To appear in : Proceedings of the Association for Computing Machinery
National Conference ACM 74, San Diego, California, 1974.

TR-29

SKELETON PLANNING SPACES FOR NON-NUMERIC HEURISTIC OPTIMIZATION*

L. Siklóssy & M. A. Haecker

Computer Sciences Department
University of Texas
Austin, Texas, 78712, U.S.A.

Keywords: Problem-Solving, Heuristic Optimization, Robotics, LAWALY, Planning.

ABSTRACT

The AFTERMATH system implements a heuristic technique for improving long solutions (up to about 250 steps) for robot planning problems. AFTERMATH transforms the given solution into a skeleton solution that focuses attention on repetitious and opposite moves. AFTERMATH attempts to obtain an alternate, improved skeleton. From the alternate skeleton, an alternate solution is built (if possible) to the original problem. If the alternate solution is an improvement, AFTERMATH accepts it as input, and cycles.

Although not guaranteeing optimality, AFTERMATH improves many solutions, sometimes gradually in several cycles. Examples can be built for which AFTERMATH obtains an arbitrarily large improvement in one cycle.

1. INTRODUCTION

In the past few years, problem-solving in simulated worlds of robots has attracted much interest [1-11]. Two robot problem-solving systems, LAWALY [6] and AMINA [8] have routinely solved tasks requiring hundreds of steps, thereby showing the feasibility of programs to overcome successfully the challenge of finding solutions to long problems in enormous search spaces resulting from the combinatorial explosion. If the number of steps in the solution is considered, the solutions found by LAWALY or AMINA were optimal only for very short solutions (of the order of ten steps). Longer solutions were almost never optimal. Here, we present a system of programs, called AFTERMATH, which acts as a postprocessor to any robot problem-solver, whether human or machine, and attempts to optimize solutions to long robot tasks.

The basic mechanism that moves AFTERMATH is simple: AFTERMATH tries to detect inefficiencies in the solution that is proposed to it. The inefficiencies could result from the robot making some move, and sometime later an opposite move, when perhaps neither the original move nor its opposite is really necessary. Another possible inefficiency could result from the robot's making the same move again and again along its solution path, when perhaps the move need not be made quite so often.

To detect these interesting repetitious or opposite moves, AFTERMATH transforms the solution into what has been termed a "planning space" [12] in which (essentially) all non-interesting moves are ignored. In other words, from the original solution a skeleton solution of interesting moves is built. AFTERMATH tries to find an alternate skeleton solution to the problem. When such an alternate skeleton solution is found, AFTERMATH attempts to "lift up" this skeleton solution into an alternate full solution to the original problem. If the alternate full solution is an improvement over the original solution, it becomes a new input to AFTERMATH. If it is not, or a full solution could not be lifted up, AFTERMATH backtracks to find another alternate skeleton solution. The number of skeleton solutions is finite, and AFTERMATH will stop when no further alternate skeleton solution is found, unless, of course, it has previously exhausted its resources of time or computer memory.

The interest in AFTERMATH lies not only in the fact that it does indeed improve solutions to robot paths, but also in the use of skeleton solutions as an intermediary problem space in which the important aspects of the inefficiencies of the solution become apparent and can be readily removed (at least many times). In removing inefficiencies, AFTERMATH uses a variety of heuristics that can be justified both on plausible grounds --they are based on common sense observations-- and on grounds of efficacy --they work! These heuristics are of a non-numeric nature: AFTERMATH makes no use of statistics, and its arithmetic capabilities are limited to some elementary counting and comparisons on the integers.

2. THE PROBLEM SPACE.

We briefly review the axiomatization, first introduced in [1], which defines the types of robot problems which AFTERMATH tries to optimize. More complete axiomatizations were proposed in [10] and [11].

The world is described by a set of true facts, for example (see Figure 1): (INROOM BROOM ROOMB) (OPEN DOORAB) (INROOM ROBOT ROOMB) (STATUS LS1 OFF), etc. The world can change if an operator can be

*Partially supported by grant GJ-34736 from the National Science Foundation.

applied to it. For example, the robot can goto an object. The goto(object) operator can only be applied to a world which satisfies the conditions of the operator. One axiomatization of goto would have conditions: (ONFLOOR) (INROOM ROBOT place) (INROOM object place), indicating that the robot must be on the floor, and that she and the object must be in the same room.

When an operator is applied, the world is changed into a new world by use of the add set and delete set of the operator:
new-world := ((old-world - delete-set-of-op) + add-set-of-op), where "-" and "+" are set difference and union.
An axiomatization of the add-set and delete-set of goto could be:
Delete-set: (AT ROBOT $) (NEXTTO ROBOT $);
where "$" is a universal variable that will match anything.

A task is specified as a set GOAL of conditions that the world must satisfy. A particular task, whose solution AFTERMATH tries to improve, was defined by the GOAL set:
(NEXTTO ROBOT LIGHTSWITCH2) (NEXTTO DUSTPAN BROOM) (WATERED TERRARIUMF) (INROOM BOX1 ROOMA) (INROOM BOXO ROOMF) (NEXTTO BOX4 DISPOSAL) (STATUS TABLE DUSTED) (NEXTTO BOX2 BOX3).
Clearly, the solution cannot be solved in the order given: the robot moves around to accomplish her work, and cannot first go to LIGHTSWITCH2 and stay there! Therefore, not all 8! permutations of the subgoals in the above GOAL are possible. But, among the many possible ones, some will have shorter solutions than others.

In general, AFTERMATH was applied to the problems of a robot janitor. Housecleaning and janitorial tasks undoubtedly represent the largest, and most neglected, industry in the world in terms of the man-hours spent on the job!

3. AN EXAMPLE.

A major problem in describing the behavior of AFTERMATH originates from the problems on which it works. These are fairly long solutions to robot tasks. Just writing down a hundred-step solution would consume close to a page! Hence, we shall choose a small problem which, although it does not do jutice to AFTERMATH's capabilities, illustrates some of its main features. AFTERMATH is described in detail in [13].

Figure 1 represents the initial configuration of the world. The goal is:
(NEXTTO BOX2 BOX3) (STATUS LS1 ON). LS1 is the lightswitch in room D (abbreviated RMD). DRCD is the abbreviation of the door joining rooms C, RMC, and D, RMD. The robot needs to climb on BOXO to turn on a light. The first solution input to AFTERMATH is S1.* (Operators are numbered for the reader's convenience.)

*S1 may have been obtained by some problem solver, perhaps human. The source of the solution is immaterial to AFTERMATH. In many cases, the solutions were output by LAWALY.

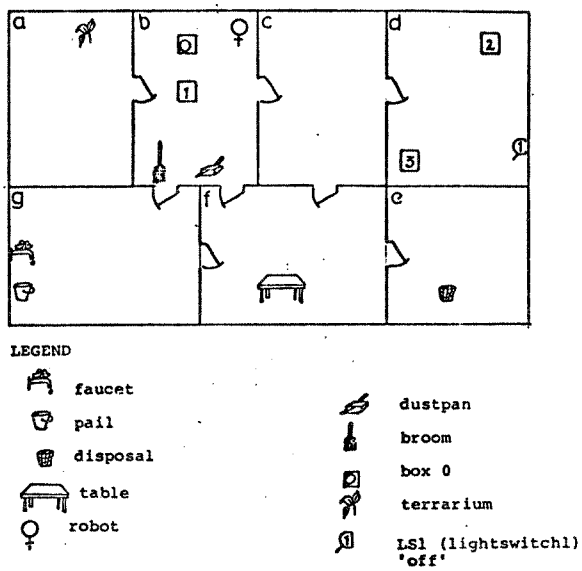

Figure 1. A Robot World

S1: 1-goto(DRBC), 2-gothrudr(DRBC RMB RMC), 3-goto(DRCD), 4-gothrudr(DRCD RMC RMD), 5-goto(BOX2), 6-pushto(BOX2 BOX3), 7-goto(DRCD), 8-gothrudr(DRCD RMD RMC), 9-goto(DRBC), 10-gothrudr(DRBC RMC RMB), 11-goto(BOXO), 12-pushto(BOXO DRBC), 13-pushthru(BOXO DRBC RMB RMC), 14-pushto(BOXO DRCD), 15-pushthru(BOXO DRCD RMC RMD), 16-pushto(BOXO LS1), 17-climbonbox (BOXO), 18-turnonlight(LS1).

3.1 Starred Operators.

AFTERMATH determines which operator last achieved each of the subgoals in the goal. In S1, 6-pushto (BOX2 BOX3) achieves the subgoal (NEXTTO BOX2 BOX3) and will be called *1. The last operator 18-turnonlight(LS1) is *2.

3.2 Interesting Operators.

Besides starred operators, AFTERMATH will only keep those operators which are repetitious, or have opposites. Similar operators, such as gothrudr(DRab RMb RMa) and pushthru(b DRab RMb RMa) are merged. (The information on similar and opposite operators is an input to AFTERMATH, although it is conceivable that it could be generated by the system.)

In S1, the following are the interesting operators:
X : 2-gothrudr(DRBC RMB RMC), 13-pushthru(BOXO DRBC RMB RMC); its opposite X' : 10-gothrudr(DRBC RMC RMB).
Y : 4-gothrudr(DRCD RMC RMD), 15-pushthru(BOXO DRCD RMC RMD); its opposite Y' : 8-gothrudr(DRCD RMD RMC).
(Note: Whether 4 is Y or Y', and 8 is Y' or Y is irrelevant.)

3.3 The skeleton solution.

Eliminating all but starred and interesting operators, we obtain the skeleton solution SS1 : (X Y *1 Y' X' X Y *2)

3.4 Neighborhoods of Starred Operators.

The neighborhood of a starred operator includes all interesting operators left and right of it, up to the next starred operator. (Two starred operators which are not separated by any interesting operator are considered as a single starred operator.) The neighborhood of the last starred operator consists of the preceding interesting operator only. Thus, in SS1, the two neighborhoods are : (X Y *1 Y' X' X Y) and (Y *2).

3.5 Vicinities of Starred Operators.

To achieve a subgoal, typically a robot will have to set up a starred operator, then afterwards undo some of the moves it took to set up the operator in order to achieve some other subgoal. The vicinity of a starred operator is an estimate of the moves needed to set up, and then undo the set-up, of a starred operator. It is obtained from the neighborhood by going right to left and finding the first interesting operator which has an inverse on the left of the starred operator (if several, the leftmost token is chosen.) In case of failure, the immediate left and right neighbors of the starred operator constitute its vincinity.

In SS1, the vicinities are : (X Y *1 Y' X') and (Y *2).

3.6 Matching Vicinities.

AFTERMATH orders all vicinities from the longest to the shortest. An attempt is made to rearrange the vicinities one inside the other to reduce the number of interesting operators in the skeleton solution. In the match-process, various heuristics are used; they are exemplified in Appendix A.

In SS1, the Y of the vicinity of *2 can be identified with the Y of the vicinity of *1, and we obtain the base vicinity (X Y *2 *1 Y' X'). No other choices are possible in this case.

3.7 Lifting of the Alternate Skeleton Solution.

After matching vicinities, the base vicinity shows an alternate order for the starred operators, here *2 *1, which avoids a repetition of a Y operator. AFTERMATH will now try to construct a solution which generates the subgoals corresponding to the starred operators in the order in which the starred operators occur in the final base vicinity.

In general, numbering the subgoals from 1 to n in the solution, the original solution is: *1, *2, ... , *n. An alternate solution suggested by the base vicinity will be *p(1), *p(2), ..., *p(n), where p is a permutation of (1, 2, ..., n). AFTERMATH calls a problem solver with the sequence of tasks: from the original state, achieve *p(1), resulting in state st(1). From state st(1), achieve *p(2), resulting in state st(2), but without ever destroying *p(1). More generally, from state st(j), achieve *p(j+1), resulting in state st(j+1), but without destroying *p(1), *p(2), ..., *p(j). The process stops when j=n-1, or when a subgoal cannot be reached within the constraints. We call a path from st(j) to st(j+1) a step solution. The alternate full solution to the original problem will be the concatenation of all the step solutions. If the length of the alternate full solution is shorter than the original solution, an improved solution was found by AFTERMATH.

AFTERMATH needs to use a robot problem solver which can find step solutions. Moreover, the step solutions given should be optimal, or close to optimal; otherwise the cure may be worse than the disease! The STRIPS problem solver finds non-optimal solutions even to very simple tasks [1], and therefore should be disqualified. On the other hand, LAWALY [6] does qualify, and was used exclusively as the auxiliary problem solver in AFTERMATH.

In the example under consideration, reordering the tasks does indeed result in a shorter solution S1' of 11, instead of 18, steps.
S1' : goto(BOX0), pushto(BOX0 DRBC), pushthru(BOX0 DRBC RMB RMC), pushto(BOX0 DRCD), pushthru(BOX0 DRCD RMC RMD), pushto(BOX0 LS1), climbonbox(BOX0), turnonlight(SL1), climboffbox(BOX0), goto(BOX2), pushto(BOX2 BOX3).

The example is unsatisfactory, since it might imply that AFTERMATH simply generates all other permutations of the goal set. This is not the case. In fact, very few of the permutations are generated in the skeleton space, and sometimes none at all. Appendix A shows a more realistic generation of a skeleton solution.

4. STRENGTHS AND WEAKNESSES OF AFTERMATH.

4.1 Strengths.

AFTERMATH is quite successful at rearranging subgoals in such a way that locally grouped subgoals are accomplished together. Locally grouped subgoals are subgoals which need not be separated by interesting operators. In the robot world that we considered, locally grouped tasks would be accomplished in the same room. In other worlds, the concept of locally grouped subgoals may refer to subgoals performed in the same house, street, city, country, or planet!

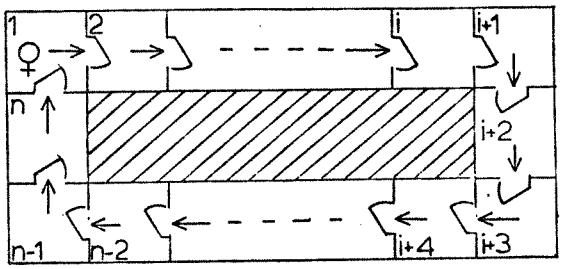

Figure 2. The Ring House

A particularly strong showing of AFTERMATH occurs in the world of Figure 2. The N rooms form a ring, and the robot must accomplish a task Ti in each room i. If the robot must move clockwise, and decides to accomplish the tasks in the (worst possible) order TN, T(N-1), ..., T1, the solution path would be:

$a_{1,2}a_{2,3}\cdots a_{N-1,N}$ *1 $a_{N,1}a_{1,2}\cdots$

$a_{N-2,N-1}$ *2 $a_{N-1,N}\ldots\ldots a_{1,2}$ *(N-1)

$a_{2,3}\ldots a_{N,1}$ *N

where *j accomplishes T(N-j+1), and $a_{j,k}$ represents going from room j to room k.

The vicinities are : ($a_{j,j+1}$ *(N-j) $a_{j+1,\ j+2}$) for j=1 to N-1 (with 1 replacing N+1), and (a_{N-1} *N). The vicinities are ordered from that of *1 to that of *N. The first match (of the first two vicinities) gives the base vicinity:

$a_{N-2,\ N-1}$ *2 $a_{N-1,\ N}$ *1 $a_{N,1}$

with no alternative. The final base vicinity will be unique:

$a_{1,2}$ *(N-1) $a_{2,3}\cdots a_{N-1,N}$ *1 $a_{N,1}$ *N, (no alternative).

The new ordering of starred operators suggests solving the tasks in the order T2, T3, ..., TN, T1, since *j corresponds to T(N-j+1).

The original solution requires N^2 steps (counting Ti as one) while the new solution requires 2N steps. With N sufficiently large, the improvement is as close to 1 (or 100%) as one wishes. However, the new solution is not optimal. The optimal solution would require solving tasks in the order T1, T2, ..., TN, and has a length of (2N - 1). AFTERMATH finds no additional skeleton when it cycles on its first improved solution, and hence does not find the optimal solution to the problem.

The above example illustrates AFTERMATH's capabilities in accomplishing a task on its way to another task. The skeleton solution makes it easy to recognize that a task may be inserted in a tentative partial skeleton solution. For example, (see Appendix A) if the vicinity (Z' *7) is to be inserted in the base vicinity (X' *1 *3 X Z *2 Z' X') an obvious choice would result in a new base vicinity (X' *1 *3 X Z *2 Z' *7 X') which would indicate that *7 could possibly be accomplished on the way out from *2.

4.2 Weaknesses.

Perhaps the major weakness of AFTERMATH is that the optimal solution need not be found. There may be several reasons for the failure to optimize totally and each of these has in fact shown up in actual test cases: the lack of interesting operators, the elimination of optimal paths during cycling, and the lack of direction for long tasks.

4.2.1 Lack of interesting operators

In the improved solution to the ring problem (section 4.1) there are no repeated steps, hence no interesting operators. Similar situations occur when access to a room occurs through different doors. AFTERMATH may not recognize the various different accesses as similar.

4.2.2 Elimination of Optimal Paths during Cycling.

If an initial solution S1 is improved with a non-optimal solution S2, AFTERMATH does not return to alternate ways of improving S1. Instead, it concentrates its efforts on improving S2. If S2 cannot be improved by AFTERMATH's heuristic methods, the optimal solution will not be found. It could be that an alternate skeleton solution of S1 would have led to an optimal solution. But this alternate skeleton was eliminated as soon as S2 was found to be an improvement over S1.

The above two types of weakness might be remedied by making the AFTERMATH heuristics less sharp, and therefore filtering out far fewer candidate alternate skeleton solutions. Although the weaknesses might be occasionally removed, the performance of AFTERMATH would be considerably degraded overall as the third type of weakness indicates.

4.2.3 Lack of Direction for Long Tasks

For long problems (about 200 steps), AFTERMATH tends to generate far too many alternate skeleton solutions, each of which must be lifted to an actual solution. It appears that in these cases and unlike shorter problems, AFTERMATH lacks direction as to the most promising alternate solutions. A broadening of its heuristics, as mentioned at the end of the above section, would only make matters worse.

5. RESULTS.

Table 1 shows the performance of AFTERMATH on fourteen problems. In general, we have no firm knowledge about the optimality of the final solution reached by AFTERMATH. Breadth-first search would be a way to find a shortest path, but an unfeasible one; and human patience is rapidly exhausted on such long problems! The times, in seconds, refer to a compiled implementation in LISP on the CDC 6600. Of particular interest are the successive improvements by AFTERMATH, as it takes as input an improved solution that it had generated. (See problems 9, 10 and 13 in particular.)

As was mentioned in section 4.2.3, the performance of AFTERMATH was weaker on long problems. Problems with initial solutions of 172 to 259 steps were improved (to 160 and 241 steps), with no improvements over 12%, and rarely any natural termination of AFTERMATH. Needless to say, the cost of experimenting with such long solutions is prohibitive!

TABLE 1

problem #	initial no. of steps	final no. of steps	initial solution longer by	iterations successes	iterations total	total time (s)	finished?	breakdown*
1.	17	15	13%	1	1	10	yes	(1-15)
2.	18	18	0%	0	1	5	yes	
3.	24	24	0%	0	1	8	yes	
4.	36	31	16%	2	15	104	yes	(1-33)(3-31)
5.	36	36	0%	0	6	150	no	
6.	40	34	18%	2	12	56	yes	(9-39)(11-34)
7.	51	39	31%	2	3	50	yes	(1-47)(2-39)
8.	66	53	25%	2	4	85	yes	(1-59)(4-53)
9.	79	61	30%	5	6	132	yes	(1-74)(2-70)(3-67) (4-64)(6-61)
10.	87	61	43%	6	15	199	yes	(1-83)(2-79)(5-77) (11-67)(13-63)(14-61)
11.	93	78	19%	1	8	208	no	(1-78)
12.	105	35	200%	1	1	27	yes	(1-35)
13.	107	86	24%	5	9	300	no	(1-99)(4-98)(5-94) (6-92)(7-86)
14.	108	91	19%	3	11	247	no	(1-105)(2-100)(4-91)

*In the breakdown column, the terms describe the successful iterations by (i-j) where i is the iteration number and j is the number of steps the solution was reduced to. Times: compiled LISP on CDC 6600.

6. CONCLUSIONS.

The skeleton solution extracted by AFTERMATH from the solution to a robot problem may be considered as the structure of the solution. AFTERMATH manipulates, and changes this structure to a new structure which will, hopefully, correspond to an improved solution to the initial problem. Hence, AFTERMATH can be viewed as a structural optimizer, and should be contrasted to other techniques of optimization which rely more heavily on computation.

The particular skeletons that are extracted, and the heuristics used in optimizing the skeleton, were developed in the framework of housecleaning robots. The justification of the heuristics lies in their efficacy. AFTERMATH does indeed improve many solutions. Other heuristics may be more powerful in the world that we have chosen, and yet other worlds may require quite different heuristics. Nevertheless, the basic ideas in the design of this structural optimizer should find applications in many areas.

7. ACKNOWLEDGMENT

We are grateful to J. Dreussi who participated in the development of LAWALY, and who helped to modify her so as to make her find step solutions and interact with AFTERMATH.

8. REFERENCES.

1. Fikes, R. E. and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, 2, 3/4, 189-208, 1971.

2. Fikes, R. E., Hart, P. E. and Nilsson, N. J., "Learning and Executing Generalized Robot Plans," Artificial Intelligence, 3, 4, 251-288, 1972.

3. Fikes, R. E., Hart, P. E. and Nilsson, N. J., "New Directions in Robot Problem Solving," Machine Intelligence 7, Michie, D. and Meltzer, B. (Eds.), Edinburgh University Press, Edinburgh University Press, Edinburgh, Great Britain, 1972.

4. Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces," Third International Joint Conference on Artificial Intelligence, Palo Alto, California, 1973.

5. Siklóssy, L., Modelled Exploration by Robot. Technical Report TR-1, Computer Sciences Department; University of Texas, Austin, 1972.

6. Siklóssy, L. and Dreussi, J., "An Efficient Robot Planner which generates its own Procedures," Third International Joint Conference on Artificial Intelligence, Palo Alto, California, 1973.

7. Siklóssy, L. and Roach, J., "Proving the Impossible is Impossible is Possible: Disproofs based on Hereditary Partitions," Third International Joint Conference on Artificial Intelligence, Palo Alto, CA, 1973.

8. Siklóssy, L. and Dreussi, J., Simulation of Executing Robots in Uncertain Environments, National Computer Conference and Exposition, Chicago, Illinois, 1974.

9. Siklóssy, L. and Roach, J. Model Verification and Improvement using DISPROVER. Technical Report TR-26, University of Texas, Austin, July 1973.

10. Hendrix, G., Beyond Omnipotent Robots, Technical Report N1-14, University of Texas, Austin, 1973. (To appear under the title "Modeling Simultaneous Actions and Continuous Processes," in Artificial Intelligence Journal.)

11. Siklóssy, L. and Dreussi, J., Robot problem-solving with negative goals, Technical Report TR-23, University of Texas, Austin, 1973.

12. Newell, A., Shaw, J. C., and Simon, H. A. Report on a general problem-solving program for a computer. Information Processing, Proc. Internat. Conf. Inform. Processing, pp. 256-264. Paris: UNESCO, 1959.

13. Haecker, M. A. An Algorithm for the Heuristic Optimization of Modeled Robot Solution Paths. Thesis. Computer Sciences Dept. University of Texas, Austin, November 1973.

14. Siklóssy, L. and Roach, J. Collaborative Problem-Solving between Optimistic and Pessimistic Problem-Solvers. IFIP Congress 74, Stockholm, Sweden, 1974.

9. APPENDIX A. AN EXAMPLE OF VICINITY MATCHING.

In this example we describe some of the heuristics used in building up base vicinities. The original solution was mapped into the skeleton solution (X Y Y' X' *1 X Z *2 Z' X' *3 X W *4 *5 W' *6 W Z' *7). The neighborhoods of the starred operators are: (X Y Y' X' *1 X Z), (X Z *2 Z' X'), (Z' X' *3 X W), (X W *4 *5 W'), (W' *6 W Z'), (W Z' *7). The vicinities are: (X' *1 X), (X Z *2 Z' X'), (X' *3 X), (W *4 *5 W'), (W' *6 W), (Z' *7). The vicinities are ranked from "strongest" to "weakest", according to length. Ties are broken by preferring vicinities that are bound by opposite operators. Further ties are broken by preferring starred operators with smaller indices, i.e. *3 to *6. The above vicinities would be ranked in the vicinity list as 1-(X Z *2 Z' X'), 2-(W *4 *5 W'), 3-(X' *1 X), 4-(X' *3 X), 5-(W' *6 W), 6-(Z' *7), where numeric labels are inserted for the convenience of the reader.

The initial base vicinity is the first vicinity list. Since it is the longest vicinity, it holds the greatest promise for the successful insertion of other vicinities within its bounds.

First base vicinity: (X Z *2 Z' X').
-Insert 2-(W *4 *5 W'). Fail. Save vicinity 2.
-Insert 3-(X' *1 X). Two choices result as possible new base vicinities: (X' *1 X Z *2 Z' X') and (X Z *2 Z' X' *1 X). A shorter base vicinity would always be preferred. However, here the two candidate base vicinities have the same length, and the tie is broken in favor of the first base vicinity since it more closely preserves the original ordering of subgoals. If necessary, backtracking can return to the second choice, in this as in other cases where several possibilities for insertion are present.
-Insert 4-(X' *3 X). The new base vicinity preferred is (X' *1 *3 X Z *2 Z' X'). It is shorter than the choice obtained by identifying the last X' of the base vicinity with the X' of the 4-th vicinity. The combination *1 *3 is always chosen over *3 *1 since it is closer to the ordering in the initial solution.
-Insert 5-(W' *6 W). Fail. Save vicinity 5.
-Insert 6-(Z' *7). The new base vicinity is (X' *1 *3 X Z *2 Z' *7 X'), since it is shorter. Returning to previous failures, we insert vicinity 2. Insertions could be at the front or at the end of the base vicinity. The end insertion is preferred, since its order is closer to the initial ordering of subgoals. We obtain: (X' *1 *3 X Z *2 Z' *7 X' W *4 *5 W'). Of the two choices for inserting 5-(W' *6 W), the resultant vicinity list (X' *1 *3 X Z *2 Z' *7 X' W' *6 W *4 *5 W') is preferred since the vicinity 5 was inserted inside the previous base vicinity. The desire to maintain the boundaries of the base vicinity overrides the fact that the ordering of subtasks in the alternate insertion is closer to the original solution.

From the new skeleton solution, a new tentative ordering for the subtasks is suggested: (*1 *3 *2 *7 *6 *4 *5). AFTERMATH will make repeated calls to LAWALY to try to lift up this suggested solution. (In practice, the tentative ordering is compared to previous full solutions, and common initial overlaps in the solutions are exploited to save computer time.)