where v is a state variable and i is an interval. *within* checks if in the current state s, $\underline{s(v)} \geq \underline{i}$ and $\overline{s(v)} \leq \overline{i}$ hold. *not-within* checks if $\overline{s(v)} \leq \underline{i}$ or $\underline{s(v)} \geq \overline{i}$.
- The effects indicate changes in value to the state variables. Effects are represented as function tuples $\mathbb{F} = \{(f_1, f_2) | f_1 \text{ and } f_2 \text{ are functions}\}$ (open and mixed intervals are allowed). The effects can be represented as the partial mapping $eff : V \nrightarrow \mathbb{F}$, where each variable $v \in V_{eff}$ therefore has the function tuple (f_1^v, f_2^v), where f_1^v and f_2^v are the changes to the left and right boundaries to the variable v.
- Effects takes a variable $v \in V$ and assigns v value $[f_1(\underline{v}), f_2(\overline{v})]$, where $(f_1, f_2) \in \mathbb{F}$ and $f_1(x) \leq f_2(x)$ for all x.

The set of goals $\mathcal{G}$ is a mapping $\mathcal{G}$: $V \rightarrow \mathbb{I}$. We say $\mathcal{G}$ is satisfied by a state s if for every variable v_g in $\mathcal{G}$, $within(s(v_g), \mathcal{G}(v_g))$ is true. A plan π is executable if for each action a_i, the preconditions for a_i hold in state s_i and state s_{i+1} is the result from applying a_i to s_i. A plan π is correct if it is executable and transforms s_0 into a state s_{n+1} that satisfies all the goals in $\mathcal{G}$.

A2. Formal Definition of the Two Operators

We define $D = A \oplus_S B$, where A are a collection or variable-interval assignments, S is the current state and B are an action's effects. More generally, for any partial functions $A : V \nrightarrow \mathbb{I}$, $S : V \rightarrow \mathbb{I}$, and $B : V \nrightarrow \mathbb{F}$, $A \oplus_S B$ is a partial mapping $D : V \nrightarrow \mathbb{I}$ defined as follows:

1. if $v \in V_A \cap V_B$ and $B(v) = (f_1, f_2)$, then $D(v) = [f_1(\underline{A(v)}), f_2(\overline{A(v)})]$.
2. if $v \in V_A - V_B$ then $D(v) = A(v)$.
3. if $v \in V_B - V_A$ and $B(v) = (f_1, f_2)$, then $D(v) = [f_1(\underline{S(v)}), f_2(\overline{S(v)})]$.
4. for all other variables D is undefined (i.e., $V_D = V_A \cup V_B$)

Informally, $A \oplus_S B$ applies the function tuple in $B(v)$ either to $A(v)$ when the variable v is defined in A and B (Case 1), or to $S(v)$ when v is defined in B but not A (Case 3). If the variable v is defined in A but not B, it's assigned $A(v)$ (Case 2). When it's undefined in A and B, then it's left undefined (Case 4).

Example. If A, S, and B are defined as:

- $A = \{a : [2, 3]\}$
- $S = \{a : [2, 3], b : \neg[7, 7], c : [8, 9], d : [6, 6]\}$
- $B = \{a : [x - 2, x - 1], b : [x + 1, x + 2], d : [x \times 2, x \times 3]\}$

Then $D = A \oplus_S B = \{a : [0, 2], b : \neg[8, 9], d : [12, 18]\}$. In the resulting partial function $D(a) = [0, 2]$ is obtained by evaluating the functions tuple $B(a) = [x - 2, x - 1]$ on the interval $A(a) = [2, 3]$ which gives $[(x - 2)(2), (x - 1)(3)] = [0, 2]$ (Case 1); $D(b) = \neg[8, 9]$ is obtained

by evaluating the functions tuple $B(b) = [x+1, x+2]$ on the value of $S(b) = \neg[7,7]$ which gives $\neg[(x+1)(7), (x+2)(7)] = \neg[8,9]$ (Case 3); and $D(d) = [12,18]$ is obtained by evaluating the functions tuple $B(d) = [x \times 2, x \times 3]$ on the value of $S(d) = [6,6]$ which gives $[(x \times 2)(6), (x \times 3)(6)] = [12,18]$ (Case 3).

We define $D = A \ominus_P B$, where A are some variables-interval assignments, P are an action's preconditions and B are the action's effects. More generally, let $A : V \nrightarrow \mathbb{I}$, $P : V \nrightarrow \mathbb{I}$, and $B : V \nrightarrow \mathbb{F}$, we define $A \ominus_P B$ as a partial mapping $D : V \nrightarrow \mathbb{I}$ with:

1. if $v \in V_A - V_B$ then $D(v) = A(v)$.
2. if $v \in V_A \cap V_B$ and $B(v) = (f_1, f_2)$, then $D(v) = [f_1^{-1}(\underline{A(v)}), f_2^{-1}(\overline{A(v)})]$.
3. if $v \in V_P$ then $D(v) = P(v)$
4. for all other variables D is undefined (i.e., $V_D = V_A \cup V_P$)

Informally, $A \ominus_P B$ results in a new partial mapping that is defined for all variables from A and P. The new mapping takes the value $A(v)$ if v is defined in A but not in B (Case 1). If a variable v is defined in A and B, the new mapping takes the values after applying the inverse of the functions tuple defined in $B(v)$ to the value of $A(v)$ (Case 2). If a variable v is defined in P but not in A, the new mapping takes the value of $P(v)$ (Case 3). If a variable is not defined in either A or P, it is left undefined (Case 4).

Example. if we have the three partial functions A, P, and B, as follows:

- $A = \{a : \neg[2,3], b : [5,6]\}$
- $P = \{c : [4,4]\}$
- $B = \{b : [x+1, x+2]\}$

Then $D = A \ominus_P B = \{a : \neg[2,3], b : [4,4], c : [4,4]\}$. In the resulting function, $D(a) = \neg[2,3]$ since a is defined in A but not in B (Case 1); $D(b) = [4,4]$ since $A(b) = [5,6]$ and $B(b) = [x+1, x+2]$, hence the inverse functions are $[x-1, x-2]$ so we get $[(x-1)(5), (x-2)(6)] = [4,4]$ (Case 2); and $D(c) = [4,4]$ since c is defined in P (Case 3).

A3. Formal Definition of Informed Expectations

$X_{inf}(\pi, s_i, \emptyset) = inf_i$. Each inf_i is generated as follows: $inf_0 = \emptyset$. That is, before the first action is executed, we have no accumulated effects. For $i > 0$, inf_i is defined recursively as follows: $inf_i = inf_{i-1} \oplus_{s_{i-1}} eff_{s_i}^{a_i}$. Agents using Informed Expectations check that the compounded effects are valid in the environment.

Example. If we have just completed action a_2 of the plan trace π (the second instance of $move\_north$ in Table 1), we calculate the Informed Expectations $X_{inf}(\pi, s_2, \emptyset) = inf_2$ as follows from the initial state. We have:

- $inf_1 = \{at{-}y : \{r1 : [1,1]\}, fuel : \{r1 : [8.9, 9.1]\}\}$
- inf_2 compounds inf_1 with the effects from a_2, the second $move\_north$: $eff_{s_2}^{move\_north} = \{at{-}y : \{r1 : [x-1, x-1]\}, fuel : \{r1 : [x-1.1, x-.9]\}\}$
- Thus, $inf_2 = inf_1 \oplus_{s_1} eff_{s_2}^{move\_north} = \{at{-}y : \{r1 : [0,0]\}, fuel : \{r1 : [7.8, 8.2]\}\}$

For computing *at-y(r1)* we compute $[(x-1)(1), (x-1)(1)] = [0,0]$ and for *fuel(r1)* we compute $[(x-1.1)(8.9), (x-.9)(9.1)] = [7.8, 8.2]$. This expectation set means that we expect to have between 7.8 and 8.2 units of fuel and to be at y=0 on the coordinate frame (any x coordinate is fine)

A4. Formal Definition of Regression Expectations

We distinguish between two types of regression expectations: *goal regression*, when the goals are known and hence $\mathcal{G} \neq \emptyset$, and *regression* when $\mathcal{G} = \emptyset$; an example of the latter is when π is generated by a user but we don't know the goals. Another example is when the plan is generated using HTN planning techniques (Erol et al., 1994). In HTN planning the objective is given by tasks; tasks doesn't need to map to particular atoms to be true in the state. Therefore, unlike goals, they cannot be determined to be true by just examining the atoms in a state.

Formally, $X_{regress}(\pi, s_i, \mathcal{G}) = reg_i$. Each reg_i is generated as follows: $reg_n = \mathcal{G}$. That is, when in the last state, the agent expects $\mathcal{G}$ to hold (when the goals are unknown, $\mathcal{G}$ equals the empty set $\{\}$). For $i < n$, reg_i is defined recursively as follows: $reg_i = reg_{i+1} \ominus_{pre^{a_{i+1}}} eff_{s_{i+1}}^{a_{i+1}}$.

Example. If we have just completed a_3 of the plan trace π in Table 1 (i.e., the first instance of $move\_east$), we calculate the Regression Expectations $reg_3 = X_{regress}(\pi, s_3, \emptyset)$ as follows (the preconditions and effects for $move\_east$ and $light\_beacon$ have not been shown before):

- $reg_5 = \{lit : \{Beacon1 : [1,1]\}\}$, i.e,. the goal.
- $reg_4 = \{at{-}y : \{r1 : [0,0]\}, at{-}x : \{r1 : [2,2], lit : \{Beacon1 : [0,0\}\}$ which are the preconditions for $a_5, light\_beacon$.
- $reg_3 = reg_4 \ominus_{pre^{a_4}} eff_{s_4}^{a_4}$, where:
  - $pre^{a_4} = \{at{-}x : \{r1 : [0,1]\}, fuel : \{r1 : [1.1, \infty)\}\}$
  - $eff_{s_4}^{a_4} = \{at{-}x : \{r1 : [x+1, x+1]\}, fuel : \{r1 : [x-1.1, x-.9]\}\}$
- Thus, $reg_3 = \{at{-}y : \{r1 : [0,0]\}, at{-}x : \{r1 : [1,1]\}, fuel : \{r1 : [1.1, \infty)\}, lit : \{Beacon1 : [0,0\}\}$

reg_4 follows from the preconditions of $light\_beacon$. $reg_3 = reg_4 \ominus_{pre^{move\_east}} eff_{s_4}^{move\_east}$, where a_4 is the second instance of $move\_east$. Since $fuel(r1)$ is in pre^{move_east}, in reg_3: $fuel(r1) = pre^{move\_east}(fuel(r1)) = [1.1, \infty)$. Since $at{-}y(r1)$ and $lit(Beacon1)$ are in reg_4 and not in $eff_{s_4}^{move\_east}$, they carry over from reg_4 into reg_3. $at{-}x(r1)$ is in both, reg_4 and $eff_{s_4}^{move\_east}$, so we compute the inverse functions from $eff_{s_4}^{move\_east}$ to get in reg_3: $at{-}x(r1) = [(x-1)(2), (x-1)(2)] = [1,1]$. Overall, this expectation set means that we expect to have at least 1.1 units of fuel, to be at y=0 and x=1 on the coordinate frame, and for the beacon to not be lit.

A5. Formal Definition of Goldilocks Expectations

We define Goldilocks Expectations as $X_{gold}(\pi, s_i, \mathcal{G}) = gold_i$, where $gold_i = (inf_i, reg_i)$. That is, for every state s_i, $gold_i$ is the pair containing the Informed and Regression Expectations for that state. An agent using $X_{gold}(\pi, s_i, \mathcal{G})$ checks the overlap of the regressed and the informed intervals, $[\underline{inf_i(v)}, \overline{inf_i(v)}] \cap [\underline{reg_i(v)}, \overline{reg_i(v)})]$. This ensures completing the goals while checking for inferred considerations from the action model such as efficiency.