Appears in Working Notes of the AAAI Workshop on Constraints and AI Planning
Austin, TX, 2000

MACBeth: A Multi-Agent Constraint-Based Planner

Robert P. Goldman, Karen Zita Haigh, David J. Musliner, Michael Pelican
Honeywell Technology Center
Minneapolis, MN 55418
{goldman, khaigh, musliner}@htc.honeywell.com

Introduction

MACBeth is a constraint-based tactical planning engine for multi-agent teams. MACBeth is designed for domains in which a human user must quickly specify a mission to a team of autonomous agents. In these domains, "puzzle-mode" thinking to come up with novel plans is not important; the key task is to rapidly and accurately tailor existing plans to novel situations. To this end, MACBeth combines hierarchical task network planning with modern constraint reasoning techniques, into a mixed-initiative planning system. This mixed-initiative planning system is driven by a graphical user interface inspired by a "playbook" metaphor, to generate, check and modify plans for teams of heterogeneous agents. MACBeth has been tested in two robotics domains: Unmanned Combat Aerial Vehicles (UCAV) sorties and Tactical Mobile Robotics (TMR).

MACBeth is designed for domains in which a human user must quickly specify a mission to a team of autonomous agents. To do this, the user must be able to simply and easily specify a high-level "play" for the team to perform. The user must, however, be able to tailor this plan; completely automatic planning is not acceptable. In every tactical planning scenario, the human user brings context information to the planning process that is not directly available to the automation. For example, in the UCAV domain, the wing commander must identify the objective of the sortie. In some cases, the commander may want to add further restrictions on the behavior of the UCAVs. For example, he might indicate that the UCAV should circumnavigate airspace belonging to a noncombatant nation, or select a different attack formation based on his own skills and preferences.

In tactical applications, the mission planning system must *ensure that plans are feasible.* An attempt by the user to specify an infeasible plan should be flagged by the planning algorithm as soon as possible, so that she can retract a decision or reconcile conflicting objectives early in the planning process. For example, in a UCAV domain, the commander might initially assign two widely separated reconnaissance targets to a single UCAV with insufficient fuel to reach both locations. The planner should identify the inconsistency quickly, ideally before it expands planning effort on plan details, such as determining final approach routes for the two targets, and immediately alert the user that she may need to modify her objectives or the allocation of her resources.

The combination of HTN planning, constraint programming and playbook GUI was chosen to meet these needs. HTNs provide a representation for plans that is relatively easy for people to understand. They can also support very efficient planning, when the ability to construct novel plans (completeness) is less of interest. The HTN plans provide a skeleton on which to perform constraint reasoning.

Constraints on MACBeth's plans capture the complex relationships between (otherwise unrelated) tasks, allowing MACBeth to reason about the tradeoffs of different resource allocations. MACBeth's constraint handling assists its users in resource management and in appropriately assigning mission roles to team members, supporting human planners who must manage heterogeneous teams. The user does not need to reason about which agent does which task, or how to reallocate agents in order to achieve her goals.

A domain-specific "playbook" interface allows MACBeth's user to control a team of agents at the level of task and basic requirements on that task. Using the playbook GUI, the user specifies requirements to the planner, which will generate a plan for the team of agents. This interface allows for multiple degrees of adjustable autonomy, providing the user with varying levels of control depending on his requirements.

The playbook GUI does not impose much cognitive or perceptual load on the system's user. The automatic consistency-checking supplied by MACBeth's constraint-handling also simplifies mission planning, e.g., ensuring that a planned mission will not exceed fuel supplies.

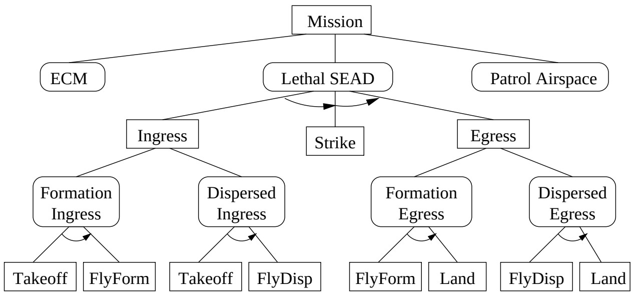

Figure 1: A hierarchical task decomposition for a UCAV domain. Ovals represent *and-nodes*; rectangles represent *or-nodes*.

Mission Task Model

We use Hierarchical Task Networks(HTNs) as the framework to encode, track and dynamically modify the plans, goals, tasks and objectives specified by humans. HTNs provide a number of advantages: They provide a representation that is relatively understandable by human users, for mixed-initiative planning. Although in the worst case, HTN planning is more difficult than, e.g., STRIPS planning (Erol, Hendler, & Nau 1994a), in practice it is often more efficient (Wilkins 1988), as in our application. Also, unlike "first principles" planners, which choose actions based on their effects, HTNs make it easy to specify actions to be done, or goals to be achieved "just because" — e.g., in this case I know that taking an aspirin is a good thing to do, although I don't have a good model of the effects of the action. Most important for this discussion, the HTNs provide a structure for encoding constraints and incorporating them as appropriate.

In Hierarchical Task Network (or "decomposition") planning, the problem is to flesh out a sketchy plan for the planning problem, rather than to build a path from an initial state to some member of a goal state set. Part of an HTN for the UCAV domain is shown in Figure 1. In the initial state of an HTN planning problem, one has a task network that specifies only the goal to be achieved; in the example of Figure 1, say a network with only a "Mission" node. The problem is to build a full tree down from the initial node through "methods" and "goals." Methods (or operators) are AND nodes that provide a means of achieving some goal, in terms of a set of steps (or sub-goals). E.g., to perform a "Lethal SEAD" mission, you must first "Ingress," then "Strike" and finally "Egress." Goals are OR nodes that specify a set of methods that are applicable. For example, one can perform "Ingress" flying in formation or dispersed.

The problem of HTN planning is one of *constrained* AND/OR tree search. Constraints may be attached to nodes in the initial network (e.g., fuel supply is 10,000 gallons). Method definitions specify the constraints to be created when the operator is instantiated. Constraints can be specified between tasks and their parents in the abstraction hierarchy, their siblings, and their children, or on the values of their parameters (perhaps as related to the parameters of their parents, siblings, or parents). For example, the fuel used during the ingress phase of a aircraft sortie will be greater than or equal to the sum of the fuel used by its children. In Figure 1 we graphically indicate temporal precedence constraints.

Playbook GUI and User Constraints

The task model is designed to enable easy intuitive human control. This representation allows us to develop a *playbook GUI* (Miller & Goldman 1997), a natural way for the human to interact with the planner. The playbook concept was influenced by sports playbooks, where each player knows the broad outline of a "play", and each play has been well-practiced by the team. During an actual game, the play is "tailored" by a coach or team captain to meet the conditions on the field.

In the MACBETH paradigm, "calling a play" means that the human user (analogous to the coach) declares that a certain task needs to be accomplished. The user can "tailor" the play by adding constraints and requirements specified by operating conditions. Tailoring a plan essentially allows for adjustable autonomy on the robots. In this manner, the user can task automation subsystems to behave appropriately for the performance of that task.

As the planner expands the plan, it displays the consequences of planning decisions to the user, who can retract previous choices, or make better-informed decisions about available choices.

A playbook GUI should be designed to allow different levels of control depending on the domain, task and the target user. As the user's knowledge and skills decrease, the robot's autonomy needs to increase. In TMR, the target user is a soldier squad leader or robot operator within a squad. The plays contain internal timing constraints, and certain actions (e.g. take a photo) have related hardware requirements. Most remaining constraints are map- or environment-related, including for example navigation waypoints and goal locations (specified by the user) and required locomotion capabilities (specified by the route planner).

Figure 2 shows a sample playbook GUI for the TMR domain. In the upper left corner, the GUI displays the list of available plays. In the lower left corner, the GUI shows status information for a selected robot. The right hand side displays the map, the home base

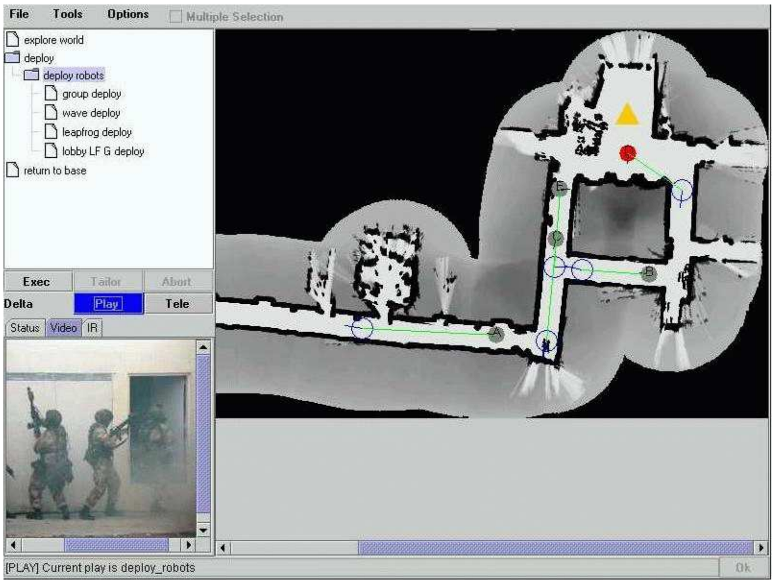

Figure 2: Playbook GUI for a TMR deployment domain.

(triangle), the robots' current locations (filled circles), and the robots' goal locations (clear circles). When the user selects a play, he can then either (1) click `Execute`, and MACBeth will generate and execute a plan, or (2) click `Tailor`, and MACBeth will generate a (constraint-free) plan, and then allow the user to set constraints and otherwise tailor the play to the current situation, executing the (constrained) plan only after the user clicks `Execute`.

The TMR GUI was written in Java for operation inside a web browser. The UCAV GUI was written using Tcl/Tk and the daVinci graph visualization tool (Frohlich & Werner 1997).

MACBeth Planning

MACBeth uses Hierarchical Task Network (HTN) planning (Erol, Hendler, & Nau 1994b; Wilkins 1988) and constraint logic programming techniques (Jaffar & Michaylov 1987; Hentenryck 1989) to generate mission plans that meet specifications provided by a human user. MACBeth analyzes the user's specifications for feasibility, and automatically generates mission plans consistent with the user's requirements and restrictions. MACBeth will also prefer optimal plans when multiple alternatives are consistent and feasible.

The design of the mission task model ensures that the broad outline structure of a mission is known before the planning has begun. MACBeth's job is to assist the user by managing the resources, deadlines, and constraints between alternative actions during the development of the plan. The planner represents these limited quantities as constraints on and between individual plan operators, and manages them with an internal constraint management engine.

MACBeth differs from existing "strategic" planning systems in that decomposition and method choice is not the primary focus of the planning activity. Strategic planning systems (e.g. (Georgeff & Lansky 1987; Nourbakhsh 1997; Veloso *et al.* 1995)), on the other hand, typically construct plans based on a study of the preconditions and effects of actions, and hence may construct plans whose steps are not well-known to the user. Instead of focusing on such "puzzle-mode" thinking, which is not appropriate in the tactical setting, MACBeth focuses on handling the complex constraints needed to manage resources such as fuel, to follow communications doctrine, to synchronize multiple agents, and to influence role assignments. Previous planning systems, even constraint-centered planners like O-Plan2 (Bienkowski & desJardins 1994; Pell *et al.* 1997; Tate, Drabble, & Dalton 1994), do not incorporate modern constraint-solving techniques, and few are easy to integrate with special-purpose solvers.

On the other hand, constraint programming provides a unifying framework in which to address problems like resource management, synchronization, etc., but does not provide a convenient way of specifying such problems. The HTN planning framework provides a representation that enables MACBeth to systematically compose hybrid constraint problems that integrate resource management, role assignment and method choice.

Combining HTN planning and constraint propagation provides an efficient way to specify a wide variety of problems. MACBeth is implemented in SICStus Prolog, using Constraint Logic Programming for Finite Domains (clp(FD)) to manage time and other resources.[1]

Planning Process

MACBETH uses HTN planning to flesh out the high-level (non-primitive) tasks in the plan. Whenever the planner identifies a non-primitive task in the plan, it decides whether the node can be decomposed by applying one or more methods to the task. In the case that more than one method applies, the planner develops all of the alternative paths. Infeasible alternatives are pruned from consideration.

At each step in the planning process, MACBETH determines the feasibility of the current partially specified plan. As each task is expanded, constraints from the corresponding method are added to the clp(FD) constraint management engine. As the plan is built, constraints propagate both up, from sub-tasks to tasks, and down, from tasks to their expansions. This constraint propagation process enables the planner to quickly identify flaws in the plan. If some methods for achieving a task do not meet the user-imposed mission constraints, then MACBETH will eliminate those methods from consideration.

For example, if there are two possible attack routes for an aircraft, MACBETH will verify that both routes meet fuel and timing constraints. If both routes meet all constraints, then MACBETH will select the better of the routes (based on some previously-specified evaluation criteria). If one route requires too much fuel or takes too long, then MACBETH will eliminate that route from consideration, selecting the feasible route. If neither route meets the user's constraints, then MACBETH will tell the user that his plan is infeasible.

MACBETH can also use constraint management techniques to manage resources that are widely separated in use, assuming that the constraints were specified in the task model. For example, in the TMR domain, if one task requires a certain hardware capability, then the constraint store will keep track of which robots could be assigned to that task. This kind of constraint is automatically propagated throughout the plan, allowing MACBETH to effortlessly detect complex resource interactions that most other planning systems could only detect with difficulty. For example, if a "guard" task requires a colour camera, and robots *Alpha* and *Beta* both have colour cameras, then MACBETH will ensure that one of these two robots is assigned to the "guard" task. If *Beta* is later assigned a different, unrelated task, the MACBETH will automatically assign *Alpha* to the "guard," with no additional introspection or backtracking.

[1] We expect later to incorporate more flavors of CLP to incorporate different kinds of resources, etc., but clp(FD) is adequate for our current applications.

Another benefit of the constraint management engine is that MACBETH can easily coordinate different tasks. If two different tasks need to be, say, initiated at the same time, the designer can easily specify this timing constraint. During plan expansion, MACBETH will bind the two start times together. Any variable that affects the start time of either action will hence affect the start time of both actions. Each time such a variable is modified, MACBeth's constraint engine will propagate the the effects throughout the plan. In this way, for example, two TMR robots could easily be instructed to enter a building from different doors at exactly the same time. If one arrives at its door before the other, then the first one will wait until the second one is ready.

Table 1 outlines the planning process. The core of MACBeth's resource management capabilities lie in the function instantiate_variables(). Prolog's variable unification mechanism allows constraints to flow throughout the tree as specified by the connections described in the operators (lines 11 through 19). Note that for an *or-node*, variables are copied rather than unified with the value of the parent node (lines 16 and 17); in this manner, MACBETH can expand each method individually without interference from other unrelated methods. When the user chooses between methods (or MACBETH selects one in preference to the others), the constraints are propagated up to the parent and hence through the rest of the tree. The clp(FD) constraint management engine then stores and keeps track of the effect of constraints on the variables (lines 20 and 21).

The planner uses constraint propagation techniques to manage resources, coordinate the individual plays, to ensure that user-imposed constraints are met, and to opportunistically prune regions of the plan space when choices in one part of the plan force choices in another part.

We have not constructed proofs of soundness and completeness for the algorithm, but we believe it to be so. The tree search in `generate_plan_tree` will be sound and complete as handled by Prolog's depth-first-search, and we have not added excess constraints that to the clp(FD) engine that would prune any correct plans. Note, however, that the planner may be viewed as incomplete with respect to an understanding of its domain, because the domain knowledge engineer may force certain parameter choices to be made by the user. For example, the knowledge engineer in the UCAV domain stipulated that the user must specify the target of an airstrike, rather than permitting the planner to choose this location at its own discretion! User specifications are discussed in the following section.

```
generate_plan_tree(Node)
    if Node has mandatory, unbound variables
        cannot expand node until value is specified by user
    else if Node has available methods (Or-node)
        foreach available method,
            create a Child of Node for method
            instantiate_variables("or-node", Child, Node )
            generate_plan_tree(Child)
    else if Node has required steps (And-node)
        foreach available step,
            create a Child of Node for step
            instantiate_variables("and-node", Child, Node )
            generate_plan_tree(Child)
    else primitive operator

instantiate_variables(Type, Child, Parent)
    if Type is "or-node"          foreach variable in Parent to be passed to Child,
                                      Copy the Parent's value to the Child's value.
    else if Type is "and-node"    foreach variable in Parent to be passed to Child,
                                      Unify the Parent's value to the Child's value.
    constrain any typed parameters in Child
    apply constraints specified in operator to variables of Child
```

Table 1: The core of MACBeth's planning process.

Handling User Specifications

After every user action, MACBeth incorporates the user's decision into the current plan. User actions shape the direction of the evolving plan. User actions fall into three classes:

1. specifying undefined variables,
2. choosing between alternative methods for task completion, and
3. introducing timing or ordering constraints.

By specifying the task parameters, the user shapes the plan's broad outline and identifies important task characteristics. Typical parameters provided by the user include mission priorities, available equipment, and specific targets. By choosing between alternative methods for task completion, the user can express a preference between feasible plans. Finally, by introducing timing constraints the user can specify team coordination, or a preferred sequence of tasks.

When the planner reacts to a user action, there are several possible outcomes. Two are failures:

1. The planner cannot find an appropriate expansion. The attempt to expand the plan fails.
2. The constraints on the new method violate existing constraints (e.g., a method requires satelite communication between aircraft but no satelites are available in this region).

A third just indicates that the planner needs more assistance from the user:

3. The planner determines that there is not enough information available to expand this node (e.g., possible ingress formations cannot be determined until the number of aircraft is specified).

A final alternative is that the planner manages to incorporate the user action successfully. Such success can trigger considerable action by the planner:

1. The new constraints eliminate some choices for the expansion of other nodes. This elimination may allow the planner to further expand that part of the plan.
2. The new constraints sufficiently restrict the range of a parameter so that another task in the network can be expanded.
3. The expansion of the task creates new non-primitive tasks that may be expandable.
4. A primitive (unexpandable) task is created.

Some constraints are mandatory; that is, MACBeth will not execute a plan until the information has been supplied. For example, the user must specify a target location in the UCAV domain.

Whenever a non-mandatory constraint has not been supplied by the user, MACBeth will assume it has complete freedom of choice. For example, unless the user specifies location to guard, MACBeth will assume that the TMR robots are free to explore unknown space.

Similarly, when there are multiple paths through the plan tree (i.e. multiple possible methods or available resources) that will accomplish the task and that meet constraints, MACBeth will arbitrarily select which one to attempt. If a preference function has been supplied (e.g. select the shortest route, select the robot with the most power), MACBeth will select the preferred option.

As the user supplies more and more constraints, the planner's freedom is more and more reduced. Essentially, this relationship specifies degrees of adjustable autonomy. At the highest level, the user needs to only supply mandatory constraints. MACBeth will then assume full autonomy, and generate a plan accordingly. At an intermediate level (the lowest while still using the playbook), the user specifies every constraint, including which of multiple alternative methods to select to accomplish a task. The robots are still free to avoid obstacles and navigate between waypoints autonomously. Finally, at the lowest level of robot control, the user can tele-operate the robot, giving the robot no freedom at all.

The planner currently only invokes the labeling process when the plan is to be completed. The labeling process assigns values to those variables for which constraint propagation has not forced a choice. Variable assignment may not be forced either because the existing set of constraints does not dictate a single value for the variable or because constraint propagation is incomplete. For the latter case, a co-process using labeling could provide further pruning of inconsistent plan choices. We have not implemented this alternative because Prolog's constraint propagation is complete for the class of constraints we have used in our domains (i.e., any value that is labeled as consistent is, in fact, consistent).

Integration With Specialized Problem-Solvers

The structure of the MACBeth planner provides a straightforward way to incorporate special-purpose problem solvers like a route planner. When a problem-solver's inputs can be mapped to plan operator parameters, and its interactions with other operators can be described using constraints, it can be integrated with MACBeth.

In the UCAV domain, the route planner uses a dynamic programming algorithm to find a minimal altitude trajectory between two locations, that is, one that results in minimal exposure to enemy radar and visual detection. The resulting path determines the minimum flight time and fuel requirements for the action, which are posted as constraints on the plan. If the flight time or fuel requirements of the action violate current constraints on the plan, the route planning action will be undone and removed from the plan tree.

Conclusions

MACBeth combines HTN planning with constraint propagation techniques. It provides support for a playbook GUI, through which human users can specify instructions and constraints on tasks. MACBeth has been used for a UCAV domain and a TMR domain. For TMR, MACBeth is part of a complete end-to-end system, in which a human user uses a playbook GUI to control a team of robots at Carnegie Mellon University.

References

Bienkowski, M. A., and desJardins, M. E. 1994. Planning-based integrated decision support systems. In Hammond, K., ed., *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS-94)*, 196–201. Chicago, IL: (Menlo Park, CA: AAAI Press).

Erol, K.; Hendler, J.; and Nau, D. S. 1994a. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1123–1128. Menlo Park, CA: AAAI Press/MIT Press.

Erol, K.; Hendler, J.; and Nau, D. S. 1994b. UMCP: A sound and complete procedure for hierarchical task network planning. In Hammond, K. J., ed., *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference*, 249–254. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Frohlich, M., and Werner, M. 1997. The daVinci graph visualization tool. http://www.informatik.uni-bremen.de/~davinci.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-87)*, 677–682.

Hentenryck, P. V. 1989. *Constraint Satisfaction in Logic Programming.* (Cambridge, MA: MIT Press).

Jaffar, J., and Michaylov, S. 1987. Methodology and implementation of a CLP system. In *Proceedings of the Fourth International Conference on Logic Programming.* (Cambridge, MA: MIT Press).

Miller, C., and Goldman, R. P. 1997. "tasking" interfaces; associates that know who's the boss. In *Proceedings of the Fourth Human Electronic Crew Conference.*

Nourbakhsh, I. 1997. *Interleaving Planning and Execution for Autonomous Robots.* (Dordrecht, Netherlands: Kluwer Academic). PhD thesis. Also available as technical report STAN-CS-TR-97-1593, Department of Computer Science, Stanford University, Stanford, CA.

Pell, B.; Bernard, D. E.; Chien, S. A.; Gat, E.; Muscettola, N.; Nayak, P. P.; Wagner, M. D.; and Williams, B. C. 1997. An autonomous spacecraft agent prototype. In *Proceedings of the First International Conference on Autonomous Agents*, 253–261. Marina del Rey, CA: (New York, NY: ACM Press).

Tate, A.; Drabble, B.; and Dalton, J. 1994. Reasoning with constraints within O-Plan2. In Burstein, M. H., ed., *ARPA/Rome Laboratory Knowledge-based Planning and Scheduling Initiative*, 99–109.

Veloso, M. M.; Carbonell, J.; Pérez, M. A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7(1):81–120.

Wilkins, D. 1988. *Practical Planning.* (San Mateo, CA: Morgan Kaufmann).