Least-Cost Flaw Repair: A Plan Refinement Strategy for Partial-Order Planning

David Joslin* and **Martha E. Pollack†,***
*Intelligent Systems Program
†Department of Computer Science
University of Pittsburgh, Pittsburgh, PA 15260
joslin@cs.pitt.edu, pollack@cs.pitt.edu

Abstract

We describe the least-cost flaw repair (LCFR) strategy for performing flaw selection during partial-order causal link (POCL) planning. LCFR can be seen as a generalization of Peot and Smith's "Delay Unforced Threats" (DUnf) strategy (Peot & Smith 1993); where DUnf treats threats differently from open conditions, LCFR has a uniform mechanism for handling all flaws. We provide experimental results that demonstrate that the power of DUnf does not come from delaying threat repairs *per se*, but rather from the fact that this delay has the effect of imposing a partial preference for least-cost flaw selection. Our experiments also show that extending this to a complete preference for least-cost selection reduces search-space size even further. We consider the computational overhead of employing LCFR, and discuss techniques for reducing this overhead. In particular, we describe QLCFR, a strategy that reduces computational overhead by approximating repair costs.[1]

Introduction

Current research in plan generation in AI centers on partial-order causal link (POCL) algorithms, which descend from McAllester and Rosenblitt's SNLP algorithm (McAllester & Rosenblitt 1991; Penberthy & Weld 1992; Barrett & Weld 1993; Collins & Pryor 1992; Peot & Smith 1993; Kambhampati 1993). POCL planning involves searching through a space of partial plans, where the successors of a node representing partial plan P are refinements of P. As with any search problem, POCL planning requires effective search control strategies.

In POCL planning, search control has two main components. The first, *node selection*, involves choosing which partial plan to refine next. Most POCL algorithms use best-first search to perform node selection. Once a partial plan has been selected, the planner must then perform *flaw selection*, which involves choosing either a threat to resolve or an open condition to establish. Threats can be resolved by promotion, demotion, or separation; open conditions can be established by adding a new step to the plan or adding a new causal link to an existing step. Unless it is impossible to repair the selected flaw, new nodes representing the possible repairs are added to the search space.

Both the SNLP algorithm and its implementation in the UCPOP system (Penberthy & Weld 1992) adopt a flaw-selection strategy in which threats are resolved before open conditions. However, neither SNLP nor UCPOP specify any principles for selecting which threat or which open condition to repair. Peot and Smith (Peot & Smith 1993) relax the requirement that threats always be resolved before open conditions, and examine several strategies for delaying the resolution of some threats. One of the most effective strategies that they studied is what they call "Delay Unforced Threats" (DUnf.) In DUnf, a threat is selected only if there is only a single way to repair it (or if there is no way to repair it, i.e., it represents a dead end.) Such threats are called "forced." If all the current threats are unforced, i.e., have multiple possible repairs, then an open condition is selected for establishment instead. Peot and Smith do not indicate what happens in the case in which the only remaining flaws are unforced threats, but one must assume that in these cases, some threat is selected.

In this paper, we describe and examine the Least-Cost Flaw Repair (LCFR) strategy, a generalization of DUnf. We define the *repair cost* of any flaw—either threat or open condition—to be the number of nodes generated as possible repairs. LCFR is the strategy of always selecting a flaw with the lowest possible repair cost at a given node. Like DUnf, LCFR will delay any threat that is unforced (repair cost > 1) in favor of a threat that is forced (repair cost $<= 1$.) But by treating all flaws uniformly, LCFR also applies a similar strategy to open conditions, preferring to handle open conditions that are forced over open conditions, or threats, that are not. Similarly, LCFR handles the case in which all that remain are unforced threats: the

[1]This work has been supported by the Air Force Office of Scientific Research (Contract F49620-92-J-0422), by the Rome Laboratory (RL) of the Air Force Material Command and the Advanced Research Projects Agency (Contract F30602-93-C-0038), and by an NSF Young Investigator's Award (IRI-9258392).

LCFR strategy will select a threat with minimal repair cost.

The LCFR strategy is similar to one of the search heuristics used in the O-Plan system (Currie & Tate 1991). The contribution of this paper is to isolate this strategy and examine it in some detail, in order to explain its success and that of the related DUnf strategies.

In the following sections we provide more details about LCFR and its relationship to other flaw-selection strategies, and then describe experiments we conducted to compare the performance of POCL-planners employing these alternative strategies. We then examine the question of secondary selection strategies: what flaw should LCFR select in cases in which there are two or more flaws with minimal repair cost for a given node? We also consider techniques for reducing the computational overhead involved in calculating repair costs. In particular, we describe QLCFR, a strategy that reduces computational overhead by approximating repair costs. The final section discusses directions for future research on LCFR and related flaw-selection strategies.

Comparison of Flaw-Selection Strategies

The original POCL planning algorithms—SNLP and UCPOP—always prefer to repair threats before open conditions. Neither specifies how to select among alternative threats, or among alternative open conditions, although the UCPOP code employs a LIFO mechanism, i.e., it always selects the threat (or open condition if there are no threats) that was most recently introduced into the partial plan. Peot and Smith examine the effects of modifying this strategy to delay the repair of some threats. In particular, one of their most effective strategies, DUnf, will select a threat only if it is forced or if there are no open conditions remaining in the plan. The DUnf strategy does not include a commitment to a particular way to select among open conditions, although Peot and Smith suggest three alternatives: FIFO, LIFO, and "least-commitment." The "least-commitment" strategy selects an open condition with the fewest children, i.e., using the terminology introduced in the previous section, one with minimal repair cost.

Our hypothesis was that the principle of "least-cost" selection ought to be extended to *all* flaws. In other words, the power of the DUnf strategy comes not from the relative ordering of threats and open conditions, but instead from the fact that DUnf has the effect of imposing a partial preference for least-cost flaw selection. DUnf will always prefer a forced threat, which, by definition has a repair cost of at most one; thus, in cases in which there is a forced threat, DUnf will make a low-cost selection. What about cases in which there are no forced threats? Then DUnf will have to select among open conditions, assuming there are any. If our hypothesis is correct, a version of DUnf that makes this selection using a least-cost strategy ought to perform better than a version that uses one of the other strategies. In fact, if it is the selection of low-cost repairs that is causing the search-space reduction, then the idea of treating threat resolution differently from open condition establishment ought to be abandoned. Instead, a strategy that always selects the flaw with minimal repair cost, regardless of whether it is a threat or an open condition, ought to show the best performance. This is the Least-Cost Flaw Repair (LCFR) strategy.

To test our hypothesis, we began with the UCPOP system, and implemented various modifications of it. Two of these—DUnf and Dunf-LCOS—encode Peot and Smith's strategy. Both delay selection of unforced threats until all open conditions have been established, but the former selects among open conditions using a LIFO strategy, like UCPOP, while the latter performs least-cost selection of open conditions. A third modification, LCFR, implements the generalization of the least-cost strategy: it always selects a flaw with minimal cost, without regard to whether that flaw is a threat or an open condition. Finally, we also implemented a variant called LCOS, which, like UCPOP, always selects threats before open conditions, but which uses a least-cost strategy to choose among open conditions. LCOS was included to verify that the state-space reduction results from the preference for flaws with minimal repair costs: if this is true, then LCOS should show a decrease in state-space size even though it does not delay *any* threats. These five flaw-selection strategies are described in algorithmic form in Figure 1.

Experimental Results

The five planners were each tested on 49 problems from a variety of domains. In all the experiments, the node-selection strategy is best-first search, where the heuristic evaluation function is the sum of the number of steps and the number of flaws in the partial plan. These are the defaults provided with UCPOP. We also imposed a search limit of 8000 generated nodes. In reporting our results, we give the number of nodes examined, which is typically less than the number of nodes generated.

The 49 test problems are divided among 15 domains. Table 1 lists the total number of problems attempted for each domain, as well as the number of problems from that domain solved by each planner within the 8000-node limit. All of the problems except those from the TileWorld domain are taken directly from the sample problems distributed with UCPOP version 2.0. Eight miscellaneous domains are grouped together in the last row of the table. All five planners solved the same ten problems in this group.

As Table 1 shows, LCFR solved more problems (44) than any of the other four planners. None of the problems on which LCFR failed were solved by any of the other four planners. Figure 2 plots the percentage of

If any threats exist in the set of flaws
    select a threat
Else select an open condition
    (LIFO for UCPOP; Least-cost for LCOS.)

UCPOP, UCPOP-LCOS

If there are any threats with repair cost = 0
    select a threat from that set
Else if there are any threats with repair cost = 1
    select a threat from that set
Else if there are any open conditions
    select an open condition
    (LIFO for DUnf; Least-cost for Dunf-LCOS)
Else select a threat (unforced.)

DUnf, DUnf-LCOS

Select a flaw, minimizing repair cost.

LCFR

Figure 1: Flaw Selection Strategies

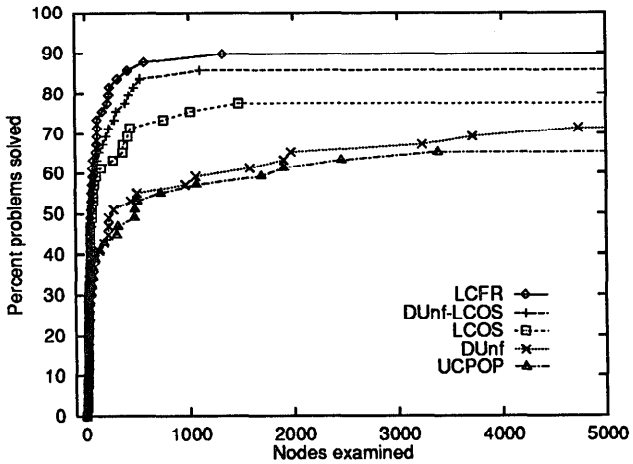

Figure 2: Comparison of planner search spaces

problems solved by each planner within a fixed number of nodes examined. (Each point $\langle x, y \rangle$ denotes that x% of the 49 test problems were solved by examining no more than y nodes.) Table 2 provides summary statistics for the experiment.

Discussion

The experiment described above confirms our original hypotheses. DUnf performs only marginally better than UCPOP: the percentage of problems solved by DUnf within any fixed number of nodes examined is only slightly higher than the percentage solved by UCPOP. On the other hand, DUnf-LCOS, which not only delays unforced threats but also performs least-cost open condition selection, performs significantly better than UCPOP, solving more problems within any fixed number of nodes, and, on average, searching far fewer nodes. Simply delaying unforced threats does not, in and of itself, lead to much improvement, but doing this in combination with a preference for minimal-cost open conditions does.

Our hypothesis that the search-space reduction is primarily due to selection of least-cost flaws is further bolstered by the performance of LCOS. Recall that LCOS does not delay *any* threats; nonetheless, its performance is significantly better than either UCPOP or DUnf.

Finally, note that LCFR, the only algorithm that uniformly selects flaws with minimal repair cost, shows the greatest reduction in search-space size. It solves the most problems overall (44), and it solves more problems than any other planner within any fixed number of nodes. The average number of nodes it examines is significantly less than any of the other planners except DUnf-LCOS.[2] Although LCFR is only marginally better than DUnf-LCOS, it has the advantage of being conceptually simpler in that it provides a uniform treatment of all flaws.

The relative performance of the five planners was not uniform across all the domains. Most notable was the TileWorld domain, consisting of a grid on which holes and tiles are scattered. The agent's goal is to fill one or more holes by picking up and carrying tiles (but carrying no more than four at a time), taking them to holes, and dropping one tile in each hole. LCFR and DUnf-LCOS solved all six of the problems taken from the TileWorld domain, while the other strategies solved at most two. The task of filling two holes is solved by LCFR after generating only 73 partial plans; the same problem was not solved by UCPOP even when allowed to run for over eight hours.

One can readily see the reason for such dramatic differences by looking at just the first few nodes examined in the plan-generation process. There the TileWorld domain is dominated by the establishment of open conditions that vary widely in their repair costs. In almost every case, at least one open condition in a partial plan has repair cost 1; at the same time, there are often open conditions with repair costs as high as 8. This occurs, for example, when an open condition that the agent be holding a tile can be established using any of eight tiles on the grid in the initial state. The planning strategies that do not perform least-cost selection of open conditions (UCPOP and DUnf) often select an open condition with an unnecessarily high repair cost,

[2] Using a paired-sample t test over all 49 problems, the reduction in the number of nodes examined by LCFR over any of the other planners is significant ($p < 0.01$), though for LCFR over DUnf-LCOS the significance is marginal ($p = 0.08$). DUnf-LCOS also shows a significant improvement over LCOS, DUnf and UCPOP ($p < 0.04$). The improvement of DUnf over UCPOP is marginally significant ($p = 0.096$).

Domain	Total Probs	UCPOP	DUnf	LCOS	DUnf-LCOS	LCFR
Briefcase world (Pednault 1988)	8	6	7	8	8	8
Office World (based on (Pednault 1988))	7	6	6	5	5	7
TileWorld (Pollack & Ringuette 1990)	6	0	1	2	6	6
Russell's Tire World (Russell 1992)	6	5	5	5	5	5
Blocks world	5	3	4	5	5	5
Monkeys and Bananas	3	2	2	2	2	2
STRIPS robot world	2	0	0	1	1	1
(Eight misc. domains)	12	10	10	10	10	10
TOTALS	49	32	35	38	42	44

Table 1: Problems solved by each planner, by domain

Planner	Problems each solved					Problems all solved				
		Nodes examined					Nodes examined			
	N	Min	Max	Mean	S. Dev	N	Min	Max	Mean	S. Dev
UCPOP	32	9	3380	444	795	31	9	3380	404	775
DUnf	35	9	4733	628	1142	31	9	3230	378	724
LCOS	38	9	1475	165	303	31	9	1475	157	325
DUnf-LCOS	42	9	1104	130	202	31	9	1104	112	219
LCFR	44	8	1320	114	215	31	8	1320	107	248

Table 2: Statistical comparison of search spaces (successful problems only)

which leads to excessive branching.

LCOS of course avoids the pitfall of poorly choosing an open condition. However, its undoing is its rigid preference for threats over open conditions. In many cases, LCOS prefers a higher-cost threat to a lower-cost open condition: for example, we observed it bypassing open conditions with repair cost 1 for threats with repair costs of 3 or more.

Finally, DUnf-LCOS does just about as well as LCFR: both solve all the TileWorld problems. However, examination of DUnf-LCOS's planning process shows that there are times in which it makes the opposite mistake from LCOS: it prefers higher-cost open conditions to lower-cost unforced threats. Although this appears not to have significantly hurt DUnf-LCOS on the TileWorld problems, it may account for the two problems from other domains on which DUnf-LCOS failed but LCFR was successful, and may suggest a potential problem for other applications.

What this analysis shows is that a uniform preference for least-cost flaws is especially important in domains in which flaw repair costs vary widely.

Secondary Flaw-Selection Strategies

As we have already pointed out, LCFR does not specify a strategy for selecting among the flaws with minimal repair cost. An obvious question is whether the performance of LCFR could be improved by the choice of a secondary flaw-selection strategy that made such decisions.

Before exploring particular secondary strategies, however, we wanted to determine just how sensitive LCFR might be to secondary selection. We therefore conducted a second experiment in which LCFR was run ten times on each of the 49 test problems, selecting flaws randomly from the set of flaws with minimal repair cost. Recall that 44 problems out of 49 were solved successfully in the initial set of experiments. The randomized LCFR solved 42 problems successfully in all ten trials, failed to solve four problems in any of the ten trials, and solved the remaining three problems seven, eight, and nine times, respectively.

Figure 3 shows the mean number of nodes examined for each of the 45 problems that LCFR solved successfully at least once. The error bars show the minimum and maximum number of nodes examined for each problem over the ten trials. The three error bars that are clipped at the top of the graph are those that exceeded the search limit on one or more trials.

We can note that the majority of problems are relatively insensitive to secondary selection. For example, the range of the number of nodes examined (i.e., the difference between the maximum and the minimum) was 100 or less for 30 out of the 45 problems solved at least once. Given the low number of nodes searched by LCFR, on average, in the first experiment, it would be surprising if we had not found this kind of insensitivity to secondary selection.

More interesting is the fact that *all* of these 45 problems were solved at least once by examining a very small number of nodes (415, in the worst case). This is true even for problems for which the mean number of nodes examined over the ten trials is several thousand, including problems for which LCFR sometimes failed. Note further that, although most of the problems showed little variation over the ten trials,

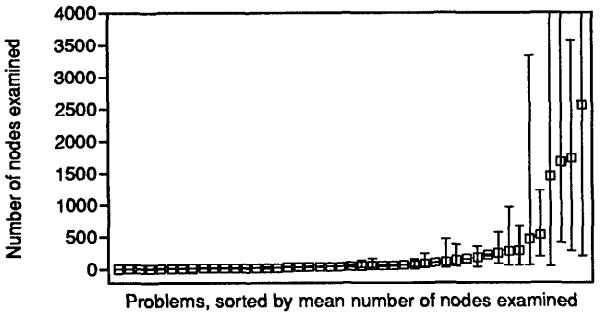

Figure 3: Results of sensitivity experiment

most of those that showed extreme variation had a distinctly bi-modal distribution of the number of nodes examined. Three problems failed on one or more trials, exceeding the search limit of 8000 nodes generated (though examining fewer than that), but succeeded on other trials by searching as few as 50 nodes. One of these problems had seven successful trials, examining a minimum of 211 and a maximum of 648 nodes, while exceeding the search limit on three trials. Other problems had bi-modal distributions even though they succeeded on all ten trials. One problem, for example, succeeded one time after examining over 3000 nodes, and nine times after examining 250 or fewer nodes.

These results suggest that while a more sophisticated secondary selection strategy would not significantly improve LCFR's performance on most of our test problems, it could have a substantial positive effect on those problems that LCFR sometimes found difficult. An obvious candidate for a secondary strategy would be to prefer threat resolution to open-condition establishment, assuming that the repair costs are equal. Our preliminary investigations of this strategy, however, did not show significant improvement. The nature of a good secondary selection strategy remains an open question.

Improving the Performance of LCFR

Although LCFR searches far fewer nodes than UCPOP, it incurs a significant overhead in computing repair costs. Our implementation of LCFR, for example, used less CPU time than UCPOP on only four of the 32 problems on which both were successful. Although LCFR examines far fewer nodes than UCPOP, it spends an average of over forty times as long expanding each node (104 ms vs. 2.4 ms). Clearly, if LCFR is going to live up to its promise of making POCL planning more efficient, then the cost of flaw selection must be significantly decreased.

Fortunately, there are some clear-cut ways to do this. Our implementation of LCFR took advantage of the fact that the repair cost for a flaw could be calculated by allowing UCPOP to make all repairs to that flaw, generating a new node for each repair, and then discarding all the newly generated nodes except those associated with the selected flaw. This approach required a minimum of modification to the UCPOP code, but is obviously inefficient: the repair cost could be calculated without actually allocating the node structures. In addition, there are methods that could be used to reduce the amount of work done in recalculating repair costs for flaws that have already been considered. For example, if a threat was not separable the last time it was considered, there is no way that other changes to the partial plan could have made that threat separable. In recalculating its repair cost, we need only look at promotion and demotion.

Another alternative—and the one we explore in this paper—involves reducing the overhead of flaw selection by accepting some inaccuracy in the repair-cost calculation. One way to do that is to calculate the repair cost of each flaw only once, when that flaw is first encountered. In any successor node, if that flaw has still not been repaired, we assume that its repair cost has not changed. We refer to this variation of the strategy as "Quick LCFR" (QLCFR.) Note that QLCFR will sometimes produce inaccurate repair costs, because it is possible that repairing one flaw will change the repair costs of other flaws, either by eliminating possible repairs, or by adding new options. For example, adding a new step to the plan may add the option of reusing an effect of that new step to satisfy another open condition.

By assuming that repair costs are fixed, QLCFR was able to expand a node in an average of 4.6 ms, a huge reduction from LCFR's 104 ms. Although this is still higher than the average amount of time spent per node by UCPOP, the reduction in search space is now sufficient to allow QLCFR to solve problems on average about twice as fast as UCPOP, for problems that both solve. And QLCFR solves more problems than UCPOP: for our 49 test problems, QLCFR solved 38 problems, compared to 32 for UCPOP. QLCFR solves fewer problems within the node limit than either LCFR (44) or DUnf-LCOS (42), but QLCFR is much faster than either of these.

These results are very encouraging. Even given our currently inefficient method of calculating repair costs, QLCFR executes in time comparable to that of UCPOP, solving more problems and searching fewer nodes on those problems that both solve. A more efficient implementation of the repair cost calculation should reduce the time spent examining each node even further, and more intelligent decisions about when to recalculate repair costs (rather than simply never recalculating), should improve on the accuracy of the repair cost estimate.

Future Research

Given the importance of making plan generation more efficient, techniques such as LCFR and its variants are obviously worth pursuing further. In particular, we see

at least four key areas for further investigation.

First, it is worth examining additional techniques for reducing the overhead of computing repair costs. In the previous section we noted some improvements that could be made to the way in which we implement the repair-cost calculation, and described QLCFR, which approximates repair costs by assuming that they do not change. One plausible extension of QLCFR would involve keeping track of the "age" of a flaw, and recalculating its repair cost only if that age exceeds some threshold. Another possibility would involve scanning the list of flaws within a partial plan until a flaw is found with repair cost at or below a fixed threshold, or until we reach the end of the list, in which case a flaw with minimal repair cost would be selected. With a threshold of zero, this algorithm reduces to a slightly optimized version of LCFR; with a threshold of one or more, the principle of least-cost selection would sometimes be violated, but with a potential savings in computational overhead. Varying the threshold allows one to trade flaw-selection costs for quality of flaw selection fairly directly.

Second, we can consider more sophisticated definitions of "repair cost" than the one we have been using. As we have defined it, the repair cost of a flaw takes into account only the immediate branching factor for a given repair. It may be, however, that the best flaw to repair has branching factor higher than the minimum; consider the simple case in which a flaw with a repair cost of N actually has $N - 1$ descendents that are quickly recognizable as dead ends. It may be that some degree of "look ahead" in the calculation of a repair cost may be advantageous, in spite of the additional computational cost. One such strategy was implemented in O-Plan (Currie & Tate 1991).

Third, it is worth returning to the issue of node selection, and reconsidering heuristic evaluation functions for POCL planning in light of a least-cost flaw selection strategy. An evaluation function that estimates the repair costs for the flaws in each node might be more effective than one that simply treats all flaws equivalently.

Finally, and perhaps most significantly, it is worth considering how to extend the lesson learned from the LCFR experiments—namely, that during POCL planning, it pays to focus first on flaws with minimal repair costs—to develop techniques for *reducing* the repair costs of particular flaws. One possibility would involve the use of a richer representation for temporal ordering constraints. We can think of a causal link as a constraint on a temporal interval whose endpoints are defined by the establishing and consuming steps. We could reduce the repair cost of a threat by replacing promotion and demotion with a single constraint that the threatening step occur at some time "not during" the causal link being threatened. In effect, this means carrying a disjunction (promotion or demotion) that otherwise would be handled by creating two separate nodes. The computational complexity of working with such constraints is greater than that of the simpler ordering constraints used by SNLP or UCPOP, but perhaps not so much greater as to outweigh the benefits of reduced branching in the search space. Similarly, we might look for other techniques that make tradeoffs between reduced repair costs and richer representations of constraints.

The exploration of these and related extensions to search control for POCL planning are left for future research. For now we note that our experimental analyses of LCFR and QLCFR indicate the promise of flaw-selection strategies that focus on the degree of branching caused by repairing a flaw.

References

Barrett, A., and Weld, D. 1993. Partial-order planning: Evaluating possible efficiency gains. To appear in *Artificial Intelligence.*

Collins, G., and Pryor, L. 1992. Achieving the functionality of filter conditions in a partial order planner. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 375–380.

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

Kambhampati, S. 1993. Planning as refinement search: A unified framework for comparative analysis of search space size and performance. To appear in *Artificial Intelligence.*

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 634–639.

Pednault, E. P. D. 1988. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence* 4(4):356–372.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, 103–114.

Peot, M., and Smith, D. E. 1993. Threat-removal strategies for partial-order planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 492–499.

Pollack, M. E., and Ringuette, M. 1990. Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 183–189.

Russell, S. J. 1992. Efficient memory-bounded search algorithms. In *Proceedings of the Tenth European Conference on Artificial Intelligence.*