Rationale-based Visual Planning Monitors

Zohreh Alavi and Michael T. Cox
Department of Computer Science and Engineering
Wright State University
Dayton, OH 45435
{alavi.3, michael.cox}@wright.edu

Abstract

In this paper, we introduce a new technique for planning in a world under continuous change. Our approach is to make vision sensitive to relevant changes in the environment that can affect plans and goals. For this purpose, we applied a rationale-based monitor technique to the SHOP hierarchical planner. SHOP generates plan monitors that interact with a vision system and react only to those environmental changes that bear on current planning decisions. Thus when the monitors detect these changes, they execute specific plan transformations as needed. We also present MIDCA_1.3 which uses the extended version of SHOP and the rationale-based monitors. MIDCA communicates with a Baxter humanoid robot to accomplish a goal in a dynamic environment using the monitors to focus vision and adapt plans. An experiment in a blocks world demonstrates the effectiveness of our approach.

1 Introduction

The ability to act and respond to exogenous events in dynamic environments is crucial for robust autonomy. In dynamic environments, external changes may occur that prevent an agent from reaching its goal(s). But the default strategy in most agent systems is to wait until a plan step fails, then repair the plan or start planning anew. Instead, we claim that an intelligent agent should actively watch for what can go wrong and anticipate mistakes before they occur.

Vision has traditionally been a distinct area of research and vision systems act independently of planning and agent behavior. The "goal" of vision is to accept a visual scene as input and to label the objects and perhaps identify the relations between objects as output (see for example [Marr, 1982]). Agent goals are irrelevant. Once an agent architecture receives the output of vision, the system can search for objects or relationships that bear on goals and plans. This division of labor is quite inefficient since many of the objects in a visual scene will likely never affect the agent. In contrast, an active approach to vision asserts that the vision system should operate with the goals and plans as guide (c.f., [Findlay and Gilchrist, 2003; Fermuller and Aloimonos, 1994]). In the research reported here, we propose a novel integration of planning and interpretation that uses goals and plans to bias an active vision component.

We introduce a new system for planning in a world under continuous change in an agent with visual perception. Our main contribution is making vision sensitive to relevant changes in the environment that affect an agent's plans. We apply a rationale-based monitor technique [Veloso *et al.*, 1998] to the SHOP Hierarchical Task Network (HTN) planner [Nau *et al.*, 1999]. Rationale-based monitors provide a means of focusing visual attention on features of the world likely to affect the plan. We modified SHOP to generate plan monitors to interact with a vision system and react only to those environmental changes that bear on current planning decisions. Thus when the monitors detect any relevant changes, corresponding plan transformations are executed as needed. We have added our extended SHOP planner to the planning phase of a cognitive architecture named MIDCA [Cox *et al.*, 2016] refined the integration with a Baxter robot, and tested it on a simulated domain. MIDCA communicates with Baxter to accomplish a goal in a dynamic environment using the monitors to focus vision and adapt plans.

Not only are the plans of the agent important for vision, but the goals themselves are as well. So too should the justification for goal selection be monitored. Let's say a baby is crying. We reason about this anomaly and so generate an explanation such as $hungry(baby) \rightarrow cry(baby)$. As a result, the goal is $\neg hungry(baby)$, and MIDCA starts planning for its achievement by selecting a method to feed the child. Now suddenly the baby stops crying. Thus the reason that led us to choose this goal is no longer valid. The vision system should recognize this and help the planning component to retract the goal altogether. For the purposes of this paper, however, we will limit the scope of our discussion to the means by which plans and their rationale can inform vision.

We begin the paper by describing the MIDCA cognitive architecture and the SHOP planner embedded within it. The following section will then describe the vision monitors concept and how it is implemented in the SHOP planner. An preliminary empirical evaluation of this technique follows with related research and a conclusion finishing the paper.

2 The MIDCA Architecture and the SHOP Planner

The *meta-cognitive, integrated dual-cycle architecture (MIDCA)* [Cox *et al.*, 2016; Paisner *et al.*, 2013] consists of "action-perception" cycles at both the cognitive level and the meta-cognitive level (see Figure 1). In general, a cycle performs problem-solving to achieve its goals and tries to comprehend the resulting actions and those of other agents. The output side of each cycle consists of intention, planning, and action execution, whereas the input side consists of perception, interpretation, and goal evaluation.

In problem solving, the *Intend* component commits to a current goal from those available. The *Plan* component then generates a sequence of Actions (a hierarchical-task-net plan). The plan is executed by the *Act* component to change the actual world through the effects of the planned Actions. The agent stores the goal and plan in memory to provide it with expectations about how the world will change in the future. The agent will then use these expectations in the next cycle to evaluate the execution of the plan and its interaction with the world with respect to the goal.

Comprehension starts with perception of the world in the attentional field via the *Perceive* component. The *Interpret* component takes as input the resulting percepts and the expectations in memory to determine whether the agent is making sufficient progress. The Evaluate component incorporates the concepts inferred from the percepts thereby changing the world model, and the cycle continues. This cycle of problem-solving and action followed by perception and interpretation functions over discrete state and event representations of the environment.

The architecture includes a standard simulator for various planning domains (again see Figure 1). In MIDCA Version 1.3, we have added an API interface to communicate with the Robot Operating System (ROS) and a humanoid Baxter robot. This interface is responsible for sending messages to ROS as requested by MIDCA. Also, the interface places messages received in appropriate queues for MIDCA to process. During the Perceive phase, these messages will be accessed and stored in main memory. The Interpret phase is responsible for reasoning about these messages and creating world states which are represented symbolically as logical predicates.

The Plan phase in MIDCA uses SHOP planner. SHOP is an HTN planning algorithm that creates plans by recursively decomposing tasks into smaller subtasks until only the primitive tasks are left which can be accomplished directly [Nau *et al.*, 1999]. SHOP uses methods and operators. An operator specifies a way to perform a primitive task, and a method specifies a way to decompose a non-primitive task into a set of subtasks.

Following [Ghallab *et al.*, 2004], we denote an action $a = (name(a), precond(a), effects(a))$ accomplishes a primitive task t in a state s if $name(a) = t$ and is applicable to s. A method is a 4-tuple $m = (name(m), task(m), precond(m), subtasks(m))$ in which $name(m)$ is the name of the method; $task(m)$ is a nonprmitive task; and $precond(m)$ is a set of literals called the

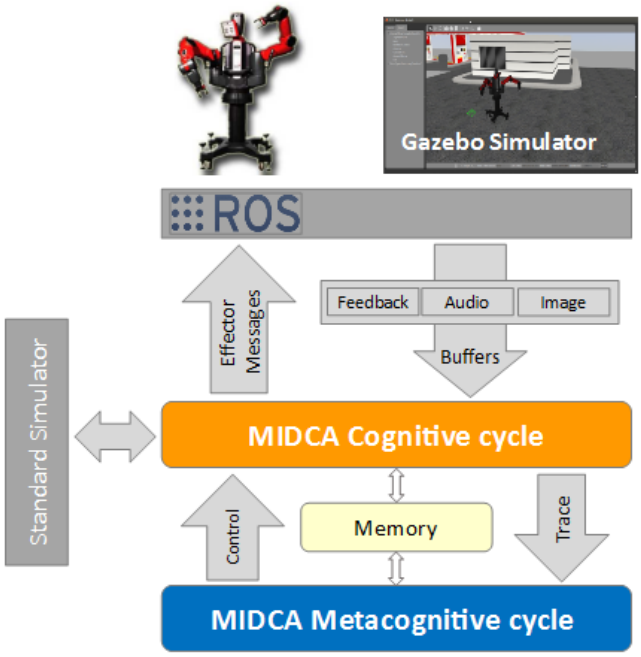

Figure 1: MIDCA_1.3 Interfaces

method's preconditions. $precond(m)$ specifies what conditions the current state must satisfy in order for m to be applied, and $subtasks(m)$ specifies the subtasks to perform in order to accomplish $task(m)$.

Let t be a task and m be a method instance. If there is substitution σ such that $\sigma(t) = task(m)$, then m is relevant for t, and the decomposition of t by m under σ is $\delta(t, m, \sigma) = subtasks(m)$.

An HTN planning problem is a 3-tuple $P = (S, T, D)$. It takes the initial state, S, which is a symbolic representation of the state of world, and a set of tasks, $T = \langle t\_1, ..., t\_k \rangle$, to be accomplished. Also, it takes a knowledge base, D, including operators and methods. A plan $\pi = \langle a_1, ..., a_n \rangle$ is a solution for a planning problem to accomplish T. This means that there is a way to decompose T into π in such a way that π is executable in s, and upon execution will transform the start state into the goal state [Ghallab *et al.*, 2004]. We have modified SHOP to use such plan information to bias vision processes in MIDCA.

3 Rationale-based Vision Monitors

A *rationale-based vision monitor* provides a means of focusing visual attention on features of the world relevant to what the agent is trying to do. That is, vision should be informed by planning activity, because a plan represents the intended actions to achieve the agent's goals and therefore contains the objects and states of interest to the agent. Vision should thus monitor states that form the basis (i.e., rationale) of planning choices. When a feature being monitored changes, and the change is detected, we say that the monitor fires. Deliberation can then be performed to decide whether the plan under construction should be changed. If the planner decides to ac-

count for the new change, it will update the plan and alter the planning search. In particular, parts of the plan may be deleted because they have become unnecessary; new tasks may be added and current ones refined; and prior decisions about how to achieve particular goals may be revisited. Originally, rationale-based monitors were implemented in the state space planner Prodigy [Veloso *et al.*, 1998]. Our work differs in using these monitors in the SHOP planner and applying them to make vision more goal-driven (and less data-driven).

A change in the world can result in different kinds of plan transformations. We organize these transformations into three different categories:

1. Extending the plan with additional actions;
2. Shortening the plan by removing actions;
3. Substituting a different plan with alternative bindings or steps.

This paper focuses on the first two types. See [Veloso *et al.*, 1998] for a discussion of the third type of transformation. We also address monitoring changes in the world that occur at planning time, leaving the extension to plan execution-time monitors for future research.

3.1 The Influence of World State in Planning Decisions

Each of the planning decisions are influenced by the planner's beliefs about the state of world and their goals. Given a task t, the planner repeatedly needs to decide how to decompose the task to achieve t until the decomposition reaches primitive tasks represented as operators, i.e., executable actions. The decision to pursue a decomposition over another one will be highly dependent on the current state of the world. When a satisfied precondition of an operator becomes not satisfied during planning, the planner needs to add steps to the plan to reestablish the condition. In other situations, when a portion of the current plan serves to establish some condition c, it may become necessary to cut those actions from the plan, should c become true.

For example, assume two blocks, A and B as shown in panel a) of Figure 2. The goal is $on(A, B)$, and the task to accomplish this goal is $stack\_T(A, B)$. Because both blocks are clear, the planner generates the plan $\pi = pickup(A), stack(A, B)$. Now consider the situation whereby another agent puts the block C on top of A while MIDCA is planning. This new state violates the precondition of pickup that the item to be grasped be clear. Thus, the planner must add further actions to the plan before continuing. The new plan to achieve this goal will be $\pi' = pickup(C), putdown(C), pickup(A), stack(A, B)$.

3.2 Vision Monitors in SHOP

We have implemented rationale-based monitors within the SHOP planner. Algorithm 1 shows the overall procedure. The SHOP algorithm takes the initial state, s, a set of tasks, $\langle t'_1, ..., t'_c \rangle$, and a knowledge base, D, including operators and methods. We added another argument, l, to the planner to keep track of the recursion tree depth. A plan $\pi = \langle a_1, ...a_m \rangle$ is the solution of this algorithm.

Algorithm 1 SHOP with Rationale-based Monitors

$mnts \leftarrow \langle\rangle$ ▷ list of generated monitors
$l \leftarrow 0$
$T' \leftarrow \langle t'_1, ..., t'_c \rangle$ ▷ goal tasks
procedure $SHOP(s, \langle t_1, ..., t_k \rangle, D, l)$
  $(new\_s, \langle (p_1, l_1), ...(p_n, l_n) \rangle) \leftarrow fired(mnts)$
  if $n \neq 0$ **then** ▷ at least one monitor fired
    $s' \leftarrow backtrack(l_1)$
    $s' \leftarrow$ update s' with new_s ▷ state changed
    return $SHOP(s', T', D, l_1 + 1)$
  end if
  if $k = 0$ **then**
    return $\langle\rangle$ ▷ i.e., the empty plan
  end if
  if t_1 is primitive **then**
    $active \leftarrow \{(a, \sigma) | a$ is an instance of an operator in D, σ is a substitution such that a is relevant for $\sigma(t_1)$, and a is applicable to $s\}$ ▷ active is a set of relevant operators for task t_1
    if $active = \phi$ **then**
      return failure
    end if
    nondeterministically choose any $(a, \sigma) \in active$
    $\pi \leftarrow SHOP(\gamma(s, a), \sigma(\langle t_2, t_3, ..., t_k \rangle), D, l+1)$
    if $\pi =$ failure **then**
      return failure
    else
      $generate\_monitors(a, l, s, mnts)$
      return $a.\pi$
    end if
  else if t_1 is nonprimitive **then**
    $active \leftarrow \{(m, \sigma) | m$ is an instance of a method in D, σ is a substitution such that m is relevant for $\sigma(t_1)$, and m is applicable to $s\}$
    if $active = \phi$ **then**
      return failure
    end if
    nondeterministically choose any $(m, \sigma) \in active$
    $w \leftarrow subtasks(m).\sigma(\langle t_2, ..., t_k \rangle)$
    return $SHOP(s, w, D, l + 1)$
  end if
end procedure

Algorithm 2 Generate monitors

procedure $generate\_monitors(o, l, s, mnts)$
  for p in $precond(o)$ **do**
    if satisfied(p, s) **then**
      $mnts \leftarrow (p, l).mnts$
    else if $\neg$ satisfied(p, s) **then**
      $mnts \leftarrow (\neg p, l).mnts$
    end if
  end for
end procedure

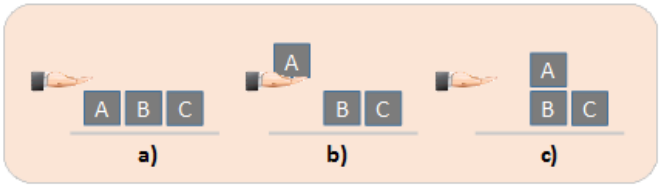

Figure 2: Blocksworld problem to put A on B

The algorithm takes the first task, t_1, and proceeds according to one of the following cases.

Case 1 If t_1 is a primitive task, it nondeterministically chooses an operator, a, from the set of relevant operators for t_1 which is applicable to s. After applying the first action to the state and get the next state, it calls the SHOP planner with the new state and the remaining tasks (steps 14-27).

Case 2 If t_1 is a non-primitive task, then it nondeterministically chooses a method to decompose t_1 into its subtasks and adds them to the list of tasks. Then, it calls SHOP with the current state, and the set of new tasks (steps 27-34).

To integrate with rationale-based monitors, two procedures are added to the SHOP planner. First, the monitors are generated when an operator is added to the current plan, π (line 24 in the algorithm). $generate\_monitors$ takes the operator, o, the current depth, l, state ,s, and the list of monitors ,$mnts$, as input parameters.

Monitors observe features that directly influence π. This includes preconditions of all the operators in π. Algorithm 2 shows the details of monitor generation for the preconditions of an operator. Some of these preconditions will be true when they are added to π; they therefore must be monitored, because, should they become false, π will fail unless additional planning is performed. Other preconditions will be initially false; should they become true, then the portions of π that established them may become unnecessary (steps 3-6 in algorithm 2).

Second, at each planning cycle, the SHOP planner checks for fired monitors. If a monitor fires, the planner goes back to the depth that the monitor was generated to refine the plan. It also updates part of the state based on the perceived state (step 5-9 in algorithm 1)

Algorithm 3 shows the details of checking for fired monitors. It checks to see if the preconditions of all operators in the plan so far are still satisfied in the new perceived state.

Plan transformation is done by backtracking to the depth that the fired monitor was generated. When monitors are generated, the current recursion depth is recorded and backtracking uses this information. Then, the process continues with the new state (the state at the depth that the algorithm backtracked to) and the list of goal tasks (steps 7-9 in algorithm 1).

3.3 A Vision Application

We have added our extended SHOP planner to the planning phase of MIDCA_1.3 and tested our system with a Baxter humanoid robot in a blocksworld domain to show how vision monitors improve planning in a dynamic environment. Additionally, we describe how planning improves the efficiency of vision.

Algorithm 3 Check for fired monitors

```
1: procedure fired(mnts)
2:     s ← perceive the world
3:     for (p, l) in mnts do
4:         if s ⊭ p then
5:             fired_list ← (p, l).fired_list
6:         end if
7:     end for
8:     return (s, fired_list)
9: end procedure
```

In this section, first we describe creating the predicates from the image received from the Baxter' camera, and then we explain how plan monitors restrict the vision. We also examine the relationship between Interpret, Plan and Perceive phases when monitors are running.

Creating a Symbolic World from Visual Images

MIDCA generates world states which are represented symbolically as logical predicates from perceived objects and their description. The inference process operates in two stages. Figure 3 shows these two stages to create the symbolic world from an image. First, as Baxter's camera reads in images, a visual detection node performs a simple object detection procedure to locate the objects and sends object data (e.g., color, location) about any known objects to MIDCA_1.3 (see Figure 4 for Baxter executing plans in a physical blocksworld). The visual detection node is a ROS node that is running concurrently with MIDCA and sends the information about objects to MIDCA. In section 2, we described how our API manages the communication between ROS nodes and MIDCA. The object detection algorithm uses functions from the OpenCV library (opencv.org) to processes the image received, filters color from a certain range in the HSV color space, transforming it in a black and white image. Then it searches for the largest contour on that image, and we assume that this contour will represent the object. After that, the algorithm finds a point in the center of that contour, which will represent the pixel of the object on the image plane.

The Perceive phase in MIDCA receives information concerning detected objects and infers the predicates representing a symbolic world from their xy coordinate locations. For example, if the x values are within a specific threshold, the algorithm will infer that one block is on top of the other one. Based on the y values, it can infer which one is on top of the other one. For example, the predicates that describe the image in Figure 3 are:

$on(green\_block, red\_block)$
$clear(red\_block) = false$
$on - table(red\_block)$
$clear(green\_block) = false$

Planner, Vision and Interpreter

In this section, we discuss how the Planning, Perceive and Interpret phases interact during planning. Perceive and Inter-

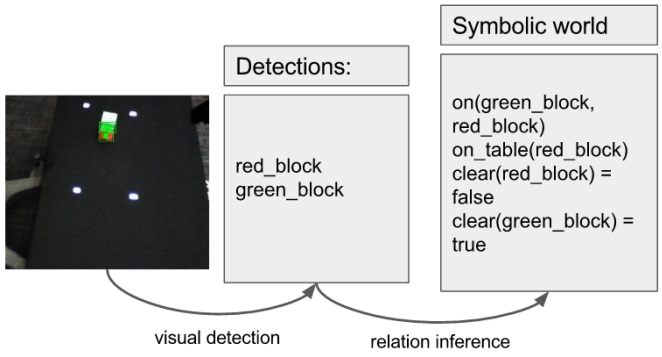

Figure 3: From image to symbolic world

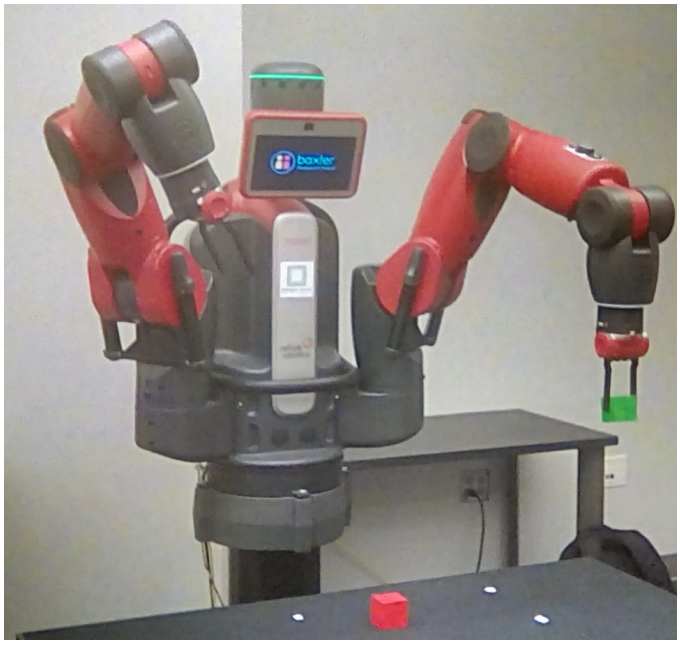

Figure 4: Baxter stacking a green block on a red block

pret Phases are involved in the process of monitoring during the planning.

The planner generates plan monitors to check for related changes in the world state. Every monitor is running asynchronously with the Plan phase. To detect changes in the world, plan monitors parameterize Perceive to seek specific feature changes of the world during planning. In other words, the monitor restricts vision to look for specific relations among the perceived objects (stage 2 in Figure 3). Perceive updates the state in the memory. Then the Interpret phase checks to see if the observed state is the same as the expected state (various approaches to using expectations to detect discrepancies are described in [Vattam *et al.*, 2013; Dannenhauer and Munoz-Avila, 2015]). If any change happens, the interpreter will notify the planner. Then, the planner can refine the plan based on new changes in the world.

Note that in Figure 5 the red block is about to become not clear. A monitor fires and adds steps $(unstack(green\text{-}block), putdown(greenblock))$ to Baxter's plan.

Figure 5: Someone puts a green block on the red block during planning.

Traditionally, vision observes the whole environment and labels all objects, most of which may not be related to any goal. We focus vision on that part of the world that is relevant to successfully accomplishing the goal. Monitors are designed in a way that they guide vision to label parts of the world that have influence on the plan under construction.

4 Empirical Evaluation

MIDCA_1.3 is the latest implementation of the MIDCA architecture which was briefly described in section 2. In this section, we describe our experiment with MIDCA on a modified blocksworld domain [Paisner *et al.*, 2013]. We use a standard simulator to simulate the world state and actions (see Figure 1).

Our work focuses on the relationship between the Perceive and Planning phases. The Perceive phase assists the Plan phase by monitoring the relevant features of the world during planning time. Relevant features are those that relate to the current plan and current goals. In the evaluation below, we show how vision assists planning. We leave the evaluation of planning assisting vision to future work.

4.1 Blocksworld Domain

To evaluate the performance of our approach to the relationship between vision and planning, we ran the system in a modified blocksworld [Fikes and Nilsson, 1971;

Winograd, 1971] domain. The goal of this experiment is to examine the benefit from using vision monitors to improve planning in a dynamic environment.

This version of blocksworld includes both rectangular and triangular blocks, which compose the materials for simplified housing construction. The initial goals for problems in this domain are to build houses consisting of towers of blocks with a roof (triangle) on each. Specifically, the housing domain goes through a cycle of three state classes in building new "houses".

We use a world simulator that simulates actions specified using predicate logic. The types of actions which can be performed are specified prior to startup in a domain file. Actions MIDCA produces during the Act phase will be simulated, as well as actions performed by other agents and natural events.

We added the possibility that blocks could catch fire and before any block was picked up, the fire should first be extinguished. In order for an extinguisher to be used, it must first be taken out of the box. The box itself is represented as a block. If the box is not clear, the planner generates a plan to make the box clear. Furthermore, there were additional actions available to MIDCA allowing it to deal with these refinements. The three new types of actions are as follows:

put-out-fire(C, ext) If C is on fire, extinguish C
preconditions: on-fire(C)=true; holding=[ext]
effects: on-fire(C)=false

get-extinguisher(ext, B) if B is clear, take out the extinguisher ext from B
precondition: clear(B)=true; in-box(ext,B); holding=[]
effects: holding=[ext]

make-box-clear(B) if B is not clear, unstack all blocks on top of B
precondition: clear(B) = false
effects: clear(B) = true

4.2 Experimental Results

In this experiment, we changed the world state in the middle of planning to make the current plan not valid. Our hypothesis is that the planner will refine the plan to successfully achieve the goal.

In each planning problem we set the initial state to be one with a block A on fire, a separate tower with C as its bottom-most block, and a fire extinguisher, ext, inside C. The goal is to put A on B ($on(A, B)$). Figure 6 (a) shows an example for this problem (the height of the tower here is 3). Block A is on fire, $onfire(A)$ is true, and in order to pickup A, the fire needs to be extinguished first. Since the height of tower is 3, the planner has to unstack and putdown 2 blocks, D and E, in order to obtain the fire extinguisher from block C and use it on block A (Figure 6 (b) and (c)).

If the fire goes out during the planning process, the monitor watching the precondition $onfire(A)$ fires. Then the planner cuts parts of the plan related to putout fire and simply generate the plan $pickup(A)$.

Here, the purpose of monitoring is to observe such a change as the fire going out, and suggest a cut in the plan.

By varying the height of the tower, we can vary the complexity and length of the solution. In this experiment, we varied the height of tower, n, from 4 to 26 in increments of 3. During planning, the monitor which observes the state of $onfire(C)$ fires and suggests a plan refinement. We vary the time at which this monitor fires during the planning process, namely after 0, 10, 30, 50 planning steps.

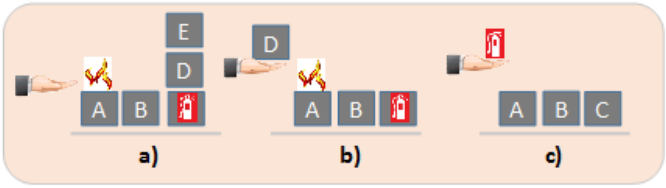

Figure 6: Blocksworld example to put A on B when a fire breaks out

Figure 7 shows the results of the experiment and plots the total time as a function of n. As can be seen, when the environment does not change, the amount of time increases with n. However, with the rationale-based monitors, the planner can react to the state changes and find a solution faster. As would be expected, when the changes occur later, the savings benefit of the planner is reduced, because it has already performed significant planning. When there is no monitor, it has to unstack all the blocks to get the extinguisher. When there is no delay, it means the fire goes out in the beginning and it is like when there is no fire.

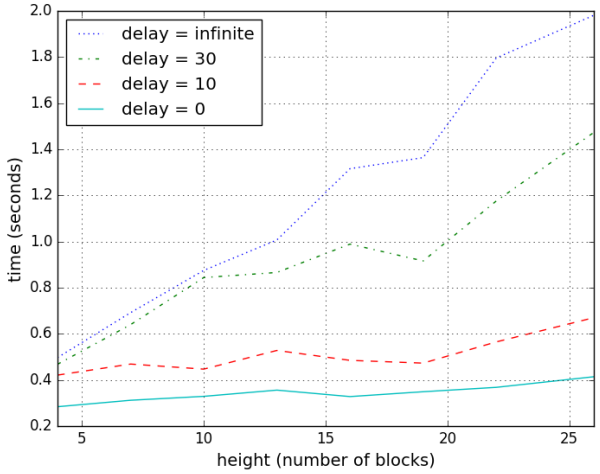

Figure 7: Planning performance using rationale-based monitors in the SHOP planner. The curves refer to different delays of the state change during the planning process.

5 Future Work

There are many promising avenues for future work. First, we plan to examine the benefit of using these monitors during the act (execution) phase. This approach could allow the agent to

respond to unexpected changes during execution (i.e., after planning has already succeeded). This helps the agent to focus only on what is important. In [Ayan *et al.*, 2007], the authors introduce an HTN-based planning system which revises the plan if any action fails due to a state change. Our work is different in a way that we know the action failure earlier, so we have a chance to revise the plan sooner before reaching the failed action.

In our current work, we detect changes during planning time using rationale-based monitors. In doing so, we interleave perception and planning. This suggests that cognitive tasks may benefit from calling other cognitive tasks and/or changing the order in which cognitive processes operate. Exploring this idea further is another avenue of future work. In this modified cognitive architecture, different phases will be able to call each other as needed. Also, if any change happens in the world, MIDCA needs to decide how to respond to those changes. These changes may result in the consideration of new plans or alter the agent's intentions regarding its own reasoning processes. Some changes may cause the system to replan many times, but the better solution might be to change focus completely and pursue new goals.

As mentioned in the introduction, vision should also monitor goals and their rationale. We have used rationale-based monitors in the past to track universally quantified expressions within an action's preconditions [Veloso *et al.*, 1998], but not for vision processing. Furthermore, top-level goals may also be universally quantified and require monitoring. Open-world quantified goals [Talamadupula *et al.*, 2010] represent such an example that could benefit from rationale-based monitors.

6 Conclusion

The integration of planning and interpretation in a cognitive architecture is not a simple one way interaction. Here we have argued that vision should serve the needs of the planner. The planner generates visual monitors for the vision system based on the rationale for plan decisions (e.g., preconditions), and the vision system detects when these conditions are violated. However, it can equally be argued that the planning component should serve the needs of vision and interpretation. Given a particular scene or situation, MIDCA's interpretation component recognizes new problems in terms of expectation failures or discrepancies. The interpretation system will then attempt to explain the discrepancy and use the explanation to generate a goal to remove the problem. The goal is passed to the problem-solving module of MIDCA, and the planner will generate a plan to achieve it. This technique typifies the goal-driven autonomy approach to goal reasoning (e.g., [Aha *et al.*, 2010; Cox, 2013]).

The results in this paper support the idea that vision has an important role in supporting the intentions and actions of the agent. The work is still preliminary in nature however, so much future work remains to be performed.

7 Acknowledgments

ONR supports this research under grant number N00014-15-1-2080. We thank the anonymous reviewers for their comments and suggestions.

References

[Aha *et al.*, 2010] DW Aha, M Klenk, H Munoz-Avila, A Ram, and D Shapiro. Goal-driven autonomy: Notes from the aaai workshop, 2010.

[Ayan *et al.*, 2007] N Fazil Ayan, Ugur Kuter, Fusun Yaman, and Robert P Goldman. Hotride: Hierarchical ordered task replanning in dynamic environments. In *Planning and Plan Execution for Real-World Systems–Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop. Providence, RI*, volume 38, 2007.

[Cox *et al.*, 2016] Michael T Cox, Zohreh Alavi, Dustin Dannenhauer, Vahid Eyorokon, and Hector Munoz-Avila. Midca: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. In *AAAI*, 2016.

[Cox, 2013] Michael T Cox. Question-based problem recognition and goal-driven autonomy. In *Goal Reasoning: Papers from the ACS workshop*, page 10, 2013.

[Dannenhauer and Munoz-Avila, 2015] Dustin Dannenhauer and Hector Munoz-Avila. Raising expectations in gda agents acting in dynamic environments. In *International Joint Conference on Artificial Intelligence (IJCAI-15)*, 2015.

[Fermuller and Aloimonos, 1994] C. Fermuller and Y. Aloimonos. Vision and action. *IVC*, 1994.

[Fikes and Nilsson, 1971] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[Findlay and Gilchrist, 2003] J. M. Findlay and I. D Gilchrist. Active vision: The psychology of looking and seeing, 2003.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Elsevier, 2004.

[Marr, 1982] David Marr. A computational investigation into the human representation and processing of visual information. *Vision*, pages 125–126, 1982.

[Nau *et al.*, 1999] Dana Nau, Yue Cao, Amnon Lotem, and Héctor Muñoz-Avila. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th IJCAI-Vol. 2*, pages 968–973. Morgan Kaufmann, 1999.

[Paisner *et al.*, 2013] Matt Paisner, Michael Maynord, Michael T Cox, and Don Perlis. Goal-driven autonomy in dynamic environments. In *Goal Reasoning: Papers from the ACS Workshop*, page 79. Citeseer, 2013.

[Talamadupula *et al.*, 2010] K. Talamadupula, P. Benton, J.and Schermerhorn, M. Scheutz, and S. Kambhampati. Integrating a closed-world planner with an open-world robot: A case study. In *Proceedings of AAAI 2010*, 2010.

[Vattam *et al.*, 2013] Swaroop Vattam, Matthew Klenk, Matthew Molineaux, and David W Aha. Breadth of approaches to goal reasoning: A research survey. In *Goal Reasoning: Papers from the ACS Workshop*, page 111, 2013.

[Veloso *et al.*, 1998] Manuela M Veloso, Martha E Pollack, and Michael T Cox. Rationale-based monitoring for planning in dynamic environments. In *AIPS*, pages 171–180, 1998.

[Winograd, 1971] T. Winograd. Procedures as a representation for data in a computer program for understanding natural language, 1971.