From: AAAI Technical Report SS-93-03. Compilation copyright © 1993, AAAI (www.aaai.org). All rights reserved.

Neoclassical Planning

Alberto Maria Segre[†] segre@cs.cornell.edu
David Sturgill[†] sturgill@cs.cornell.edu
Jennifer Turney[‡] turney@wilkes1.wilkes.edu

[†]Department of Computer Science
Cornell University
Ithaca, NY 14853

[‡]Department of Mathematics and Computer Science
Wilkes University
Wilkes Barre, PA 18766

Abstract

Classical planning systems have historically been used to provide a domain-independent framework for expressing and using domain-dependent problem solving information. Unfortunately, these systems are intrinsically intractable. In this paper, we describe a system that blends a classical planning formalism with a novel adaptive inference engine in order to produce an adaptive, approximate, planner.

1. Introduction

Classical planning systems perform search. This observation highlights the fundamental problem with these systems: in large domains — like those typically encountered in real world planning problems — intrinsically weak methods coupled with a classical planning domain representation are overwhelmed by the size of the search space.

In order to be of practical use, classical planning systems must necessarily rely on domain-specific heuristics to reduce the amount of search. Real planning domains often require complex domain-dependent heuristics that are difficult to design and even harder to refine or correct. The message of this paper is that real-world planning systems must necessarily be *adaptive*, that is, they must reconcile themselves to their environment by improving, refining, and perfecting their own search reduction heuristics. Therefore, by incorporating automatic acquisition of search reduction heuristics in the underlying inference engine design, we can make a classical planning system practically useful even in non-trivial domains. We claim our planning system is essentially a classical planning system that promises to scale well; in short, a *neoclassical* planner.

This paper describes an adaptive planning system based on an extension of the situation-calculus called PERFLOG [5] and novel *adaptive inference engine*: a theorem prover whose performance changes with experience [18]. Thus because of its underlying theorem prover, our planner is both *approximative* and *incremental*, in that it is able to generate plausible candidate plans quickly and refine these plans when allotted more computation time. The planner serves as one component of our *SEPIA intelligent agent architecture* [19]; SEPIA is a hybrid system, combining our neoclassical planning component with a real-time reactive executive.

We begin by introducing our theorem prover and its general capabilities, exclusive of its search reduction techniques. Next, we briefly describe the PERFLOG formalism, and how it is supported by our adaptive inference engine. We next turn to the theorem prover's underlying search reduction techniques: explanation-based learning, bounded-overhead success and failure caching, heuristic antecedent reordering strategies, learned-rule management facilities, and a dynamic abstraction mechanism. Finally, we briefly discuss our current research on building a distributed (*i.e.*, large-grain parallel) version of our adaptive inference engine.

2. The Inference Engine

We have implemented a backward-chaining definite-clause theorem prover in Common Lisp. The prover's inference scheme is essentially equivalent to PROLOG's SLD-resolution. Axioms are stored in a discrimination net database along with rules indexed by the rule head. The database performs a pattern-matching retrieval guaranteed to return a superset of those database entries which unify with the retrieval pattern. The cost of a single database retrieval in this model grows linearly with the number of matches found and (roughly) logarithmically with the number of entries in the database.

The system relies on *iterative deepening* [10] in order to force completeness in recursive domains while still taking advantage of depth-first search's favorable storage characteristics. As generally practiced, iterative deepening involves limiting depth-first search exploration to a fixed depth. If no solution is found by the time the depth-limited search space is exhausted, the depth limit is incremented and the search is restarted. In return for completeness in recursive domains, depth-first iterative deepening generally incurs a constant factor overhead when compared to regular depth-first search: the size of this constant depends on the branching factor of the search space and the value of the depth increment. Changing the increment changes the order of exploration of the implicit search space and, therefore, the performance of the inference engine.

We perform iterative deepening on a generalized, user-defined, notion of depth while respecting the overall search resource limit specified at query time. Fixing a depth-update function (and thus a precise definition of depth) and an iterative-deepening increment establishes the exploration order of the theorem prover. For example, one might define the iterative-deepening update function to compute depth of the search; with this strategy, the system is performing traditional iterative deepening. Alternatively, one might specify update functions for *conspiratorial iterative deepening* [4], *iterative broadening* [7], or almost any other search strategy.[1]

Unlike PROLOG, our theorem prover produces a structure representing the derivation tree for a successful query rather than just an answer substitution. When a failure is returned, the prover indicates whether the failure was due to exceeding a resource limit or if in fact we can be guaranteed absence of any solution. In addition, the prover supports procedural attachment (*i.e.*, escape to Lisp), which, among other things, allows us to alter the prover's resource limits dynamically.

3. PERFLOG

The PERFLOG planning formalism is an adaptation of classical situation-calculus planning without explicit frame axioms. In their place, PERFLOG uses two simple meta-theoretic rules called *causation* and *cancellation*. The causation rule:

$$holds(?p, do(?a, ?s)) \leftarrow causes(?a, ?p, ?s)$$

states that proposition $?p$ is true in the situation resulting from performing action $?a$ in previous situation $?s$ if action $?a$ is known to cause $?p$ when applied in situation $?s$. For example, if one were operating in the blocksworld, it would be reasonable to assert:

$$causes(holding(?x), pickup(?x), ?s)$$

as part of the description of the *pickup* operator.[2]

The cancellation rule obviates the need for explicit frame axioms:

$$holds(?p, do(?a, ?s)) \leftarrow holds(?p, ?s) \wedge \neg cancels(?a, ?p, ?s)$$

This rule states that one can conclude that $?p$ holds in the situation resulting from performing action $?a$ in previous situation $?s$ provided $?p$ was previously true and action $?a$ does not cancel $?p$ when applied to situation $?s$. Others have proposed similar schemes in the past. The cancellation rule is similar to McCarthy's *inertia axiom* for the situation calculus [12]:

$$holds(?p, do(?a, ?s)) \leftarrow holds(?p, ?s) \wedge \neg abnormal(?p, ?s)$$

Unlike *abnormal*, however, *cancels* discriminates on the action $?a$. In addition, when treated with the standard PROLOG *negation by resource-limited failure* strategy [1], the negated antecedent $\neg cancels(?a, ?p, ?s)$ essentially constitutes an explicit *assumption*: something that is true only because we fail to find a counterproof.

We can use our theorem prover's run-time resource allocation capability to further extend the operational status of PERFLOG assumptions. To see how, consider initially allocating exactly zero resources to all negated subgoals (akin to having asserted an extra database fact $\neg ?x$). This causes every PERFLOG assumption subgoal to succeed immediately, in effect pruning the AND/OR search tree below assumption subgoals (note that the

[1] The *conspiracy size* of a subgoal corresponds to the number of other, as yet unsolved, subgoals in the current proof structure. Thus *conspiratorial best-first search* prefers narrow proofs to bushy proofs, regardless of the actual depth of the resulting derivation. Iterative broadening is an analogous idea that performs iterative deepening on the breadth of the candidate proofs.

[2] While it is possible to describe plan knowledge as a collection of rules that rely explicitly on the PERFLOG causation and cancellation rules, it is much more convenient to describe problem solving operators in a language similar to that of STRIPS [6]. Thus a SEPIA operator consists of sets of *preconditions* and *subgoals*, as well as an *add list* and a *delete list*. While the add list and delete list have the normal STRIPS semantics, preconditions and subgoals differ as in the ARMS planner [15]. More precisely, preconditions are immutable relations, *e.g.*, $block(A)$, which the planner makes no attempt to expand recursively. Subgoals are relations such as $on(A, B)$ which may be recursively achieved by the planner. In the situation calculus, these relations appear augmented with a situation variable, *e.g.*, $holds(on(A, B), ?s)$. SEPIA operators are automatically expanded into collections of PERFLOG rules.

PERFLOG cancellation rule is the sole source of negated literals in a SEPIA domain theory). Once such a proof has been obtained, we increment the resources allocated to negated subgoals and reconstruct the proof, in essence performing iterative deepening in assumption space. As additional resources are allocated to verifying a proof containing assumptions, we can increase our confidence that the assumptions are, in fact, warranted.

4. Speedup Learning Techniques

Our inference engine relies on speedup learning techniques to improve its performance with experience. The underlying assumption is that not all situations are equally likely to occur; that certain situational idioms arise repeatedly and it is thus more profitable to bias search ordering within the prover so that you will more quickly succeed more often.

One speedup mechanism used by the prover is an internal success and failure cache to record previously proven or unproven subgoals. Caching was already known to be empirically effective in reducing search for problem solving systems [4, 14]. Previous work on caching, however, relied exclusively on caches of potentially infinite size, and, therefore, potentially unlimited overhead cost. Our *bounded-overhead caches* are of a predetermined user-defined size and rely on user-specified cache-management policies to replace less useful cache entries. In [20], we show empirically how such bounded-overhead caches can be quite useful in reducing search without the increasing overhead associated with using more traditional infinite-size caches.

Another speedup technique incorporated in our theorem prover is an *explanation-based learning* (EBL) component based on the EBL* formulation of explanation-based learning algorithms [16]. An EBL* algorithm takes a completed proof as its input and produces as output a macro-operator which can be added to the original domain theory. The addition of a macro-operator biases the order in which the search space (implicitly defined by the domain theory and query) is explored. EBL* algorithms differ from traditional EBL algorithms in the process by which the macro-operator is produced from the input proof. EBL* provides a set of five proof-transformation operators which, when applied to the input proof in some prespecified fashion, result in a new macro-operator. Any extant EBL algorithm can be restated as some particular sequence of EBL* operations. The EBL* framework casts generalization as a heuristic search through the space of transformed explanations; new EBL* algorithms correspond to explicit search-control heuristics for the generalization process.

It is often hard to tell whether a given speedup technique's advantages outweigh the problems it introduces. For example, while the use of EBL may provide some reduction of search, indiscriminate application may also entail some increase in search (the well-known *utility problem* of [13]). To postpone the utility problem somewhat, our prover actively manages the rules acquired via EBL*, disposing of those which do not prove to be useful as time passes. But even without active learned macro-operator management, in [17] we demonstrate how subgoal caching and EBL* together can sidestep the utility problem, in short showing greater strength combined than their individual performance might imply.

Why do EBL* and subgoal caching work so well together? EBL* introduces redundancy in the search space and therefore suffers from the utility problem, which, loosely put, results from backtracking over these redundant paths. Success and failure caching serve to prune redundant search by recognizing the path as either valid or fruitless. Thus caching can work to mitigate the utility problem introduced by EBL* macro-operators, providing the search reduction benefits of the learned rules while avoiding their shortcomings.

5. Dynamic Abstraction Hierarchies

Yet another search reduction technique employed by our planner extends the notion of an assumption further, exploiting the resource allocation mechanism to dynamically form approximate abstraction hierarchies [21]. Previous work in hierarchical planning consists of static techniques which rely on certain characteristics of the problem domain, such as Knoblock's *ordered monotonicity property* [9]. Such techniques cannot be expected to make progress in large complex domain theories, where ordered monotonicity generally does not hold.

There are two ways to look at our use of assumptions. First, making an assumption essentially postpones a subgoal. Thus our treatment of assumptions could be seen as a global antecedent reordering strategy. A second, and perhaps more useful, way of looking at assumptions, however, is as an approximation of hierarchical planning. Consider that once a plan containing explicit assumptions has been produced, the prover is used to verify that the assumptions are in fact warranted; essentially expanding a more abstract plan into a more detailed one.[3] We justify

[3] Additional assumptions might be made in the course of verifying an assumption; these assumptions will in turn require additional verification by the theorem prover. Thus our hierarchies may be many levels deep.

our expectation that PERFLOG assumptions will usually be provable since each assumption corresponds to a frame axiom; the vast majority of relations in the world remain unchanged when any one operator is applied. While we may *expect* an expansion exists, there is no *guarantee* that a valid expansion exists: our hierarchies are only approximate.

We can extend the PERFLOG notion of an assumption even further in order to efficiently build and manage ephemeral, approximate, abstraction hierarchies. The intuition is that any subgoal — and not just the negated antecedent in the cancellation axiom — might make a reasonable assumption if we have some reason to believe the subgoal could could be verified given sufficient resources. We support our belief dynamically using success and failure statistics accumulated by the inference engine in a special-purpose data structure during execution. The data structure, created at rule definition time and maintained by the prover, keeps track of how many successful and unsuccessful attempts have been made for each type of subgoal and how much their eventual proofs or failures cost. Each subgoal is used as an index into this data structure, and, if appropriate conditions are met based on statistics from similar previously encountered subgoals, the subgoal is assumed to be true and its proof is deferred. Our motivation is the same as for traditional hierarchical planning: don't sweat the details until you have an abstract plan. What exactly constitutes a detail, however, is determined on the fly.

6. Distributed Adaptive Inference

Recently, we have begun work on a new implementation of our adaptive theorem prover that is to be configured to run transparently in a heterogeneous distributed environment (*e.g.*, on a network of workstations). The hope is that a distributed system will scale up to larger, more realistic target problems. We have implemented a prototype AND-parallel distributed theorem prover that simulates a network of workstations using only one machine.

Many others have suggested distributed or parallel implementations of PROLOG in the past. Most have proposed implementing OR-parallel systems [2]. The disadvantage of OR parallel search is that its potential payoff is limited: in the best case, the expected elapsed time for a proof cannot be less than the expected elapsed time for a perfect serial search (*i.e.*, a serial search that makes only correct choices). A few have pursued more complex AND-parallel systems, notwithstanding their intrinsic problems [3,8]. Descendents of an AND node may depend on each other since they may share one or more variables: solving sibling subgoals independently may not be sufficient to obtain a proof of their parent AND node. In recompense, the potential payoff is much higher than for OR parallelism, since, in the ideal case, all of the work performed by every processor contributes to the solution.

We are pursuing an unrestricted AND-parallel design, one that does not require compile time determination of subgoal independence nor any sort of restriction on which subgoal can bind a shared variable. What makes our work different from previous approaches is that our speedup techniques distribute cleanly and elegantly. For example, it is not necessary that each processor maintain identical subgoal caches.

Our system exploits the synergy between parallelism and each server's own adaptive bias. Recall that speedup techniques rely on an assumption regarding the future query distribution. Unlike serial applications of speedup techniques, our distributed inference engine has at least some control over the subgoal distribution sent to each individual server. By greedily selecting a mapping of subgoals to proof servers, we can exploit each individual server's bias, essentially training servers as experts in some slice of the domain theory. Subgoals can then be automatically assigned to the server that is expected to satisfy them most quickly. By exploiting the individualized strengths of each processor at the time subgoals are distributed for solution, we hope to further increase the throughput of the inference engine.

7. Discussion

We have briefly described an approach to planning that relies on the use of a special adaptive inference engine. The resulting incremental, approximate, *neoclassical* planner circumvents the problem of scaling by attacking the typically intractable search spaces of real planning domains with a variety of speedup and approximation techniques. Our latest work attempts to extend the system to operate on a loosely-coupled network of processors.

While we have used the PERFLOG formalism as both inspiration and as a convenient descriptive point of departure, we should note that there is nothing intrinsically special about this formalism. Indeed, our adaptive inference engine could be used with any planning formalism, such as, for example, situation calculus or even SNLP [11], even though the use of EBL* would violate systematicity for the latter. In fact, there is nothing special about planning: our adaptive inference engine is equally applicable to any definite-clause domain theory. Nevertheless, our new operationalization of classical planning promises to scale well by attacking the typically intractable search spaces of real planning

domains with a variety of speedup and approximation techniques.

Acknowledgements

Charles Elkan, Geoff Gordon, Alex Russell, and Daniel Scharstein contributed to the research reported here. Support for our research was provided by the Office of Naval Research grant N00014-90-J-1542, and through gifts from the Xerox Corporation and the Hewlett-Packard Company.

References

1. K. Clark, "Negation as Failure," in *Logic and Databases*, H. Gallaire and J. Minker (editors), Plenum Press, NY, 1978, pp. 293-322.

2. S. A. Delgado-Rannauro, "OR-Parallel Logic Computation Models," in *Implementations of Distributed Prolog*, P. Kacsuk and M. J. Wise (editors), John Wiley & Sons, Chichester, 1992, pp. 3-26.

3. S. A. Delgado-Rannauro, "Restricted AND- and AND/OR-Parallel Logic Computational Models," in *Implementations of Distributed Prolog*, P. Kacsuk and M. J. Wise (editors), John Wiley & Sons, Chichester, 1992, pp. 121-141.

4. C. Elkan, "Conspiracy Numbers and Caching for Searching And/Or Trees and Theorem-Proving," *Proc. of the Eleventh Intl. Joint Conf. on AI*, Detroit, MI, Aug. 1989, pp. 341-346.

5. C. Elkan, "Incremental, Approximate Planning," *Proc. of the Natl. Conf. on AI*, Boston, MA, July 1990, pp. 145-150.

6. R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *AI 3* (1972), pp. 251-288.

7. M. Ginsberg and W. Harvey, "Iterative Broadening," *Proc. of the Natl. Conf. on AI*, Boston, MA, July 1990, pp. 216-220.

8. M. V. Hermenegildo, "An Abstract Machine Based Execution Model for Computer Architecture Design and Efficient Implementation of Logic Programs in Parallel," Tech. Rep.-86-20, Ph.D. Thesis, The University of Texas at Austin, Austin, Texas, 1986.

9. C. A. Knoblock, "Learning Abstraction Hierarchies for Problem Solving," *Proc. of the Natl. Conf. on AI*, Boston, MA, July 1990, pp. 923-928.

10. R. Korf, "Depth-First Iterative Deepening: An Optimal Admissible Tree Search," *AI 27*, 1 (1985), pp. 97-109.

11. D. McAllester and D. Rosenblitt, "Systematic Nonlinear Planning," *The Natl. Conf. on AI*, Anaheim, CA, July 1991, pp. 634-639.

12. J. M. McCarthy, "Applications of Circumscription to Formalising Common Sense Knowledge," *Proc. of the Nonmonotonic Reasoning Workshop*, 1984, pp. 151-164.

13. S. Minton, "Quantitative Results Concerning the Utility of Explanation-Based Learning," *Proc. of the Natl. Conf. on AI*, St. Paul, MN, July 1988, pp. 564-569.

14. D. Plaisted, "Non-Horn Clause Logic Programming Without Contrapositives," *Journ. of Automated Reasoning 4* (1988), pp. 287-325.

15. A. M. Segre, *Machine Learning of Robot Assembly Plans*, Kluwer Academic Publishers, Hingham, MA, Mar. 1988.

16. A. M. Segre and C. Elkan, "A Provably Complete Family of EBL Algorithms," Work. Paper, Dept. of Computer Science, Cornell Univ., Ithaca, NY, Nov. 1990.

17. A. M. Segre, "On Combining Multiple Speedup Techniques," *Proc. of the Ninth Intl. Machine Learning Conf.*, Aberdeen, Scotland, July 1992, pp. 400-405.

18. A. M. Segre, C. Elkan, D. Scharstein, G. Gordon and A. Russell, "Adaptive Inference," in *Machine Learning: Induction, Analogy, and Discovery*, S. Chipman and A. Meyrowitz (editors), Kluwer Academic Publishers, Hingham, MA, Jan. 1993.

19. A. M. Segre and J. Turney, "Planning, Acting, and Learning in a Dynamic Domain," in *Machine Learning Methods for Planning*, S. Minton (editor), Morgan Kaufmann Publishers, San Mateo, CA, To appear, Mar. 1993.

20. A. M. Segre and D. Scharstein, "Bounded-Overhead Caching for Definite-Clause Theorem Proving," To appear in: *Journal of Automated Reasoning*, 1993.

21. J. Turney, "DAM: Dynamic, Approximate Abstraction for Resource-Limited Planning," Ph.D. Thesis, Dept. of Computer Science, Cornell Univ., Ithaca, NY, *In preparation*: 1993.