DOMAIN-GENERAL SIMULATION AND PLANNING WITH PHYSICAL SCHEMAS

Marc S. Atkin
David L. Westbrook
Paul R. Cohen

Experimental Knowledge Systems Laboratory
Department of Computer Science, 140 Governor's Drive
University of Massachusetts, Amherst, MA 01003-4610, USA

ABSTRACT

Physical schemas are representations of simple physically grounded relationships and interactions such as "move," "push," and "contain." We believe they are the conceptual primitives an agent employs to understand its environment. Physical schemas can be used at varying levels of abstraction across a variety of domains. We have designed a domain-general agent simulation and control testbed based on physical schemas. If a domain can be described in physical terms as agents moving and applying force, it can be simulated in this testbed. Furthermore, we show that physical schemas can be viewed as the basis for abstract plans and a domain-general planner, GRASP. Our simulation and planning system is currently being evaluated in a continuous, dynamic, and adversarial domain based on the game of Capture the Flag. The paper concludes with an example of how GRASP was applied to the problem of Course of Action generation and evaluation.

1 PHYSICAL SCHEMAS

One of the big open questions in cognitive psychology is how humans come to conceptualize the world around them. How do we learn, for example, that a stuffed animal in the shape of a cat will typically remain where it was placed, but that a real cat won't? How do we know that liquids can be transported using cups, buckets, or bowls, but not paper bags? How do we know that any of these objects could be used to transport sand, even though we may never have had experience moving sand with a cup?

In a series of papers, Jean Mandler put forward the notion that infants acquire *image schemas*, preconceptual redescriptions of their sensory input (Mandler 1988; Mandler 1992). Image schemas are not much more than pattern detectors or filters, but they form the building blocks from which adult concepts develop. One part of the concept of "cat," for example, is the ANIMATE-MOTION schema, which captures the particular way animals move.

Influenced by Eleanor Rosch's research on categorization, Lakoff and Johnson argue that categories are based less on objective features such as color, size, and shape, than on *interactional* properties and relationships, such as "graspable" and "fits-in-my-mouth," which characterize how an agent interacts with its environment (Johnson 1987; Lakoff and Johnson 1980; Lakoff 1984). At the same time, AI researchers such as Agre (Agre 1988), Chapman (Chapman 1991), and Ballard (Ballard 1989) have argued for *deictic* or agent-centered representations. Lakoff and Johnson make a convincing case that the primitive interactional knowledge acquired by infants becomes more elaborate through abstraction and metaphorical extension as the agent develops (Lakoff and Johnson 1980). The notion of "grasping," for example, which is presumably learned very early in life, is abstracted by adults to the mental realm and then describes the process of understanding an idea. Just as grasping with your hands involves getting a good grip on an object, preventing it from slipping away, and being able to feel its form and texture, grasping an idea involves "getting your mind" around an abstract concept and getting a feel for its structure. Lakoff and Johnson would argue that the notion of grasping an idea is given *meaning* by the more primitive notion of grasping an object.

Motivated by this research, we gave ourselves the task of exploiting the generality and ubiquity of schemas. It occurred to us some time ago that many of the simulators we had been writing really were just variations on a theme. Physical processes, military engagements, and games such as billiards are all about agents moving and applying force to one another (see, for example, (Tzu 1988) and (Karr 1981)). Even the somewhat abstract realm of diplomacy can viewed in these terms: One government might try to *apply pressure* to another for some purpose, or intend to *contain* a crisis before it spreads.

In order to simulate and reason in these domains, we need a set of primitives. We call them *physical schemas*. Like image schemas, they describe basic relationships and interactions between objects. In order to limit their number and solidify their definition, we require them to all be grounded in physics, specifically in the processes of moving and applying force. Examples are MOVE, PUSH, CONTAIN, BLOCK, or SURROUND. If moving an army is conceptually no different than moving a robot, both these processes can be represented with one MOVE action in a simulator. We believe that people think and solve problems in terms of physical schemas. An example: A person notices his sink is leaking, and considers what to about it. He realizes that there are cracks in the sink. The way to solve the problem is to plug the cracks. Now confront this person with a military problem: Hostile forces are moving across a mountain range into friendly territory. What should be done about it? If the person understands that military forces can behave like water, and that the cracks are passes in the mountain range, he can prevent the flow by making the passes untraversable.

2 AFS: THE ABSTRACT FORCE SIMULATOR

To evaluate the feasibility of domain-general schema-based simulation and planning, we have developed a simulator of physical schemas, the Abstract Force Simulator (AFS). It operates with a set of abstract agents called "blobs," which have a small set of physical features, including mass, velocity, friction, radius, attack strength, and so on. Blobs are currently circular, but eventually, blobs will be able to take any shape, and deform and redistribute their mass. A blob is an abstract unit; it could be an army, a soldier, or a political entity. Every blob has a small set of primitive actions it can perform, PRIMITIVE-MOVE, APPLY-FORCE (push), and CHANGE-SHAPE. All other schemas are built from these actions. Simply by changing the physics of the simulator, that is, how mass is affected by collisions, what the friction is for a blob moving over a certain type of surface, etc., we can and we did turn AFS from a simulator of billiard balls into one of unit movements in a military domain.

AFS is a simulator of physical processes. It is tick-based, but the ticks are small enough to accurately model the physical interactions between blobs. Although blobs themselves move continuously in 2D space, for reasons of efficiency, the properties of this space, such as terrain attributes, are represented as a discrete grid of rectangular cells. Such a grid of cells is also used internally to bin spatially proximal blobs, making the time complexity of collision detection and blob sensor modeling no greater than linear in terms of the number of blobs in the simulator. AFS was designed from the outset to be able to simulate large numbers (on the order of hundreds or thousands) of blobs.

The physics of the simulation are presently defined by the following parameters:

- Blob-specific parameters:
  - shape
  - density
  - viscosity and elasticity: determines how blobs interact
  - mass: the blob's effectiveness at applying force
  - position and velocity
  - acceleration
  - friction on different surfaces
  - strength coefficient: a multiplier on mass to compute the force blob can apply
  - resilience coefficient: determines how much mass a blob loses when force is applied to it
- Global parameters:
  - the different types of blobs present in the simulation (such as blobs that need sustenance or blobs than can apply force at a distance)
  - the damage model: how blobs affect each others' masses by moving through each other or applying force
  - sensor model: what information blobs can collect

AFS is an *abstract* simulator; blobs are abstract entities that may or may not have internal structure. AFS allows us to express a blob's internal structure by composing it from smaller blobs, much like an army is composed of smaller organizational units and ultimately individual soldiers. But we don't have to take the internal structure into account when simulating, since at any level of abstraction, every blob is completely characterized by the physical attributes associated with it. Armies can move and apply force just like individual soldiers do. The physics of armies is different than the physics of soldiers, and the time and space scales are different, but the main idea behind AFS is that we can simulate at the "army" level if we so desire—if we believe it is unnecessary or inefficient to simulate in more detail.

Since AFS is simulating physics, the top-level control loop of the simulator is quite straightforward: On each tick, loop over all blobs in the simulator and update

each one based on the forces acting on it. If blobs interact, the physics of the world will specify what form their interaction will take. Then update the blob's low-level sensors, if it has any. Each blob is assumed to have a *state reflector*, a data structure that expresses the current state of the blob's sensory experience. It is the simulator's job to update this data structure.

3 HAC: HIERARCHICAL AGENT CONTROL

HAC (Hierarchical Agent Control) can be viewed as a language for writing agent actions. HAC also forms the basis upon which our planner is built. HAC takes care of the mechanics of executing the code that controls an agent, passing messages between actions, coordinating multiple agents, arbitrating resource conflicts between agents, updating sensor values, and interleaving cognitive processes such as planning.

HAC organizes the agent's actions in a hierarchy (see Figure 1). As one goes up the hierarchy, actions become increasingly abstract and powerful. They solve more difficult problems, such as path planning, and can react to wide range of eventualities. Although actions lower in the hierarchy will tend to be more reactive, whereas those higher up tend to be more deliberative, the transition between them is smooth and completely up to the designer. Unlike other architectures, we do not prescribe a preset number of behavioral levels (Georgeff and Lansky 1987; Cohen, Greenberg, Hart, and Howe 1989).

A hierarchy of sensors parallels the action hierarchy. Just as a more complex action uses simpler ones to accomplish its goal, complex sensors use the values of simpler ones. These are *abstract sensors*. They are not physical, since they do not sense anything directly from the world. They take the output of other sensors and integrate and re-interpret it. Abstract sensors are used throughout HAC and our planner, GRASP, to notify actions and plans of unexpected or unpredictable events.

HAC executes actions by scheduling them on a queue. The queue is sorted by the time at which the action will execute. Actions get taken off the queue and executed until there are no more actions that are scheduled to run at this time step. Actions can reschedule themselves, but in most cases, they will be rescheduled when woken up by messages from their children.

HAC is a *supervenient* architecture (Spector and Hendler 1994). It abides by the principle that higher levels should provide goals and context for the lower levels, and lower levels provide sensory reports and messages to the higher levels ("goals down, knowledge up").

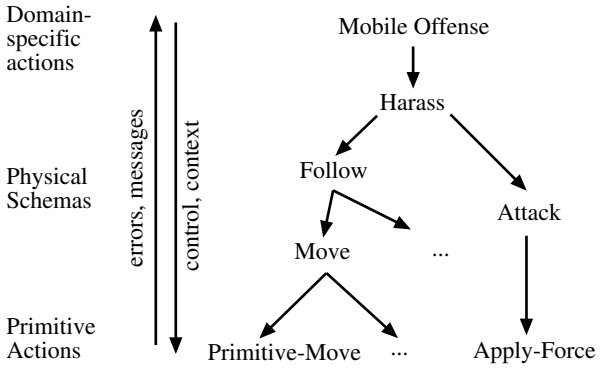

Figure 1: Actions form a hierarchy; control information is passed down, messages are passed up. The lowest level are agent effectors; the middle layer consists of more complex, yet domain-general actions called *physical schemas*. Above this level we have domain-specific actions.

A higher level cannot overrule the sensory information provided by a lower level, nor can a lower level interfere with the control of a higher level. Supervenience structures the abstraction process; it allows us to build modular, reusable actions. HAC simplifies this process further by enforcing that every action's implementation take the following form:

1. React to messages coming in from children.
2. Update state.
3. Schedule new child actions if necessary.
4. Send messages up to parent.

Figure 1 shows a small part of an action hierarchy. The FOLLOW action, for example, relies on a MOVE action to reach a specified location. MOVE will send status reports to FOLLOW if necessary; at the very least a completion message (failure or success). The only responsibility of the FOLLOW action is to issue a new target location if the agent being followed moves. HAC is an architecture; other than enforcing a general form, it does not place any constraints on how actions are implemented. Every action can choose what messages it will respond to. Actions can be deliberative or reactive. Parents can run in a parallel with their children or only when the child completes.

4 THE CAPTURE THE FLAG TESTBED

We have been developing a dynamic and adversarial domain in which to test AFS and HAC. This domain is based on the game of "Capture the Flag" (CtF). In CtF (see Figure 2) there are two teams; each has a

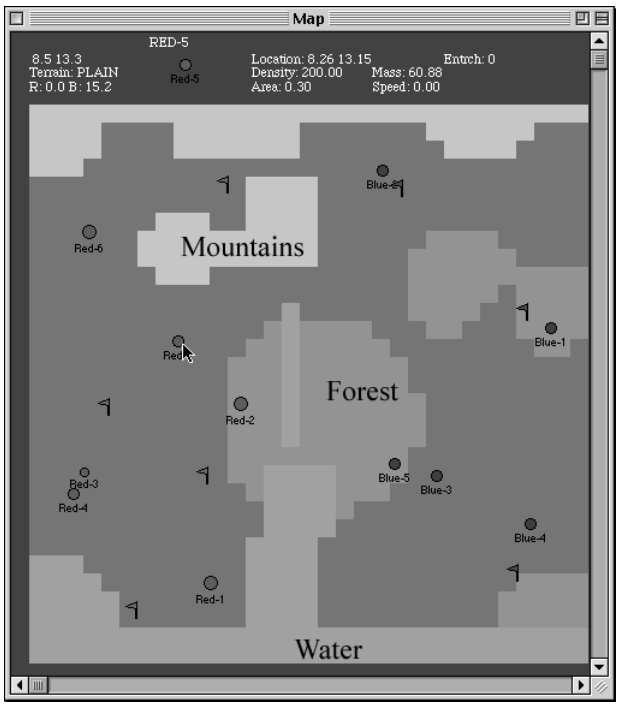

Figure 2: The Capture the Flag domain.

number of movable units and flags to protect. They operate on a map which has different types of terrain. Terrain influences movement speed and forms barriers; terrain also affects unit visibility. A team wins when it captures all its opponent's flags. This game appears deceptively simple. The player must allocate forces for attack and defense, and decide which of the opponent's units or flags he should attack. The player must react to plans that do not unfold as expected, and possibly retreat or regroup. We model limited visibility and inaccurate sensor data. This leads to additional strategies involving feints, sneak attacks, and ambushes.

The following physical schemas are currently implemented in CtF: MOVE, ATTACK, BLOCK, TURN, SPLIT, CLEAR, DAMAGE, DEFEND, DESTROY, HOLD, CONTAIN, FOLLOW, HINDER, BYPASS, BREACH, DISTRACT, SURROUND, and ENVELOP.

5 GRASP: A MULTI-GOAL PARTIAL HIERARCHICAL PLANNER

GRASP (General Reasoning using AbStract Physics) is a least-commitment partial hierarchical planner (Georgeff and Lansky 1986). Partial hierarchical planners rely on a library of pre-compiled plan skeletons which may contain sub-goals and variables that are only bound at plan time. Such a library of plans helps avoid the enormous branching factor a generative planner would face in the CtF domain.

Since most plans post sub-goals during their execution, the set of plans form a hierarchy know as the *goal-plan tree*. This hierarchy meshes well with HAC; in fact, GRASP's skeletal plans are implemented as HAC actions. They differ from simple actions in that they require a "satisfies" clause, a token that allows the planner to match goals to plans, and a "pre-conditions" clause, a function that reduces the branching factor of the planning process by pruning the number of plans that are applicable in a given situation. One way to look at plan skeletons is simply as actions that state the goal they achieve explicitly.

GRASP extends the traditional partial hierarchical planning framework by allowing multiple goals to be associated with a resource or set of resources. These are not simply conjunctive goals; instead, goals are prioritized. GRASP uses heuristics in order to achieve the largest set of high priority goals possible.

In CtF, winning involves coordinating multiple subgoals: protecting your own flags, thwarting enemy offensives, choosing the most vulnerable enemy flag for a counter-attack, and so on. Each requires resources (units) to be accomplished. Sometimes one resource can be used to achieve several tasks. For instance, if two flags are close together, one unit might protect both. Or, advancing towards an opponent's flag might also force the opponent to retreat, thus relieving some pressure on one's own flags.

Every plan must have associated with it a set of functions to assist in the resolution of multiple goals:

- *estimate-resources(plan)*: what resources is this plan likely to need?
- *goal-congruence(planA, planB)*: to what degree do plans A and B achieve the same goal?
- *merge-plans(planA, planB)*: create a new plan that achieves both of plan A and B's goals.

GRASP uses these functions and the algorithm outlined in Figure 3 to solve the resource allocation problem in the presence of multiple goals. Figure 4 shows an example of the plan generation procedure. Each goal is prioritized, then plans are generated to achieve each one. Heuristics are used to generate a small number of possible plan sets. If resource problems arise *during* a plan's execution (because a resource was destroyed and the plan using it cannot succeed without it, for example), a resource error message is sent to the plan initiator using the HAC messaging mechanism, possibly causing resources to be re-assigned or a complete replan to take place.

When several plans apply, partial hierarchical planners typically select one according to heuristic criteria.

An action or a plan posts a set of goals $G = \{g_1, g_2, ...g_n\}$. This invokes the following process:

1. For every g_i:
   - 1.1 Search the list of plans for those that can satisfy g_i.
   - 1.2 Evaluate each potential plan's pre-conditions and keep only those whose pre-conditions match.
   - 1.3 For each remaining plan, estimate its required resources.
2. Sort G by the priority of g_i.
3. *candidate_plan_sets* := nil.
4. Loop over g_i in order of priority:
   - 4.1 If only one plan achieves g_i, instantiate it (bind unbound variables) and add it to every plan set in *candidate_plan_sets*; otherwise:
   - 4.2 If several plans achieve g_i, score each one based on:
     - how many resources it uses
     - how many other goals in G it (partially) satisfies
     - other plan-specific heuristics
   - 4.3 Choose m (m is rarely > 1 to limit combinatorics) of the highest scoring plans: $p_1, ..., p_m$
   - 4.4 Loop over remaining $g_j (j > i)$: if g_i partially satisfies g_j, merge g_j into $p_1, ..., p_m$
   - 4.5 Copy the plan sets in *candidate_plan_sets* m times; add p_k to copy k.
5. Loop over *plan_set* in *candidate_plan_sets*:
   - 5.1 Evaluate *plan_set* using forward simulation.
6. Execute the plan set (make them child actions of the goal poster) that in simulation, results in a world state with the highest score.

Figure 3: The planning algorithm.

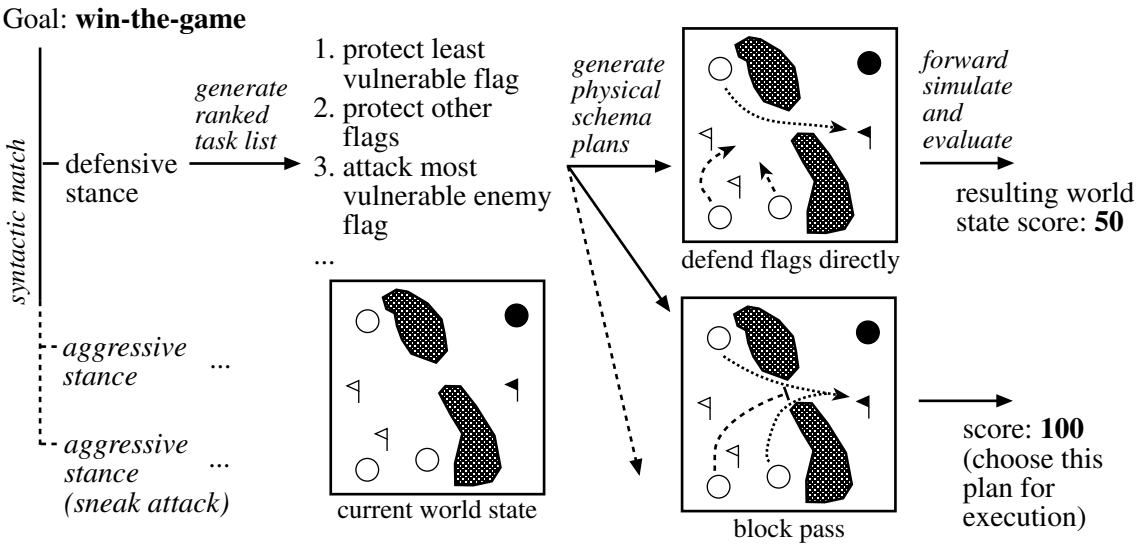

Figure 4: A planning example: White is trying to satisfy the goal **win-the-game**. Several top-level plans match this goal; the example explores what happens when **defensive-stance** is expanded. In the context of this plan, defense is considered vital, which is reflected in the task list that is generated. There are several sets of schemas that achieve these tasks, and many ways to allocate resources to these schemas. The planner uses heuristics to prune this set. In the first case, two blobs are allocated to flag defense, and one is sent out to attack. In the second plan, only one blob is needed to block the mountain pass, thus protecting the flags, leaving two blobs for the attack. This plan is more likely to succeed and is ranked higher.

GRASP instead performs a qualitative simulation on each candidate plan (or plan set), just like a human military planner would. Potential plans are simulated forward, then a static evaluation function is applied to select the best plan. The static evaluation function incorporates such factors as relative strength and the number of captured and threatened flags of both teams to describe how desirable the resulting world state is. Simulation helps alleviate the problem of not being able to specify exact post-conditions for every plan operator. Uncertainty in the world can be addressed by Monte Carlo analysis. The world and your opponent(s) are simply another set of processes to simulate.

The downside is that simulation is a costly operation. In order to do it efficiently and thus be able to evaluate plans quickly, GRASP evaluates plans at a level that is more abstract than the domain being operated in. This is much in keeping with Minsky's original conception of planning (Minsky 1961). GRASP ignores certain details of the domain, such as obstacle avoidance, during plan evaluation. More importantly, GRASP attempts to identify the time periods during which no important interactions between agents are likely to occur and skips over them (Atkin and Cohen 2000).

6 PHYSICAL SCHEMA PLANNING

Physical schemas are simple conceptual structures that describe how objects and agents commonly interact in the physical world. One reason we are interested in physical schemas is because they form a good candidate set for planning operators in a domain-general partial hierarchical planner. The basic idea is that physical schemas are the operators that a general goal-plan hierarchy, based on the manipulation of abstract physical quantities such as mass and position, ground out in.

For example, there are only so many ways to avoid colliding with an object that is approaching you: You can step out of its way, force it to miss by deflecting it, or force it to stop. Viewed at this level of abstraction, there are in turn only a few ways to stop an object: If it is powered, deprive it of its fuel supply; place an obstacle between you and the object; change the environment, for example the terrain, so that it can't maintain its speed.

Plan skeletons in such a domain-general plan hierarchy use only physical schemas to affect change in the world, and achieve goals that ultimately relate to changing or maintaining a physical quantity (for example, "not colliding" means not having one's mass or position changed by another agent). We call these plan skeletons *physical schema plans* because they rely on the set of physical schemas to implement their policies. They

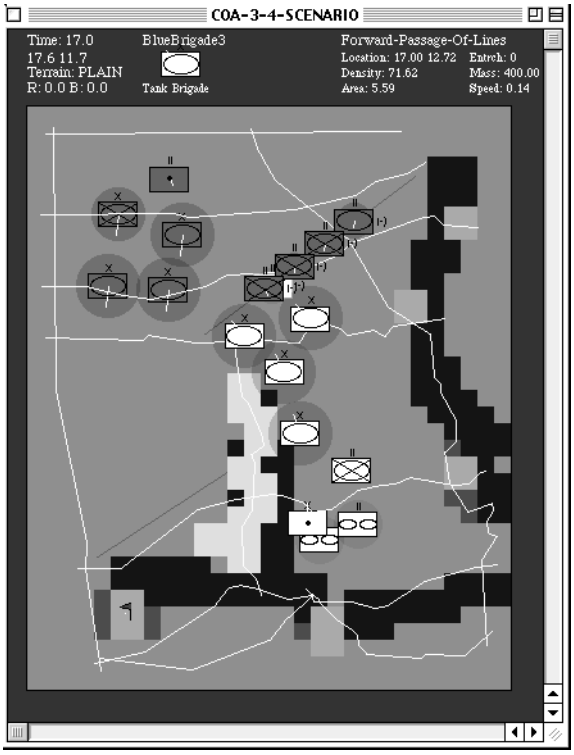

Figure 5: AFS used to simulate a military engagement for the COA analysis experiment.

are distilled knowledge, based on countless interactions with the world, of how to achieve certain desired states in certain physical situations. By the phrase *physical schema planning* we simply mean partial hierarchical planning in a domain of abstract physics, using physical schemas as our operator set.

7 USING GRASP FOR COA ANALYSIS

GRASP and HAC were used to evaluate abstract plans called Courses of Action (COA's) in the High Performance Knowledge Base project (HPKB). We used our simulator to model a military domain and simulated the possible outcomes of a COA using Monte Carlo analysis (see Figure 5). The initial conditions of a scenario were then varied slightly and the effect on the overall outcome of the scenario was measured.

We used GRASP to fill in gaps in a COA that had been sketched by a human planner. An underspecified COA fails to provide actions and goals for all the units being controlled, which can happen when an engagement goes in a direction that was not predicted. Adding the planner serves two purposes: creating more realistic opponents and aiding human planners. A COA must specify both what the friendly units are told to do

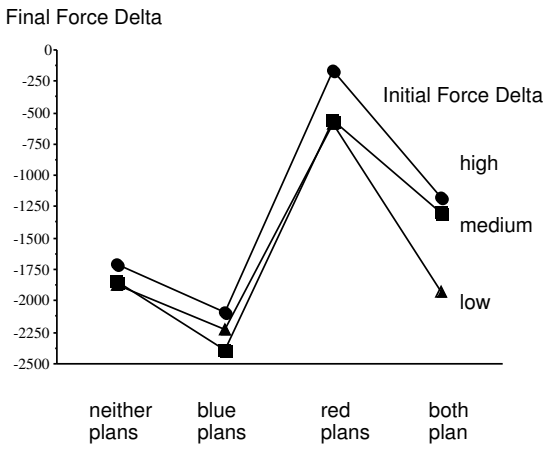

Figure 6: The effect of planning condition and initial force delta on final force delta.

and what the enemy units are likely to do. Rather than being forced to create plans for both sides, it is much more desirable to use the planner to create a reactive and intelligent opponent who can truly test the strength of the COA.

Our goal was to evaluate scripted engagements between Red and Blue forces. We compared four conditions: the COA as specified, the COA plus the Blue planner, the COA plus the Red planner, and the COA plus both Red and Blue planners.

We ran the simulation 100 times, collecting final Red and Blue mass (an abstract measure of unit strength; a brigade has a mass of around 400), the total time of the simulation and the end result. A simulation ended either when one team won or after 300 time units (12 simulated hours). Each simulation randomized the positions and masses of the units on each side. The basic composition of the forces and the tasks specified in the COA were the same in every simulation.

As expected, running either planner alone greatly improves matters for the side that is planning and running both planners moved the averages towards the middle (Figure 6). We found that even in the "no planning" condition, variance in outcome (e.g., final force delta: Red mass minus Blue mass at the end of a trial) was very large, driving home the point that in domains such as this, small differences in the start state can greatly affect the end state. The initial force delta accounts for only 15% of the final force delta. Out of all possible factors, the ability to plan was the best predictor of success.

On the surface this is not a surprising result. It is, however, a qualitative measure of GRASP's ability to produce workable plans in a continuous complex domain under real-time pressure. Furthermore, these plans have the same level of quality that plans designed by human planners have: using the planner, we were able to confirm that one variant of the scenario labeled "Red Most Dangerous" was in fact the most dangerous variant for Blue, and that certain events will lead to Blue's defeat, for example that a Blue counterattack must begin before a certain time.

8 SUMMARY AND RELATED WORK

Physical schemas are domain-general actions and relationships based on the physics of how agents interact. Out of these schemas we have a built a general simulator, AFS. It can be used in any domain that can be described as agents moving and applying force to one another. We have also seen how the physics of a domain naturally prescribes a set of plans, and we have introduced a partial hierarchical planner, GRASP, that takes advantage of these plans. In accordance with our view that physical schemas are able to describe the world at varying levels of abstraction, we simulate at a more abstract level to evaluate plans.

The GRASP planner integrates a number of new and old ideas to deal with continuous and adversarial domains in real-time. It builds upon the established notion of a control hierarchy, used in many agent architectures and hierarchical task network planners (e.g., (Wilkins 1988; Currie and Tate 1991)). GRASP extends the partial hierarchical planning framework by explicitly representing multiple goals and integrating the planner into an action hierarchy that handles resource arbitration and failure recovery. This hierarchy, implemented in HAC, allows us to plan with operators that are flexible and competent. The HPKB COA evaluation experiment provides a qualitative demonstration of GRASP's ability to generate good and timely plans in a realistic application.

The idea of reasoning using procedural knowledge has also been used in a number of other systems, including PRS (Georgeff and Ingrand 1989), PRS-Lite (Myers 1996), RESUN (Carver and Lesser 1993), PHOENIX (Cohen, Greenberg, Hart, and Howe 1989), the data analysis system AIDE (St. Amant 1996), and in languages for reactive control such as RAP (Firby 1987), XFRM (McDermott 1992) and PROPEL (Levinson 1995). The APEX architecture also attempts to manage multiple tasks in complex, uncertain environments, placing particular emphasis on the problem of resolving resources conflicts (Freed 1998).

Although many systems reason about multiple concurrent goals, GRASP is unique among partial hierarchical planners in that it places much of the burden of resolving these goals on the planner, using the availabil-

ity of resources as its primary heuristic. Unlike PRS and RAP, for example, GRASP does not require the designer of actions (tasks) to anticipate every possible event interaction. Plans that react to unforeseen events can be kept conceptually separate from those that are implementing longer term goals.

HAC and GRASP use the same representation for actions at all levels of the hierarchy, and also for plans and sensors. Contrast this with the majority of current agent control architectures, e.g. CYPRESS (Wilkins, Myers, Lowrance, and Wesley 1995) and RAP (Firby 1996), which distinguish between procedural low-level "skills" or "behaviors" and higher level symbolic reasoning. Different systems are often used to implement each level (CYPRESS combines SIPE-2 and PRS, for example).

The evaluation of our claims of domain-generality are still in progress. While AFS has been used in a number of domains with different underlying physics, we have yet to show how easily physical schema plans are transferable from one to domain to another. It is our hope, however, that we can establish a general plan hierarchy for any physical domain based on a general causal model of the physics we are simulating.

ACKNOWLEDGMENTS

We wish to thank General Charley Otstott for his expertise and enthusiastic advocacy, as well as everyone at Alphatech, particularly Eric Jones, for helping run the COA evaluation.

This research is supported by DARPA/USAF under contract numbers N66001-96-C-8504, F30602-97-1-0289, and F30602-95-1-0021. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the Defense Advanced Research Projects Agency/Air Force Materiel Command or the U.S. Government.

REFERENCES

Agre, P. E. (1988). The dynamic structure of everyday life. Technical Report 1085, MIT Artificial Intelligence Laboratory, Cambridge MA.

Atkin, M. S. and P. R. Cohen (2000). Using simulation and critical points to define states in continuous search spaces. To appear in *Proceedings of the 2000 Winter Simulation Conference.*

Ballard, D. (1989). Reference frames for animate vision. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligenc.* Morgan Kaufmann Publishers, Inc.

Carver, N. and V. Lesser (1993, November). A planner for the control of problem solving systems. *IEEE Transactions on Systems, Man, and Cybernetics, special issue on Planning, Scheduling, and Control 23*(6), 1519–1536.

Chapman, D. (1991). *Vision, Instruction and Action.* MIT Press.

Cohen, P. R., M. L. Greenberg, D. M. Hart, and A. E. Howe (1989, Fall). Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine 10*(3), 32–48. also Technical Report, COINS Dept, University of Massachusetts.

Currie, K. and A. Tate (1991). O-Plan: The open planning architecture. *Artificial Intelligence 52*, 49–86.

Firby, R. J. (1987). An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, pp. 202–206.

Firby, R. J. (1996). Modularity issues in reactive planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, pp. 78–85.

Freed, M. (1998). Managing multiple tasks in complex, dynamic environments. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI, pp. 921–927.

Georgeff, M. P. and F. F. Ingrand (1989). Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, pp. 972–978. AAAI Press, Menlo Park, CA.

Georgeff, M. P. and A. L. Lansky (1986). Procedural knowledge. *Proceedings of the IEEE Special Issue on Knowledge Representation 74*(10), 1383–1398.

Georgeff, M. P. and A. L. Lansky (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 677–682. MIT Press.

Johnson, M. (1987). *The Body in the Mind.* University of Chicago Press.

Karr, A. F. (1981). Lanchester attrition processes and theater-level combat models. Technical report, Institute for Defense Analyses, Program Analysis Division, Arlington, VA.

Lakoff, G. (1984). *Women, Fire, and Dangerous Things.* University of Chicago Press.

Lakoff, G. and M. Johnson (1980). *Metaphors We Live By.* University of Chicago Press.

Levinson, R. (1995). A general programming language for unified planning and control. *Artificial Intelligence 76*(1-2), 319–375.

Mandler, J. M. (1988). How to build a baby: On the development of an accessible representational system. *Cognitive Development 3*, 113–136.

Mandler, J. M. (1992). How to build a baby: II. Conceptual primitives. *Psychological Review 99*(4), 587–604.

McDermott, D. (1992, December). Transformational planning of robot behavior. Technical Report YALEU/CSD/RR #941, Yale University, New Haven, CT.

Minsky, M. (1961). Steps towards artificial intelligence. *Proceedings of the I.R.E. 49*, 8–30.

Myers, K. L. (1996). A procedural knowledge approach to task-level control. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, pp. 158–165.

Spector, L. and J. Hendler (1994). The use of supervenience in dynamic-world planning. In K. Hammond (Ed.), *Proceedings of The Second International Conference on Artificial Intelligence Planning Systems*, pp. 158–163.

St. Amant, R. (1996). *A Mixed-Initiative Planning Approach to Exploratory Data Analysis.* Ph. D. thesis, University of Massachusetts, Amherst. Also available as technical report CMPSCI-96-33.

Tzu, S. (1988). *The Art of War.* Shambhala Publications.

Wilkins, D. E. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann.

Wilkins, D. E., K. L. Myers, J. D. Lowrance, and L. P. Wesley (1995). Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI 7*(1), 197–227.

AUTHOR BIOGRAPHIES

MARC S. ATKIN is a graduate student at the Experimental Knowledge Systems Laboratory at the University of Massachusetts. He did his undergraduate work at Karlsruhe University in Germany, and received his Masters in Computer Science from the University of Massachusetts. He is now pursuing a Ph.D. in artificial intelligence at the same institution. His research has focussed on the design of intelligent embedded agents. Currently, his interests include domain-general planning and techniques for efficient control of agents in real-time, continuous, and adversarial domains. His email and web addresses are `<atkin@cs.umass.edu>` and `<eksl.cs.umass.edu/~atkin>`.

DAVID L. WESTBROOK is the Technical Manager of the Experimental Knowledge Systems Laboratory at the University of Massachusetts. He received a B.A. in Computer Science from the State University of New York College at Oswego and an M.S. in Computer Science from the University of Massachusetts. His current interests include planning, simulation, agent design, visualization, and game design. His email and web addresses are `<westy@cs.umass.edu>` and `<eksl.cs.umass.edu/~westy>`.

PAUL R. COHEN received his Ph.D. from Stanford University in 1983. He is a Professor of Computer Science at the University of Massachusetts, and Director of the Experimental Knowledge Systems Laboratory. He edited the "Handbook of Artificial Intelligence," Volumes III and IV with Edward Feigenbaum and Avron Barr. Cohen was elected in 1993 as a Fellow of the American Association for Artificial Intelligence, and served as Councillor of that organization, 1991–1994. His research concerns the design principles for intelligent agents and the acquisition of conceptual structures. His email and web addresses are `<cohen@cs.umass.edu>` and `<eksl.cs.umass.edu/~cohen>`