Physical Planning and Dynamics

Marc S. Atkin and Paul R. Cohen
Experimental Knowledge Systems Laboratory
Department of Computer Science, LGRC, Box 34610
University of Massachusetts, Amherst, MA 01003
{atkin,cohen}@cs.umass.edu

Abstract

The Capture the Flag domain is uncertain, adversarial, and continuous. It poses several hard planning problems. We have developed a planner that attempts to exploit Capture the Flag's inherent dynamics, instead of being stymied by them. In particular, it uses the notion of *critical points* to define states in this continuous domain. These states are then used to efficiently evaluate plans.

Capture the Flag

The Capture the Flag project started when we were asked whether we could build a planner that beats the Army War College professors in war games. We said sure, and the next thing we knew, we were off to Carlisle, PA to meet the professors themselves. Despite mutual interest and admiration, the Army War College declined to participate, so we built our own simulation of a game we call Capture the Flag (CTF). Humans can play against the machine or let the computer play against itself; in the future, we envision CTF also as a vehicle for mixed-initiative planning in which a human/machine pair plays against a human or the machine.

In CTF (see Figure 1), there are two teams; each has a number of movable units and flags to protect. Their number and starting locations are randomized. They operate on a map which has different types of terrain. Terrain influences movement speed and forms barriers. A team wins when it captures all its opponent's flags. A team can also go after its opponent's units to reduce their strength and effectiveness. This game is deceptively simple. The player must allocate forces for attack and defense, and decide which of the opponent's units or flags he should go after. The player must react to plans not unfolding as expected, and possibly retreat or regroup. There are many tactics, from attacking all-out to trying to sneak by the opponent's line of defense. In our current implementation, both players have a global view of the game; when we add limited visibility many more strategies, such as ambushes or traps, will emerge.

CTF is in many ways a direct descendent of Phoenix: a spatial domain, continually changing, in which one plans in real time against adversaries given incomplete and inaccurate data (Cohen *et al.* 1989). Plans will only rarely be carried out without modification; too many unexpected events happen. Both sides must constantly check the execution of their plans, make adjustments, and exploit opportunities. They perform *continual* planning.

CTF also has a passing resemblance to Robocup, as it requires that teams of agents coordinate their actions to achieve a common goal in a (simulated) physical world.

AFS: The Abstract Force Simulator

Capture the Flag is implemented within the framework of a much larger system for agent simulation and control, called the Abstract Force Simulator (or AFS) (further details can be found in (Atkin *et al.* 1998)).

It occurred to us some time ago that many of the simulators we had been writing were just variations on a theme. Physical processes, military engagements, 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 intends to *contain* a crisis before it spreads.

Furthermore, it became clear that there is a common set of terms that can be used to describe all the above domains. Terms like **move**, **push**, **reduce**, **contain**, **block**, or **surround**. Collectively, we refer to these terms as *physical schemas*. 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.

Based on these ideas, we have developed a simulator of physical schemas, the Abstract Force Simulator. It operates with a set of abstract agents, circular objects called "blobs," which have a small set of physical features, including mass, velocity, friction, radius, attack strength, and so on. 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, primarily **move** and **apply-force**. All other schemas are built from these actions. Simply by changing the physics of the simulator, that is, how mass is affected

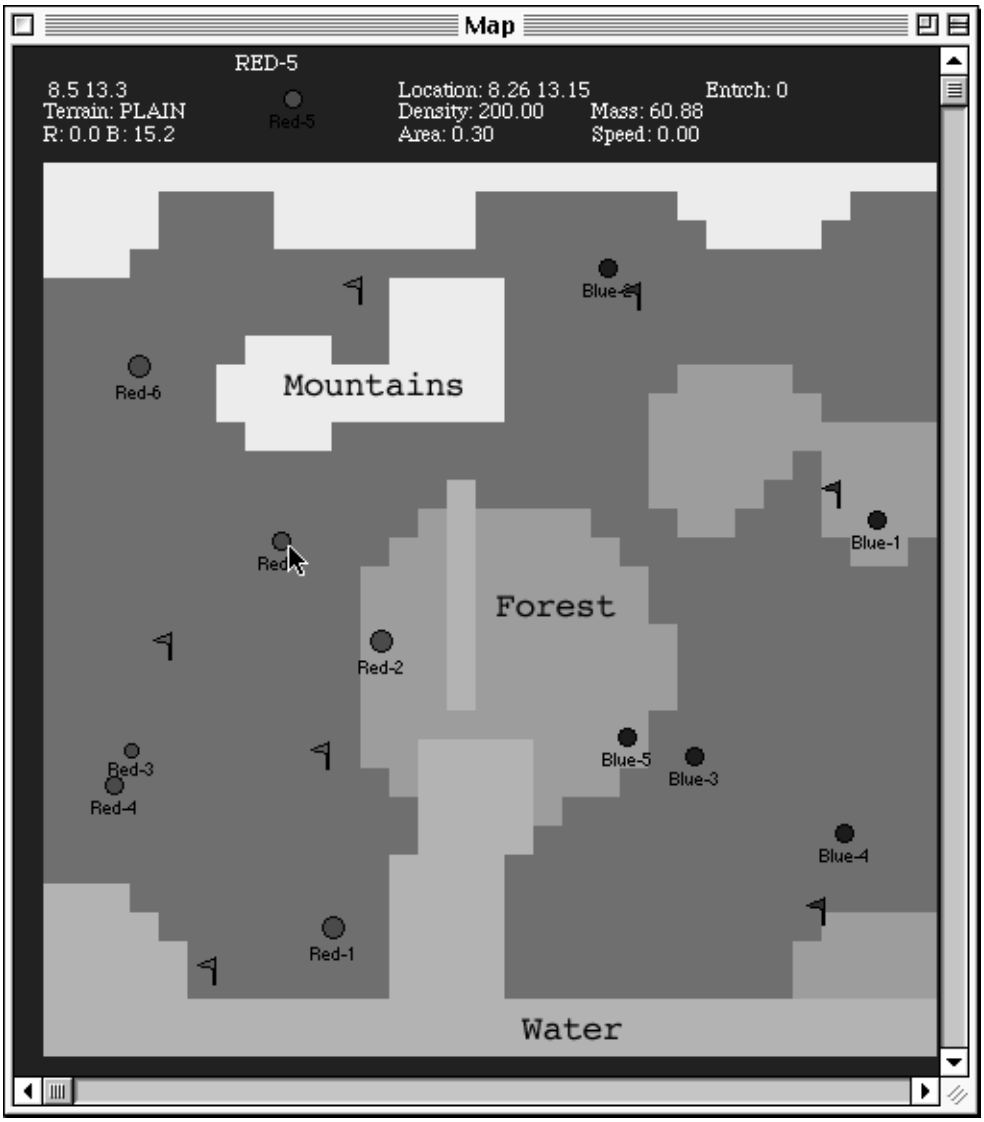

Figure 1: The Capture the Flag domain.

by collisions, what the friction is for a blob moving over a certain type of surface, etc., we can turn AFS from a simulator of billiard balls into one of unit movements in a military domain.

The blob's control architecture is hierarchical. We use the physical primitives **move** and **apply-force** to construct schemas, and schemas to construct domain-specific actions like **convoy** or **sneak-attack**. Our hierarchy is *supervenient* (Spector & Hendler 1994). This means that it abides by the principle that higher levels should provide goals and context for the lower levels, and lower levels provide sensory reports, messages, and errors to the higher levels ("goals down, knowledge up"). 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.

The Capture the Flag Planner

The CTF planner is a partial hierarchical planner (Georgeff & Lansky 1986). By having a set of precompiled skeletal solutions, we can avoid the enormous branching factor a generative planner would face in this domain. Partial hierarchical planning meshes very well with the idea that people understand and reason about the world in terms of physical schemas. Viewed at the level of physical schemas, there are only a few different ways to solve a problem. For example, if A and B are point masses, A can cause B to move by i) pushing it, ii) asking it to move (if it an intentional agent), iii) forcing it to move, or iv) initiating movement in B. These

An action or a plan posts a goal G. This invokes the following process:

1. Search the list of plans for those that can satisfy G.
2. Evaluate each potential plan's pre-conditions (including dynamics), and only keep those whose pre-conditions match.
3. For each plan, do the following:
   - 3.1 If the plan requires multiple tasks to be achieved,
     - 3.1.1 Generate and rank the task list.
     - 3.1.2 Using heuristics, generate a set of schema lists that achieve these tasks efficiently. Each set of schemas is a plan.
   - 3.2 Evaluate the plan (or set of plans) using forward simulation.
4. Execute the plan that in simulation results in a world state with the highest score.

Figure 2: The planning algorithm.

separate solutions can be written down as plans that satisfy the goal "make B move". The exciting thing about planning at the physical schema level is that the plans you use are not limited to just one domain. If you can figure out what "move" and "push" *mean* in a domain, you can use your old plans.

A problem in CTF and any other domain that involves the coordination of multiple agents is resource arbitration. Winning CTF involves multiple tasks: protecting your own flags, thwarting enemy offensives, choosing the most vulnerable enemy flag for a counter-attack, and so on. Each requires resources (blobs) to be accomplished. Sometimes one resource can be used to achieve several tasks. For instance, if two flags are close together, one blob 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.

The CTF planner solves the resource allocation problem by first ranking the list of tasks that need to be achieved to satisfy a goal. It then generates several lists of physical schemas that achieve the most important tasks. Examples for schemas are "attack flag C" or "block the mountain pass." Heuristics, such as "prefer schemas that minimize the total number of blobs needed," are used to keep the task list small. Each schema list constitutes one *plan* for achieving the list of tasks. By design, plans cannot contain resource conflicts, since every pertinent task was considered during their generation. If resource problems arise *during* a plan's execution, for example because a blob was destroyed and the schema using it cannot succeed without it, a resource error message is sent to the plan initiator, possibly causing resources to be re-assigned or a complete replan to take place. The complete planning algorithm is outlined in Figure 2. Figure 3 shows an example of the plan generation procedure.

Plan Evaluation Using Critical Points

When several plans apply, partial hierarchical planners typically select one according to heuristic criteria. Military planners will actually play out a plan and how the opponent might react to it. A wargame is a qualitative simulation. The CTF planner does the same: it simulates potential plans at some abstract level, then applies a static evaluation function to select the best plan. The static evaluation function incorporates such factors as relative strength and number of captured and threatened flags of both teams, to describe how desirable this future world state is.

Simulation is a costly operation, and in order to do it efficiently, CTF must be able to jump ahead to times when interesting events take place in the world. The problem that CTF faces is having to impose "states" on a continuous domain. It does this by defining state boundaries, called *critical points*, that are established dynamically, as the plan simulation unfolds. A critical point is a time during the execution of an action or plan where a decision might be made. If this decision can be made at any time during an interval, it is the *latest* such time.

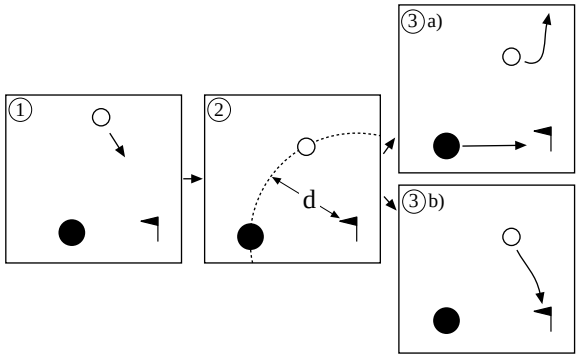

Figure 4: An example for a critical point while executing an attack action.

Simple actions, such as moving from point A to point B, only have one critical point: the time at which the action completes. This is the time at which a new decision has to be made about what to do with the blob that was moving. The critical time can easily be estimated given the terrain and the blob's typical movement speed. More complicated actions have larger crit-

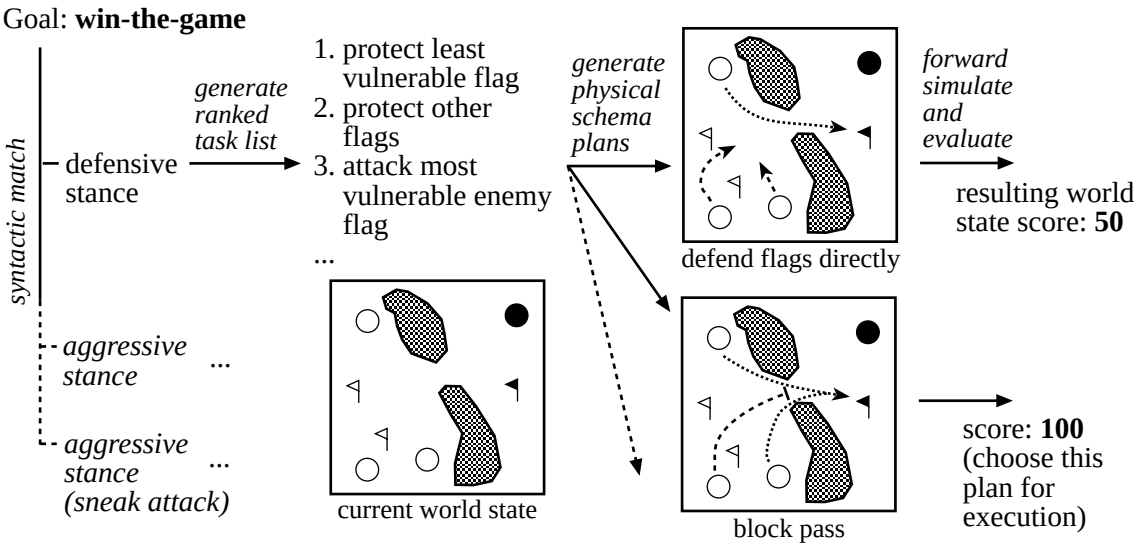

Figure 3: A planning example: White is trying to satisfy the goal **win-the-game**. Several top-level plans match this goal; the example explores what happens when **defensive-stance** is expanded. In the context of this plan, defense is considered vital, which is reflected in the task list that is generated. 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.

ical point sets. This attack action depicted in Figure 4 makes a *decision* during its execution: it will abandon the attack if the thing being attacked is protected by a blob larger than the attacker. In this example, a white blob is attacking a black flag and there is a large black blob nearby. The critical point is the time at which the white blob is closer to the flag than the black blob is now. This is the latest point in time at which Black could interfere with the attack action. If Black has started moving to the flag by this time, White will abandon the attack. If Black has remained stationary or gone somewhere else, the attack will be successful and the flag will be destroyed.

Note that critical points are only bounds, they are not the exact times at which a decision will be made. In the above example, the black blob might move to protect its flag right away, in which case White will abandon the attack sooner than the critical time. This is not a large qualitative difference to the scenario that was simulated. If we had simulated without critical points, and simply completed White's action, there *would* have been a large qualitative error: The flag would have either marked as destroyed (which wouldn't have happened if Black moved in), or White would have been destroyed by the protecting black force (which would never have happened since White would have fled before it came to that).

Critical points are essential for plan evaluation in the CTF planner, since they are used to guide forward simulation. The basic idea behind forward simulation is that instead of advancing the world tick by tick, which is time-consuming, we jump right to the next critical point. Forward simulation proceeds in the following way:

1. Add the plan P to be evaluated to all the actions currently ongoing in the simulator.
2. In simulation, loop either until a fixed time in the future or until too many errors have accumulated in the simulation:
   2.1 Compute the minimum critical time t of all actions being simulated.
   2.2 Advance all actions by t time units.
3. Evaluate the resulting world state; return this value as the score for the plan P.

For this algorithm to work, every action and plan must have two functions associated with it. The first computes the next critical time for this action. The second, **(advance** t**)**, takes as an argument a time parameter t and will change the world state to reflect the execution of this action t time units into the future. Currently, we have no automated way of generating these functions, so they are written by the designer of the action. In the case of the attack action shown in Figure 4, the decision about whether or not to abort depends on whether the white blob can get closer to the black flag than any black blob. A simple approximation of the critical point would be the time it will most likely take, given the terrain, to get as close to

the black flag as the closest black blob is now. A more accurate approximation would take the current velocities of all black blobs into account, since some might be moving towards the flag.

Forward simulation advances to the minimum of all critical points. To understand why this is necessary, let us consider action T, the action with the minimum critical time t. The decision that T must make at time t depends on the state of the world at that time. The state of the world is affected by all the other actions that are executing. If the world had not been advanced by the minimum of all critical times, T might not make the same decision in simulation as it would have if it had actually executed. The downside of having to take the minimum is that forward simulation will take shorter and shorter jumps as the number of simulated actions increases. This makes sense, though, since the more actions you have, the more possible interactions there might be. (One way to alleviate this problem is to prune the number of actions by eliminating those that will most likely have no effect on the action being evaluated.) The upside is that no action has to concern itself with the critical point computation of any other action.

Another upside is that if a critical time is very hard to compute, it is acceptable for it to be underestimated. This will cause the simulated world state to advance to a time sooner than the actual critical time, at which point the action will have another chance to estimate it correctly. In the extreme case, an action can report the smallest time increment possible. The evaluation process for this action will degenerate into normal tick-based simulation.

One last issue to address is how the opponent is handled. If we had an opponent model, it would specify how he would react in certain situations. In the absence of such a model, we simply assume he would do what we would do in his situation. During forward simulation, the action list also contains opponent actions. When CTF starts plan evaluation, it simply puts the top-level goal **win-the-game** for the opponent into the action list. The opponent action's critical times are computed just like ours, and they are advanced in the same way. Whereas our side evaluates all plans and chooses the best one, the opponent chooses the worst one (for us). This really is a form a minimax search, with the two sides executing their plans in parallel.

Towards Dynamics-Based Planning

In the last section we defined states in a continuous domain in order to do efficient plan evaluation. These states were defined dynamically using critical points. The question arises whether it is possible to do planning without "state" at all, purely in the realm of dynamics.

The traditional view in planning is that the world is described by a series of static snapshots, and that planning means finding actions that will move the world from one snapshot (or state) to the next. But this view does not exploit the dynamics inherent in the actions, and instead places artificial state boundaries between them. Consider the process of two battalions engaging. When does this "state" begin and end? Certainly one can identify qualitatively different states in a continuous space; for example, the state of having a resource continues until the resource is exhausted. But planning with such states ignores the dynamics of getting from one to another. Is the resource being used fast or slowly, continuously or erratically? Can you predict when it will be gone, and with what accuracy? So much information about a situation is conveyed by dynamics. Indeed some "state descriptions" are really descriptions of dynamics (e.g., "a pick and roll is in progress"). So if you want to plan for continually changing environments, why start by dividing the environment into states? We are trying to do away with state space search, substituting instead representations and methods from nonlinear dynamics, and reasoning directly about them.

In related work (not CTF) we have developed a representation for dynamics that is based on phase portraits (Rosenstein & Cohen 1998). In phase portraits, one typically plots a value such as position against its derivative, velocity; but the axes could in principle be any variable. We call these generalized phase portraits *dynamic maps*. Actions are trajectories through these maps. One of our more interesting results is that clusters of trajectories correspond to classes of qualitatively different situations in the world, and that these classes can be learned in an *unsupervised* manner. Furthermore, we can classify an unfolding situation very quickly and accurately using dynamic maps.

Dynamic maps could be used in the CTF planner in a number of ways. For one, we could replace the plan pre-conditions, which are currently decision rules written in LISP, by dynamic maps. A plan to besiege the opponent and win by attrition will only work if he is using resources faster than you are. If the map axes are the resource consumption rates of the two teams, being in a situation where this plan applies corresponds to being in a particular region of the map.

Similarly, dynamic maps can represent a plan as it unfolds over time. Clusters of trajectories through the map, corresponding to different outcomes of the plan, can be identified. If, during plan execution, a plan moves too close to a part of the map that will result in an undesirable outcome, it can signal a problem or try to correct its trajectory.

Taking this idea one step further, it might be possible (although we have not yet done so) to write a planner that is guided solely by dynamic maps. A goal is a region of the map you want to reach, planning involves searching over actions that will move the planner towards this region. If the map has regions of attraction, the process is simplified: The planner need not plan to reach the goal, only to reach the goal's attraction region.

Acknowledgments

This research is supported by DARPA/USAF under contracts F30602-97-1-0289 and F30602-95-1-0021. The US 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, Rome Laboratory or the US Government.

References

Atkin, M. S.; Westbrook, D. L.; Cohen, P. R.; and Jorstad, G. D. 1998. AFS and HAC: Domain-general agent simulation and control. In *Working Notes of the AAAI 98 Workshop on Software Tools for Developing Agents*, 89–95.

Cohen, P. R.; Greenberg, M. L.; Hart, D. M.; and Howe, A. E. 1989. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine* 10(3):32–48.

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

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

Rosenstein, M., and Cohen, P. R. 1998. Concepts from time series. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 739–745. AAAI Press.

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

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