Computer Science
Technical Report

Characterizing Domain Specific Effects in Flaw Selection for Partial Order Planners[*]

Adele E. Howe   Eric Dahlman
Computer Science Department
Colorado State University
Fort Collins, CO 80523
e-mail: {howe,dahlman}@cs.colostate.edu

June 1997

Technical Report CS-97-112

[*]This research was supported in part by NSF Career Award IRI-9624058 and DARPA contract F30602-95-0257.

Characterizing Domain Specific Effects in Flaw Selection for Partial Order Planners[*]

Adele E. Howe Eric Dahlman
Computer Science Department
Colorado State University
Fort Collins, CO 80523
e-mail: {howe,dahlman}@cs.colostate.edu

June 1997

Abstract

The flaw selection strategy is integral to good performance in a partial order planner. Yet, no flaw selection strategy has been shown to be superior on all problems. Two prominent strategies, ZLIFO and LCFR, perform well on different problems. We studied three domain specific factors: precondition ordering in domain theories, goal ordering in problems and dynamic ordering of flaws. For each factor, we collected data on the performance of ZLIFO, LCFR and related strategies on systematically varied problems. We found that while all strategies are sensitive to these factors, some are more so than others. Moreover, even careful control of domain and problem definition in LIFO strategies cannot produce consistently better performance than Least Cost on all problems. Based on our data, the ultimate flaw selection strategy must be dynamic, rather than relying exclusively on any of the existing strategies.

1 Introduction

As has been shown several times [Joslin and Pollack, 1994], [Srinivasan and Howe, 1995], [Gerevini and Schubert, 1996], [Pollack *et al.*, 1996], the flaw selection strategy is integral to good performance in a partial order planner. A bad flaw selection strategy can make an apparently simple partial order planning problem practically unsolvable. Although some principles of flaw selection strategies have been proven (e.g., effect of delaying separable threats when using FIFO and LIFO strategies [Peot and Smith, 1993]), no proposed solution has been shown superior on all problems; indeed, a general search algorithm for partial order planning may not be possible [Knoblock and Yang, 1995]. In fact, strategies appear quite sensitive to seemingly minor changes in domain and problem definition [Srinivasan and Howe, 1995].

Two strategies, ZLIFO and LCFR, have performed well, but differently on problems. In fact, neither seems to have a consistent edge over the other. The strategies at the core of each are Least Cost and LIFO ordering of plan flaws. Least cost is a dynamic strategy that effectively performs one step lookahead by ordering all flaws according to the number of ways in which they can be repaired. The least cost flaw is the one with the fewest options for repair,

[*]This research was supported in part by NSF Career Award IRI-9624058 and DARPA contract F30602-95-0257.

Name	*Specification*
UCPOP	$\{n\}$ LIFO / $\{o\}$ LIFO / $\{s\}$ LIFO
ZLIFO	$\{n\}$ LIFO / $\{o\}_0$ LIFO / $\{o\}_1$New^{LIFO} / $\{o\}_{2-\infty}$ LIFO / $\{s\}$ LIFO
LCFR	$\{o, n, s\}$LC^{LIFO}
LCFR-L	$\{o, n, s\}$LC^{LIFO*}
LCFR-DS	$\{o, n\}$LC^{FIFO} / $\{s\}$ R
LCOC	$\{n\}$ LIFO / $\{o\}$LC^{LIFO} / $\{s\}$ LIFO
LCOC-F	$\{n\}$ LIFO / $\{o\}$LC^{FIFO} / $\{s\}$ LIFO
Random	$\{n\}$ LIFO / $\{o\}$ R / $\{s\}$ LIFO
ZLIFO-R	$\{n\}$ LIFO / $\{o\}_0$ LIFO / $\{o\}_1$New^{LIFO} / $\{o\}_{2-\infty}$ R / $\{s\}$ LIFO
LCFR-R	$\{o, n, s\}$LC^{R}

Table 1: Flaw selection strategies tested

thus, producing the lowest immediate amount of branching in the plan search space. The calculated cost depends on the current position in the search space, which makes the ordering context dependent and dynamic. LIFO is a static strategy that, with some exceptions in ZLIFO, effectively performs depth first search on the pursuit of subgoals. The ordering is static in that it is determined by the order of goals and preconditions set forth by the programmer.

Intuitively, least cost seems superior because it is responsive to the state of the search space and more effectively exploits problem/domain constraints to prune the search space. However, LIFO based strategies have done better on some of the problems tested previously (e.g., TRAINS2 and the briefcase problems); some have argued that the LIFO approach exploits programmer knowledge better [Williamson and Hanks, 1996]. However, a recent study suggests that ZLIFO's sometimes advantage over LCFR is actually due to its delay of separable threats [Pollack *et al.*, 1996].

Our purpose is to tease apart some of the observable domain specific factors leading to differential performance of the two key flaw ordering strategies: least cost and LIFO. We examined three factors known to affect their performance: precondition ordering in domain theories, goal ordering in problems and context dependent or dynamic ordering of flaws. Our goal is to determine the factors that most influence performance and, whenever possible, how performance can be improved by careful exploitation of those factors.

1.1 Background on Data Collection

To empirically explore domain and problem effects, we tested 31 problems from the UCPOP 4.0.6 standard distribution[1]. The domains were modified to include new precondition orderings. The problems were modified to include new goal orderings.

[1]The problem set was somewhat limited by the strategies explored. Some of them do not currently handle facts.

Overall, we tested 10 different flaw selection strategy algorithms. The LIFO based algorithms were: UCPOP and ZLIFO[Gerevini and Schubert, 1996]; both come with the UCPOP distribution. The LC based algorithms were: three variants on LCFR[2][Joslin and Pollack, 1994] and two variants on LCOC (Least Cost Open Conditions). Like ZLIFO, LCOC addresses non-separable threats first and delays separable threat, making it easier to compare with ZLIFO. The LC versions differ on how equal cost ties are broken; LCFR-L uses a different method of calculating costs which results in a different LIFO ordering. We also included three strategies that substitute random selections at key junctures. The 10 algorithms are listed in Table 1 using notation from [Pollack *et al.*, 1996]. o, n, and s refer to the three types of flaws: open conditions, non-separable (threats that can only be resolved by re-ordering plan steps) and separable threats, respectively. Flaw types are ordered by LIFO, FIFO, R (Random) or New (favor new steps over initial conditions). Subscripts indicate costs; superscripts are tie-breaking orderings. Steps are separated by "/".

2 Effect of Domain Theory Definition

The domain theory consists of the operators that can comprise plans. Operators include parameters, preconditions and effects. Preconditions and parameters that are typed become open conditions in an evolving plan. Conditional effects may also contribute open conditions, but we have disregarded them as they are difficult to manipulate in a controlled manner.

When a new step is added to a plan, the preconditions from its operator are added as open conditions to be resolved. The relative ordering of the preconditions in the operator structure is preserved when they are added to the the flaw list. In LIFO strategies, the first precondition listed is likely to be the first flaw repaired.

Obviously, then precondition ordering is one source of domain specific performance differences. We sought to answer three questions about precondition orderings' contribution to domain specific performance. First, how sensitive are the strategies, particularly LIFO, to the domain theory? Second, LIFO can only be expected to do well if the programmer was knowledgeable, and the default domain ordering facilitative. So how good are the original orderings? Third, we expect even LC based strategies to be somewhat sensitive to domain theory definition because they must include a tie breaking strategy. How sensitive are they?

2.1 Precondition Ordering and LIFO

To test the sensitivity of LIFO strategies, we generated new domain theories in which all operators' precondition orderings were permuted. When the number of permutations was large, we sampled the space in three ways. First, in the flat-tire domain, we permuted only those operators applicable to each problem. Second, we included permutations in which only one operator was permuted. Third, we sampled a small number of cases in which more than one operator was permuted. Even with the sampling, two domains, fridge and strips, were still too large and were not tested.

[2]Massimo Paolucci and Martha Pollack provided the LCFR code.

Problem	Perms	Tested	ZLIFO				UCPOP			
			best	orig.	worst	diff	best	orig.	worst	diff
1. Tower-Invert3	5040	5040	38	39	63	4	55	59	131	6
2. Sussman-Anomaly	5040	5040	31	31	41	4	38	38	94	6
3. Tower-Invert4	5040	5040	76	98	589	6	330	330	1594	6
4. Hanoi-3	40320	17584	250	1128	2364	38	4137	11178	-	115
5. Test-Ferry	69120	4262	88	107	134	35	119	596	2950	545
6. Rat-Insulin	120960	31401	318	338	338	2	871	885	1449	6
7. Prodigy-Suss	138240	1862	223	321	13795	313	236	515	32131	1500
8. Monkey-Test1	165888	11653	53	96	142	4	38	165	-	206
9. Monkey-Test2	165888	60	227	1092	1092	5	228	765	1627	29
10. R-Test1	518400	21690	25	28	30	6	25	37	62	25
11. R-Test2	518400	844	5251	7567	-	38	1276	10748	-	94
12. Fix3	2x10^{2}2	13451	85	125	149	8	502	565	-	299
13. Fixit	2x10^{2}2	1156	6145	20301	-	11	-	-	-	-

Table 2: Effect of precondition ordering on LIFO based strategies as measured in plans created

Table 2 shows the best, original and worst results for those problems in which precondition order mattered. We ran 29 problems. Of them, 12 had four or fewer permutations and showed no effect; four of the fix problems, which involve few operators, showed no effects for ZLIFO and made little difference for UCPOP. *Tested* are the number of possible permutations (*perms*) actually tested. *Diff* counts the distinct values found for plans created. An entry of "-" indicates that it timed out at a search limit of 50,000. The problems have been numbered to save space on later tables.

One hypothesis is that sensitivity should increase with the number of open conditions [Pollack *et al.*, 1996]. Such sensitivity would explain why LIFO does not appear to do as well on larger problems; the more sensitive, the more careful the programmer needs to be. If the hypothesis held, then *diff* should increase with *perms*. However, Table 2 shows that it is more complicated; it depends on subgoal interaction (`r-test2` adds a subgoal to `r-test1`) and coding of state and operators (compare `sussman-anomaly` with `prodigy-sussman`).

As to exploiting programmer knowledge, in all cases, some precondition reordering led to as good or better performance than the programmer's (original) did. We analyzed the data for necessary and sufficient precondition orderings. In most cases, only a few relative orderings matter. Not too surprisingly, preconditions forcing inequality never matter; however, parameter typing does sometimes matter, particularly in domains such as rat-insulin that specialize operators by parameter types. UCPOP is most sensitive to ordering; thus, we found more specific relative orderings for it. At present, we are still trying to determine, in general, what makes for a good ordering.

Prob	LCOC			LCFR		
	best	orig	worst	best	orig	worst
1	37	37	40	41	41	46
2	28	28	29	33	33	36
3	72	84	100	99	99	116
4	5852	9464	13373	-	-	-
5	99	186	244	91	180	229
6	72	59	76	77	81	81
7	90	152	152	90	96	103
8	33	33	37	36	36	50
9	86	765	765	123	143	143
10	25	28	29	25	28	29
11	517	517	1159	865	901	1120
12	81	134	137	130	130	133
13	16532	16953	28528	31674	36420	-

Table 3: Effect on LC strategies of changing precondition order, as measured in plans created

2.2 Precondition Ordering and LC

Strategies that depend less on LIFO should be relatively insensitive to precondition reordering. So ZLIFO, which uses LIFO for a subset of open conditions, should vary less than UCPOP across different precondition ordering; in fact, as Table 2 shows, ZLIFO exhibits less difference between best and worst performance.

The LC based strategies require LIFO as a tie breaking strategy. Previous results [Pollack *et al.*, 1996] showed LCFR-DSep to be less sensitive to precondition ordering than ZLIFO on the `trains2` problem. To determine whether the LC strategies are sensitive to the same orderings as the LIFO strategies, we tested LCFR and LCOC on the original, best and worst precondition orderings for ZLIFO and UCPOP. Table 3 shows that, with the exception of `Hanoi-3`, the LC methods do vary with these changes in precondition orderings, but much less.[3]

3 Effect of Goal Ordering in Problems

The order of goals in problems determines the first step in the search space for all strategies. In LIFO strategies, goals are satisfied, more or less, in the order in which they were defined. Thus, subgoal interactions may result in variable search performance for some strategies. For example, although problems `get-paid3` and `get-paid4` differ only in the order of their goals, the number of plans created for each varies under each flaw selection strategy (see Table 4 for range in `get-paid3`).

As with preconditions, we expect all strategies to some extent to be susceptible to changes in goal ordering. It starts off search and leads it further at critical junctures. To determine how

[3]Most of the LCFR results do not agree with the results published in [Pollack *et al.*, 1996]; while we are using the same basic Lisp code, we are using a different version of UCPOP.

Problem	UCPOP			ZLIFO			LCOC			LCFR		
	best	orig	worst	best	orig	worst	best	orig	worst	best	orig	worst
2	28	38	38	27	31	31	28	31	31	30	30	31
3	105	330	330	98	98	98	77	84	84	99	99	124
4	11178	11178	-	805	1128	1128	9464	9464	9674	-	-	-
6	885	885	3055	338	338	518		76			81	
7	515	515	981	115	321	321	96	96	126	96	96	110
9	765	765	816	138	1092	1092		86		143		
11		10748		7506	7567	7567	961	964	964	900	901	901
12	565	565	951	92	125	697	112	112	136	105	105	132
13	4233	-	-	665	20301	-	14287	16953	-	11752	-	-
14 Ho-Demo	71	71	75	74	74	103		74			80	
15 Get-Pd	15	20	47	15	21	27	15	20	24	15	20	24
16 Get-Pd2	32	76	244	31	31	33	32	32	33	44	51	51
17 Uget-Pd2	43	115	282		42		43	50	50		58	
18 Get-Pd3	97	326	384	69	69	72	215	245	245	633	2291	2291
19 Uget-Pd3	185	1585	1585	109	109	121	259	275	275	27436	-	-

Table 4: Effect of goal ordering on strategies, measured in plans created.

much effect, we permuted the goal orderings and ran the four basic strategies on each problem. Some problems were already goal permutations of others; some exhibited little or no effect. As the results for the remaining problems show (Table 4), as with precondition ordering, more reliance on LIFO does, in general, lead to more susceptibility to ordering effects. The strategies can be ordered by the number of problems for which order matters and by the average difference between best and worst values (420 for UCPOP, 171 for ZLIFO, 24 for LCOC and 8 for LCFR, if `get-paid3` is exempted).

Goal re-ordering produced better performance than the original in nine, eight seven and five of the problems for UCPOP, ZLIFO, LCOC and LCFR, respectively. In the cases in the table, the best ordering with ZLIFO produces the lowest plans created in nine cases, LCOC in five, LCFR in three, and UCPOP in two. Thus, many problems would be solved more quickly if the goals could be re-ordered automatically. What makes for good orderings? To determine this, we analyzed the best orderings for the two LIFO strategies as they are the most affected by goal ordering. In all cases, a single ordering was best for UCPOP, while multiple orderings were equivalently good for ZLIFO due to its separating out zero and one cost open conditions. The "winning combinations" for ZLIFO proved to be two relative orders with the first most important: the goals that have initial cost of 1 (can only be resolved by adding actions) and the rest. In the cases examined, the goals should be ordered by their required path length through the plan; in other words, those goals that require more actions in order to be solved should go first. For the cost one goals, this order can be easily computed by solving for these goals individually and counting the number of actions in the resulting plan. For the other goals, we have yet to determine an easy method for calculating the order.

For example, the fixit problem is fairly complicated (the final plan contains 19 steps) and showed dramatic improvement for the LIFO strategies. The best ordering for ZLIFO was [7, 3, 6] for cost one and [5, {4, 2}, 0, 1] for the rest where the original goal ordering was [0, 1, 2, 3, 4, 5, 6, 7]. While not a complete reversal of the original, it demonstrates how crippled a LIFO strategy can be by poor ordering. The best ordering for UCPOP was [6, 3, 0, 4, 5, 7, 2, 1]. For the LC strategies, the good orderings for UCPOP and ZLIFO were easier for LCOC and LCFR to solve than the original ordering; in fact, LCFR's best goal ordering was the UCPOP ordering. We have not yet uncovered a pattern to the winning orderings for the LC methods, which is not surprising, given the relative complexity of these methods.

4 Effect of Dynamic Versus Static Flaw Selection Strategies

LCFR is a dynamic strategy that determines for a given point in search what next flaw will minimize the immediate branching. This strategy exploits information available at that point in the search; unfortunately, the ordering based on cost is not always right. *Iterative sampling* [Langley, 1992] follows paths randomly until a solution is found; this strategy is effective when the solution density is high. A contributor to the success of LCFR may be its dynamic nature. To help determine whether a dynamic strategy might not be best, we substituted a random choice at key points in the algorithms and compared their best and worst performance on the standard problem set. If one of the current strategies is best, then adding random choices should not improve on the previous solutions.

The three random algorithms are listed in Table 1. *Random* is UCPOP/LCOC in which an open condition is selected at random for repair; *ZLIFO-R* substitutes a random choice only for open conditions of cost more than one; *LCFR-R* breaks cost ties using a random selection. We expect that Random will produce the largest spread between best and worst. For every problem, each random strategy was executed 500 times.

The results in Table 5 show that we can often do better with a dynamic rather than fixed strategy. In every problem, the best of the random found at least as good a solution as one of the non-random strategies. In fix4 and fix5, every solution performed the same. In seven simple problems not included in the table, some strategy did as well as the best of the random. In a few cases (e.g., `monkey-test2` and `Hanoi3`), the random versions found dramatically better solutions.

We cannot say whether one random strategy is better than the other; due to the sampling, we cannot be sure whether any random algorithm might not find the same solution. However, we can point out that Random reached the search limit more and had the largest spread between best and worst (counting only worst results that are not at the search limit). The average spread was 2932 for LCFR-R, 5967 for ZLIFO-R and 5968 for Random. However, Random also found the most best solutions: ten for Random, four for ZLIFO-R and six for LCFR-R. In general, all strategies at their worst still solved over 90% of their trials; the two exceptions were `Hanoi3` in which LCFR-R solved 65%, ZLIFO-R solved 47%, and Random solved 18% and `Fixit` in which LCFR-R solved 56%, ZLIFO-R solved 12% and Random solved 4%.

Prob	LCFR-R		ZLIFO-R		Random	
	Min	Max	Min	Max	Min	Max
1	36	304	**35**	2553	43	2310
3	**67**	2024	72	-	74	-
4	331	-	**203**	-	2386	-
5	84	1374	57	1617	**56**	45012
6	**43**	271	82	6150	49	5170
7	**82**	571	255	-	208	-
8	34	50	37	219	**30**	1043
9	**62**	-	73	3969	71	7406
11	**133**	4226	196	-	190	-
12	**76**	181	78	40249	100	-
13	5600	-	**894**	-	15008	-
14	34	98	74	103	**28**	150
16	32	239	**31**	357	**31**	266
17	43	755	42	882	**38**	211
18	69	31637	48	2514	**40**	2197
Get4	90	-	50	2042	**39**	1300
19	101	-	72	7378	**55**	2571
Uget4	96	-	73	4377	**53**	6814
Road	28	34	25	40	**24**	233
Fixa	58	68	**55**	210	64	-

Table 5: Results of random choice at key points in three algorithms, as measured in plans created

5 Conclusions

Good flaw selection in partial order planning is known to be domain sensitive. In this paper, we have studied how three factors influence the performance of two core flaw selection strategies, LIFO and LC, as incorporated in well known algorithms. Our long term goal is to identify facets of problems that can be detected either before or during planning that will help determine the best strategy to apply.

5.1 Implications of the Study for Evaluation

All of the problems studied are extremely simple by almost any standard. Yet, the performance of every one of the strategies varied, sometimes widely, for each of the factors examined. As it happened, the original encoding of the problems did not consistently favor any strategy. The difference in performance is addressed in the standard problem set in a few cases, in which different goal orderings are already included (e.g., the get-paid and uget-paid problem sets). In comparing strategies, we need to be aware of possible bias in the precondition and goal orderings in problems.

As the study shows, the distributed problem set is not necessarily the best for comparison. The orderings are not usually the "best" for any strategy and sometimes are quite poor. The problem set was compiled to provide examples, but has become a benchmark of sorts. Additionally, many of the domains are quite similar at their core. For example, the briefcase world, ferry-domain and monkey-domain require planning movements using a single resource (the briefcase, boat and monkey) thus, we should expect similar performance on similar problems; the problems do differ in a few key aspects: the test-ferry problem does not direct the final placement of the boat and the briefcase can carry multiple objects. To mitigate the apparent differences, we will be developing a problem generator that will be based on our analysis of the standard problem set and the problems that have been added in other analyses (e.g., TRAINS, Tileworld, TruckWorld).

5.2 What is Best When?

Can clever reordering of preconditions and goals for LIFO strategies beat a least cost strategy? The previous sections describe factors that strongly influence the performance of LIFO strategies. The testing of random tie breaking in LCFR suggests that the tie-breaking strategy may be a major influence on LC performance. We compiled the previous results allowing the best precondition or goal ordering to represent the ZLIFO strategy and collected results for the three versions of LCFR and two of LCOC allowing the best of those to represent the LC based method. Table 6 includes those problems in which the best differed by more than 10%. Even with carefully designed domain theories, ZLIFO outperforms the LC based strategies in only half the cases. Moreover, by modifying the tie breaking strategy, the LC based methods improved performance on the Uget-paid problems, which were difficult for them previously. These results emphasize the role of dynamic strategies and the need to carefully design the tie-breaking strategies for the LC based methods.

On problems that are difficult for LCFR, LCOC often does better and sometimes as good as ZLIFO; this supports the hypothesis that the advantage of ZLIFO may be due primarily to

Prob	ZLIFO	LCOC	LCFR	Best?
3	76	84	81	ZLIFO
4	250	9464	8715	ZLIFO
5	88	140	186	ZLIFO
6	318	61	59	LCOC
7	115	96	96	LCFR
8	53	33	35	LCOC
9	138	123	86	LCOC
11	5251	517	353	LCFR-L
13	665	16953	16073	ZLIFO
14	74	74	51	LCFR-F
16	31	32	32	ZLIFO
17	42	46	49	ZLIFO
18	69	117	191	ZLIFO
19	109	138	102	LCFR-L
20	72	132	200	ZLIFO
21	121	143	109	LCFR-L
23	545	60	60	LC

Table 6: The lowest plans created for the strategies

its delaying separable and forcing non-separable threats rather than its underlying LIFO strategy. The ordering sensitivity of the LC based strategies indicates the importance of carefully designing tie breaking rules. The success of injecting random selection into problems suggests that a dynamic strategy that can adapt to the domain is the best solution.

The results presented here are suggestive and require considerable future work. Although we have found key patterns in the preconditions and goals, we have not yet identified rules for re-ordering the preconditions and goals for best performance. As the last table shows, no tie-breaking strategy held a clear advantage; the role of tie breaking needs to be further explored. We collected traces of the random selections; we will be analyzing these traces to determine: critical choice points, an alternative cost heuristic based on the choices that were shown to be best, and cases in which the choices matched any of the current strategies. We will use these results to design a new adaptive search algorithm that can recognize the domain specific feature and adapt the search strategy to them.

References

[Gerevini and Schubert, 1996] Gerevini, A. and Schubert, L. 1996. Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research* 5:95–137.

[Joslin and Pollack, 1994] Joslin, David and Pollack, Martha E. 1994. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings 12th National Conference on Artificial Intelligence (AAAI-94)*, volume 2. AAAI, AAAI Press/MIT Press. 1004–1015.

[Knoblock and Yang, 1995] Knoblock, Craig A. and Yang, Qiang 1995. Relating performance of partial-order planning algorithms to domain features. *SIGART Bulletin* 6(1).

[Langley, 1992] Langley, Pat 1992. Systematic and nonsystematic search strategies. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems.* Morgan Kaufmann Publishers, Inc. 145–152.

[Peot and Smith, 1993] Peot, Mark A. and Smith, David E. 1993. Threat-removal strategies for partial-order planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Menlo Park. AAAI, AAAI Press/MIT Press. 492–499.

[Pollack *et al.*, 1996] Pollack, Martha; Joslin, David; Paolucci, Massimo; and DeLeon, Yazmine 1996. Flaw selection strategies for partial-order planning. Technical Report 96-20, Univ. of Pittsburgh, Computer Science Dept.

[Srinivasan and Howe, 1995] Srinivasan, Raghavan and Howe, Adele E. 1995. Comparison of methods for improving search efficiency in a partial-order planner. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, CA. 1620–1626.

[Williamson and Hanks, 1996] Williamson, Mike and Hanks, Steve 1996. Flaw selection strategies for value-directed planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems.* 237–244.