

The Design of ALPS: An Adaptive Learning and Planning System*

Randall J. Calistri-Yeh
ORA
301 Dates Dr.
Ithaca, NY 14850-1326
(607) 277-2020
calistri@oracorp.com

Alberto M. Segre
Dept. of Computer Science
Cornell University
Ithaca, NY 14853-7501
(607) 255-9096
segre@cs.cornell.edu

Abstract

In order to perform adequately in real-world situations, a planning system must overcome intractable domains, limited planning time, incorrect and uncertain world knowledge, dynamic environments, and real-time response. ALPS, an adaptive learning and planning system, combines traditional components of hybrid planners with innovative techniques from machine learning, distributed computing, and probabilistic reasoning to create a unique architecture ideally suited to address these complex issues.

Introduction

There are several properties of real-world planning that make it an extremely rich and challenging research problem.

- Real planning systems have only a limited amount of time to plan their actions; in many domains, coming up with a plan too late is just as bad as not coming up with one at all. Ideally, it should be possible to interrupt the planner at any time and demand the best plan that it has found so far. Given more planning resources, the system should continue to improve the plan.
- Between the time that a real-world plan is constructed and the time that it is executed, the actions of external agents may have caused the actual world state to diverge from the planner's assumptions so the plan may no longer execute properly.
- Real domains are typically so complex that any attempt to formulate a logical domain theory describing how the world works will produce only a rough approximation of the intended domain theory. This approximation is almost certainly incomplete, incorrect, and inconsistent.
- Planning agents operating in the real world are sometimes exposed to crisis situations where they have to act immediately in order to survive. They

may have access to only limited local sensor information and may need to react "instinctively" to perceived imminent danger. This reaction may end up invalidating any plan that was being developed.

This paper presents work in progress on a new planning architecture called ALPS (Adaptive Learning and Planning System) that is being designed to address exactly these issues. We first present the general architectural design of ALPS. We then discuss some innovative features, compare ALPS to other architectures, and describe the current implementation.

The ALPS Architecture

The ALPS architecture is shown in Figure 1. It consists of three main modules: a deliberative planner that can generate base plans and refine them incrementally over time to achieve optimal performance, a plan critic that can project a proposed plan or action into the future and simulate the results, and a reactive executive that assimilates information from external sensors and the other two modules and decides on the next action to take; in the absence of other information, it can follow a hardwired "survival" strategy. These modules are supported by a probabilistic truth maintenance system (TMS) that handles queries about the world and temporal relations between events. Within the three main modules, there are capabilities for plan repair, domain repair, and machine learning techniques to improve the overall efficiency of the planner.

Control Flow

In contrast to many hybrid planning systems, which use a separate control module, all control in the ALPS design is centered in the reactive executive. The deliberative planner and the plan critic are just considered to be special sensors that feed into the reactive executive. These sensors are expensive in that they may consume valuable resources and may not produce timely results. The reactive executive must make a conscious decision to invoke one of these modules. During each control cycle, the reactive executive extracts information from all of its available sensors, consults its executive strategy, and selects an action to perform.

*Support for the research described herein has been provided by the Advanced Research Projects Agency through Rome Laboratory Contract Number F30602-93-C-0018.

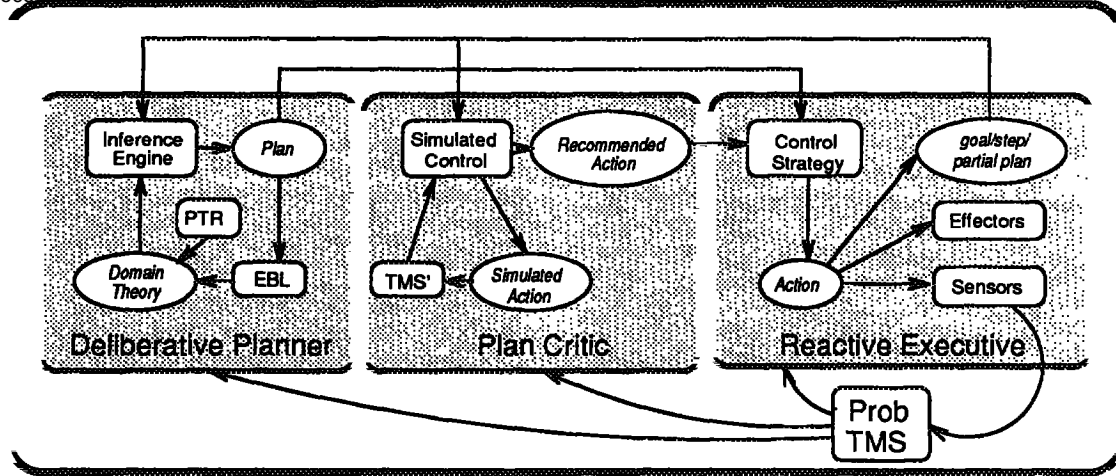


Figure 1: The ALPS architecture. Boxes are processes, ellipses are data.

Certain constant-time sensors are always guaranteed to be available and can be used for survival maneuvering in the worst case. The executive may choose to invoke the deliberative planner, the plan critic, both, or neither. It can specify a maximum search depth for each of these modules, which roughly translates into planning time.

Reactive Executive

The ALPS reactive executive assimilates sensor and plan information and decides what action, if any, should be taken at the current time. The ALPS executive has no memory; instead, it makes a decision based on a prespecified heuristic strategy and local conditions as reported by a collection of sensors. Some of these sensors operate in constant time and their data is always available immediately. Other sensors may be more expensive to operate, and their data may not always be available. In particular, the deliberative planner and the plan critic are just treated as expensive sensors by the reactive executive. If the sensor data is available, it can be used to influence the heuristic strategy; otherwise the strategy will use only the constant-time sensors. In this fashion, we guarantee that the cost of deciding what to do at any given time is not dependent on the size of the problem being solved, but rather only on the expressiveness and number of sensors.

Deliberative Planner

The ALPS deliberative planner is a classical deductive planning system that constructs plans from a given goal specification and a knowledge base of plan information. It uses DALI (Segre & Sturgill 1994), a novel deductive inference engine that is *adaptive* in the sense that its performance characteristics change with experience. This adaptive capability is not only domain-independent, but is completely transparent to

both the planning component and the author of the domain theory. Performance improvements are due to the incorporation of multiple speedup learning techniques such as bounded-overhead success and failure caching (Segre & Scharstein 1993), explanation-based learning (EBL) (Segre & Elkan 1994), and conjunct reordering heuristics. These speedup techniques work by biasing the search to find some proofs quickly at the expense of others. These techniques do not sacrifice the logical coverage of the search, but affect only the runtime performance of the search engine. In the context of resource-limited search, this means some problems that were not previously solvable in reasonable time will now be solved within the resource limit.

ALPS can support dynamic reconfiguration of system parameters in order to tune system performance. For example, our bounded-overhead success and failure caches needn't remain fixed in size; the system may dynamically enlarge or reduce the storage space used for caching in order to improve overall performance.

Another unique aspect of our adaptive inference work is that it can scale to very large problems by using heterogeneous distributed computing systems. We use an approach called *nagging* (Sturgill & Segre 1994), a low-bandwidth, robust, dynamically reconfigurable algorithm for transparently distributing planning tasks among many processors.

ALPS produces optimal plans using a method called *iterative strengthening* (Calistri-Yeh 1993). Iterative strengthening is an anytime algorithm that rapidly generates a satisficing plan, then incrementally improves it based on customizable optimality criteria. The more time the planner is given, the better a plan it can generate. This approach has the benefit that in time-critical situations we can begin to act on a correct but possibly suboptimal plan if necessary, without waiting for complete optimization.

Plan Critic

The plan critic serves as a simulator for both the deliberative planner and the reactive executive. It takes a full plan, a partial plan, or a single action and projects it into the future, "looking ahead" to see how beneficial that course of action appears. The plan critic can adjust how far ahead to project, based on how much planning time is available.

The deliberative planner can use the plan critic to simulate a plan, checking for potential trouble spots in the plan. In addition to the standard strategies such as expected utility to provide the most likely result and minimax algorithms to provide worst-case analysis for bulletproofing critical subplans, the plan critic also supports *pessimistic planning*. Pessimistic planning augments a minimax algorithm with probabilistic information about the likelihood of the actions and events, and considers only moves above a certain probabilistic threshold or adversarial utility measure. Pessimistic planning can be used to check for potentially damaging situations without getting bogged down in wildly implausible scenarios.

Plan Repair

The ALPS architecture includes mechanisms to anticipate potential failures before they occur and to recover from failures by making minor changes to the plan that already exists. Rather than having a separate plan repair module, our method of plan repair and failure avoidance spans the entire planning process from generation to execution to monitoring.

ALPS handles failure anticipation in two ways. In the first method, the deliberative planner tags certain facts in the TMS as having to be true at execution time. If the TMS ever alters a tagged fact as it updates the state of the world, the plan repair component is notified that the plan must be fixed. In many cases, this notification will happen long before the executive reaches the affected portion of the plan, and the plan can be repaired before the executive would even notice that it is broken. The second method anticipates failures at generation time instead of at runtime. If the inference engine is given sufficient time, it can build *contingency plans* directly into the plan. This approach is ideal for an anytime incremental planner such as ALPS, since generating the contingency plans is entirely optional. If planning resources are limited, the contingency plans will never be generated, but we will be no worse off than with a traditional planner. The more time that is spent on planning, the more robust the final plan will become.

When a plan fails at execution time, the ALPS executive will use its hardwired "survival" strategy to produce real-time reactive behavior until the deliberative planner can repair the plan. The deliberative planner will apply a set of domain-independent repair methods augmented by specific rules extracted from the domain theory. These repair methods differ from the pre-coded "contingency plans" described above in

that the deliberative planner still needs to be invoked to flesh out the plan.

Domain Repair

One of the main problems in building planning systems is that the domain knowledge used in plan generation is typically only approximately correct. The *theory revision problem* is the problem of how best to go about revising a knowledge base of domain theory on the basis of a collection of examples, some of which expose inaccuracies in the original theory. There may be many possible revisions that properly account for all of the observed examples; the goal is to find a revised knowledge base that is both consistent with the examples and as faithful as possible to the original theory.

Our *Probabilistic Theory Revision* (PTR) algorithm for propositional logic (Koppel, Feldman, & Segre 1994) demonstrates several advantages over other theory revision approaches (Ginsberg 1988; Ourston & Mooney 1990). First, it is capable of exploiting any additional information provided by the domain expert reflecting his or her initial confidence in each knowledge base component. Second, it honors the principle of representational transparency by retaining intact the structure of the knowledge base during the revision process. Third, PTR is efficient since it processes examples one at a time (rather than simultaneously) within the most time-critical portion of the revision process. Fourth, PTR exhibits some interesting and desirable formal properties by virtue of its underlying probabilistic framework.

Discussion

We now compare the ALPS design to other planning architectures, discuss scaleup and portability issues, and describe the current implementation status of the ALPS system.

Comparison

Our design of deliberative and reactive planning differs from other proposed hybrid designs. Lyons and Hendriks (1992) use the deliberative planner to adapt the reactive system so that the reactive behavior becomes more goal-directed. Mitchell's Theo-Agent architecture (Mitchell 1990) has its control centered in the reactive component like ALPS. However, Theo-Agent reverses the roles of planning and reacting: it reacts under normal circumstances and stops to plan in unexpected situations. In contrast, ALPS uses its reactive executive to guarantee that some action is available at all times, especially in unexpected situations.

Instead of the common design of a single isolated plan repair module, plan repair in ALPS covers the entire planning process and incorporates diverse methods such as hard protection intervals (McDermott 1990), contingency plans, domain-independent repair methods (Howe & Cohen 1991), and domain-specific recovery rules (Turner 1987).

Planners that are designed to produce optimal solutions typically implement some form of best-first search based on an algorithm such as A* that requires an *admissible* heuristic (Pearl 1984); these planners produce no useful results until the optimal plan is found. In contrast, iterative strengthening does not require an admissible heuristic and can be used as an anytime algorithm because satisfying plans are available during the optimization process.

Most of the work toward applying parallelism to logical inference has centered around either OR-parallelism (Schumann & Letz 1990), which explores nondeterministic choice points in parallel, or AND-parallelism (DeGroot 1984), which executes subprocedures in parallel. These approaches are susceptible to network failure, have high communication overhead, and do not extend well to first-order logic. Nagging, on the other hand, has very low communication overhead, is completely fault-tolerant with respect to network performance, and is directly applicable to first-order domains.

Scaleup

The ALPS architecture has been designed especially to address scaleup issues. By combining the approaches of both deliberative and reactive planning in a careful way, ALPS is able to get the best of both worlds. Our design keeps a clean separation between the polynomial modules that must operate in real time (reactive executive and TMS) and the exponential modules (deliberative planner and plan critic).

Within the deliberative planner, ALPS applies several transparent methods to speed up the plan generation process. Bounded-overhead caching stores intermediate results to avoid redundant search where the result is already known. Explanation-based learning creates generalized rules from previous problems to provide "short cuts" on similar problems in the future. Nagging exploits idle processors in a distributed network to prune "dead ends" from the search space.

Portability

The ALPS design lends itself very nicely to portability to new domains. Each part of the architecture separates cleanly into domain-dependent components and domain-independent components.

- The deliberative planner is based on a general-purpose inference engine. The only part of the deliberative planner that is domain specific is the domain theory itself, which is completely self-contained.
- The plan critic is designed as a general-purpose simulator. It relies on a theory of the specific domain to provide information about likely successes and failures of plan steps. We hope to use the same domain theory as the deliberative planner uses.
- The reactive executive is primarily a control module with access to the other modules. It employs sensors and effectors that are specific to a particular domain,

and part of the executive strategy contains domain-specific knowledge.

So porting to a new domain should (in theory) only involve providing a new domain theory, a new executive strategy, and descriptions of the domain-specific sensors and effectors. ALPS will also support hooks to include optional specialized knowledge and procedures.

Current Status

ALPS is being developed as part of the ARPA/Rome Laboratory Planning Initiative (ARPI), whose goal is to develop and demonstrate the next generation of AI planning technology as applied to crisis action transportation planning. ALPS uses a problem generator to produce fixed or random transportation problems from a knowledge base of US airports, aircraft characteristics, and cargo requirements.

The ALPS system is still under development, and some sections are not yet completed. The deliberative planner is fully implemented in Common Lisp, including the iterative strengthening optimizer and all speedup techniques discussed above except parallel nagging. Domain theories for the Lisp version are currently restricted to Horn clauses. Probabilistic theory revision (PTR) has been implemented for propositional theories and will be extended to full first-order theories in the future.¹ The deliberative planner is also being reimplemented in C. The C version supports full first-order domain theories and parallel nagging, but does not yet support EBL or theory revision.

The plan critic currently consists of a preliminary plan simulator for the transportation planning domain. Based on the probability of success of each flight segment, it will simulate each segment and report on whether the segment reached its destination and how late it was. The plan critic does not yet perform pessimistic planning, accept partial plans, or generate recommended actions.

The reactive executive is not yet implemented.

Evaluation

We have tested the Lisp version of the deliberative planner on a suite of transportation problems generated by the ALPS problem generator. We have used the iterative strengthening optimizer to produce optimal plans based on criteria such as minimizing total plan execution time, minimizing number of aircraft, maximizing number of aircraft, and maximizing probability of plan success. Preliminary results indicate that the iterative strengthening optimizer performs quite well; failure caching demonstrated a potential for dramatic speedup in iterative strengthening. But runtime overhead is still too high in the current domain implementation to achieve any practical benefit; see (Calistri-Yeh 1994) for more details.

¹PTR is currently a standalone module that is not yet fully integrated with ALPS.

While we have not yet have tested the explanation-based learning algorithms and the parallel nagging algorithm in the transportation domain, we have tested them in other domains, including many of the benchmark problems from the TPTP (Thousands of Problems for Theorem Provers) suite.

Conclusions

The architecture that we have developed for the ALPS planning system allows us to apply new and powerful methods to the challenges found in planning problems. It combines techniques from classical planning, reactive planning, machine learning, probabilistic reasoning, distributed programming, domain theory revision, and plan repair in order to overcome the problems of intractable, incomplete, and incorrect domain theories, of dynamic environments, and of real-time resource constraints. We have designed the ALPS architecture to handle scaleup to large problems gracefully, to support porting to new domains and applications, and to be appropriate for the current domain of crisis action transportation planning.

Sample Program Traces

This appendix shows an execution trace of ALPS generating an optimal plan for the transportation problem described in Figure 2. The problem represents a hypothetical rescue operation after an earthquake in San Francisco. The cargo units correspond to National Guard troops, emergency food rations, temporary shelters, and earthmoving equipment. In this example, a "punit" is equivalent to one aircraft. Tables 1 and 2 describe knowledge about the airports and aircraft that impose additional constraints on the problem.

In Figure 3, we ask ALPS to generate the plan that requires the minimum amount of time to execute. The third line of the transcript reports that the first unconstrained solution has been found (the debugging information essentially measures the amount of effort expended by the inference engine using unification steps as the basic unit of effort); three more iterations incrementally strengthen the optimization criteria until the fastest plan is found. The final plan will take 28.7 hours to execute, which is substantially better than the initial plan of 47.4 hours. Once the initial plan is constructed, the user could have interrupted ALPS at any time, and ALPS would have reported the best plan found so far. Note that although the solution appears to show instantaneous takeoffs and landings, ALPS actually accounts for loading, unloading, and refueling times; in the current transcripts, these times are aggregated into the flight time.

References

- Calistri-Yeh, R. J. 1993. Generating optimal plans in ALPS: A preliminary report. Technical Report TM-93-0054, ORA, Ithaca, New York.
- Calistri-Yeh, R. J. 1994. Iterative strengthening: An algorithm for generating anytime optimal plans. In *ARPI-94*, To appear.
- DeGroot, D. 1984. Restricted AND-parallelism. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 471-478.
- Ginsberg, A. 1988. Theory revision via prior operationalization. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 590-595.
- Howe, A. E., and Cohen, P. R. 1991. Failure recovery: A model and experiments. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 801-808.
- Koppel, M.; Feldman, R.; and Segre, A. 1994. Bias-driven revision of logical domain theories. *Journal of Artificial Intelligence Research* 1:159-208.
- Lyons, D., and Hendriks, A. 1992. A practical approach to integrating reaction and deliberation. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 153-162.
- McDermott, D. 1990. Planning reactive behavior: A progress report. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, 450-458.
- Mitchell, T. M. 1990. Becoming increasingly reactive. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1051-1058.
- Ourston, D., and Mooney, R. 1990. Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 815-820.
- Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, Massachusetts: Addison-Wesley Publishing Company.
- Schumann, J., and Letz, R. 1990. PARTHEO: A high-performance parallel theorem prover. In *10th International Conference on Automated Deduction*, 40-56.
- Segre, A., and Elkan, C. 1994. A high performance explanation-based learning algorithm. *Artificial Intelligence*. To appear.
- Segre, A., and Scharstein, D. 1993. Bounded-overhead caching for definite-clause theorem proving. *Journal of Automated Reasoning* 83-113.
- Segre, A., and Sturgill, D. 1994. Using hundreds of workstations to solve first-order logic problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. To appear.
- Sturgill, D., and Segre, A. 1994. A novel asynchronous parallelization scheme for first-order logic. In *Proceedings of the Twelfth Conference on Automated Deduction*. To appear.
- Turner, R. M. 1987. Modifying previously-used plans to fit new situations. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, 1-10.

> (pprint-problem)

Given the following Punits:

- Punit 3 (1 :C141 at CHICAGO)
- Punit 4 (1 :C5 at PHOENIX)
- Punit 5 (1 :C5 at MIAMI)
- Punit 7 (1 :KC10 at LA)
- Punit 8 (1 :KC10 at NY)

Transport the following cargo:

- Move Cargo 1 (135 PAX) from JUNEAU to LA
- Move Cargo 3 (21 BULK) from PORTLAND to LA
- Move Cargo 6 (25 OVER) from CHICAGO to LA
- Move Cargo 7 (200 PAX) from PITTSBURGH to SF
- Move Cargo 8 (11 OUT) from ATLANTA to SF

Subject to the following additional constraints:

- Cargo 7 must arrive at SF before Cargo 8 arrives at SF
- Cargo 6 must arrive at LA before Cargo 3 arrives at LA
- Cargo 3 must arrive at LA before Cargo 1 arrives at LA

Figure 2: Sample problem: "earthquake" scenario.

Airport	Runway Length
Atlanta	11782 feet
Chicago	11061 feet
Juneau	9818 feet
LA	10973 feet
Miami	9551 feet
NY	10547 feet
Phoenix	10165 feet
Pittsburgh	9755 feet
Portland	9019 feet
SF	10269 feet

Table 1: Runway lengths.

	C141	C5	KC10
Block speed	418 mph	432 mph	455 mph
Onload time	2:25 hours	3:75 hours	5:00 hours
Enroute time	2:25 hours	2:25 hours	1:50 hours
Offload time	2:25 hours	3:25 hours	3:00 hours
Min runway	7350 feet	9150 feet	10350 feet
Max range	4531 miles	6238 miles	8190 miles
Max bulk cargo	23.0 tons	69.6 tons	62.1 tons
Max over cargo	23.6 tons	65.0 tons	26.4 tons
Max out cargo	0	66.4 tons	0
Max pax cargo	153	329	257

Table 2: Aircraft properties.

> (minimize-time)

Getting initial plan

93 unifications, 145 candidates; s=52, d=11
Changing optimization cutoff to (46.40)

105 unifications, 155 candidates; s=50, d=10
Changing optimization cutoff to (31.65)

165 unifications, 227 candidates; s=50, d=11
Changing optimization cutoff to (30.10)

171 unifications, 232 candidates; s=49, d=11
Changing optimization cutoff to (27.73)

Failed to optimize with parameters (27.73)

Solution: 28.73 hours.

Punit 3 (:C141):

- departs CHICAGO at 2.64
- arrives PORTLAND at 7.26
- departs PORTLAND at 7.26 with Cargo 3
- arrives LA at 13.98 to deliver Cargo 3
- departs LA at 13.98
- arrives JUNEAU at 19.11
- departs JUNEAU at 19.11 with Cargo 1
- arrives LA at 28.73 to deliver Cargo 1

Punit 4 (:C5):

- departs PHOENIX at 0.00
- arrives ATLANTA at 4.03
- departs ATLANTA at 4.03 with Cargo 8
- arrives SF at 16.55 to deliver Cargo 8

Punit 5 (:C5):

- departs MIAMI at 0.00
- arrives PITTSBURGH at 2.58
- departs PITTSBURGH at 2.58 with Cargo 7
- arrives SF at 15.36 to deliver Cargo 7

Punit 8 (:KC10):

- departs NY at 0.00
- arrives CHICAGO at 1.80
- departs CHICAGO at 1.80 with Cargo 6
- arrives LA at 13.98 to deliver Cargo 6

Figure 3: Generating an optimal solution: minimizing time.