Local-Search Heuristics for Generative Planning

Alexander Nareyek*

GMD FIRST, Kekuléstr. 7, D - 12489 Berlin, Germany

Tel: (+49 177) 792 37 36
Fax: (+49 30) 6392 1805

alex@ai-center.com
http://www.ai-center.com/home/alex/

Abstract. Local-search approaches promise very interesting features for planning, such as an anytime computation and an uncomplicated handling of the environment's dynamics. However, most local-search approaches require the availability of maximal structures what causes problems of scalability. Local search within a generative planning process is rarely looked into. This article presents some basic heuristics for generative planning, and demonstrates the approaches' capabilities by way of solving variations of combined planning and scheduling in the Logistics Domain.

1 Introduction

In contrast to refinement search, which is a stepwise narrowing process alternating between a commitment to specific plan features and a propagation of these decisions to prune inconsistent/incompatible future decision alternatives, local search approaches perform a search by iteratively changing fully grounded plans. In each iteration, a *successor choice criterion* determines a plan which will become the new plan. The potential successor plans that can be chosen for a plan are referred to as the plan's *neighborhood*. The quality of the neighborhood plans can be computed by an objective/cost function.

Most approaches of local search to planning do not change the plan structure during search. Instead, maximal plan structures are used, which include all possible plan options. During search, specific elements are activated or deactivated. Examples are approaches based on Graphplan [4] or SAT [7, 8], in which complete structures involving all possible alternatives are constructed up to a maximal plan length. These maximal structures scale *very* badly because all possible options must be included. Moreover, they can hardly be adapted in case of dynamic real-time environments. To overcome these deficiencies, generative methods can be applied, which dynamically change the plan structure during search, e.g., by adding or deleting actions.

* This work is supported by the German Research Foundation (DFG), NICOSIO, Cross Platform Research Germany (CPR) and Conitec Datensysteme GmbH

Section 2 introduces the model structures that will be used in the following. The neighborhood and search guidance are discussed in Section 3, followed by an empirical analysis in Section 4. The conclusion is given in Section 5.

2 Model Structures

In this section, a brief overview of the applied planning model is given. Only the basic elements are described, as far as they are necessary for illustrating the local-search heuristics. Details can be found in [10].

The model focuses on resources. A resource consists of a temporal projection of a specific property's state (also called *state variable* or *fluent*), and a set of internal constraints such as possible values and possible state transitions. A resource is also subject to further constraints that are induced by the plan such as preconditions and state changes. Numerical as well as symbolic properties are uniformly treated as resources. For example, a battery's POWER and the state of a DOOR are resources:

Resource POWER:
Internal constraints:
$$\{\ \textsc{Power}(t) \in [0..100]\ ,\ \mathrm{abs}(\textsc{Power}(t)\ \text{-}\ \textsc{Power}(t{+}1)) < 10\ \}$$
Current projection:
$$\textsc{Power}(t) = \begin{cases} 0 & : \quad t \in [0..5] \\ 10 - 0.75 \times t & : \quad t \in [6..13] \\ 0 & : \quad t \in [14..\infty[\end{cases}$$

Resource DOOR:
Internal constraints:
$$\{\ \textsc{Door}(t) \in \{\ \textsc{Open}, \textsc{Closed}, \textsc{Locked}, \textsc{Unknown}\}\ \}$$
Current projection:
$$\textsc{Door}(t) = \begin{cases} \textsc{Open} & : \quad t \in [0..45] \\ \textsc{Closed} & : \quad t \in [46..60] \\ \textsc{Unknown} & : \quad t \in [61..\infty[\end{cases}$$

Figure 1 shows an example that illustrates the planning model's basic elements. An action (e.g., eating a peanut) consists of a set of different preconditions that must be satisfied (e.g., that the agent has a peanut), operations that must be performed (e.g., with the mouth) and resulting state changes (e.g., the agent's hunger being satisfied). These elements are represented by *tasks*, i.e., there are PRECONDITION TASKs for precondition tests, ACTION TASKs for operations and STATE TASKs for state changes. The tasks are collections of variables, e.g., temporal variables that determine the beginning and end of the tasks, or variables that specify how a state's properties are changed by the action.

All tasks are assigned to resources — ACTION TASKs to ACTION RESOURCEs, and PRECONDITION TASKs and STATE TASKs to STATE RESOURCEs.

The validity of a plan is checked via *constraints*. A constraint is a test that checks a specific relation between certain variables or other model elements. For example, a constraint EQUALS can check the equality between two variables. If

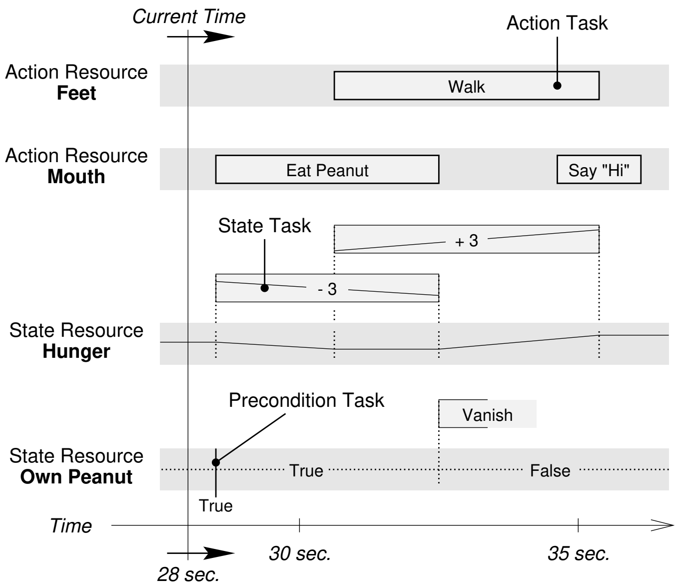

Fig. 1. Basic Plan Elements

all of a problem's constraints are fulfilled, the model structures constitute a valid solution to the problem. There are three basic types of constraints:

– An Action Resource Constraint (ARC) checks for an Action Resource if there is enough capacity to carry out the operations, i.e., that the tasks of the resource do not overlap (e.g., that the mouth does not eat and speak at the same time).
– A State Resource Constraint (SRC) checks for a State Resource if the Precondition Tasks of the resource are satisfied by the states that are deduced from the temporal projection of the resource's State Tasks (e.g., that the 'true' Precondition Task of the action to eat a peanut is met).
– A Task Constraint (TC) represents action requirements by specifying a relation between a set of Precondition Tasks, Action Tasks and/or State Tasks. The constraint is satisfied if the tasks represent a valid action, i.e., if the correct tasks are involved and if the tasks fulfill certain restrictions (e.g., that the Precondition Task and the Action Task of the action to eat a peanut begin at the same time).

When an 'action' is mentioned in the following, this equals a Task Constraint and the tasks associated with it.

3 Neighborhood & Search Guidance

The potential neighbors of a plan are given by a set of heuristics, which are described in the following. The conceptual approach is a modular one, in which the improvement heuristics are integrated into the constraints, such that the constraints dispose of measures to lower their inconsistency on their own (see [11] for details). Due to space restrictions, we will only give rough outlines of the techniques.

In contrast to applications like scheduling, in which the heuristics only need to change values of variables, the planning domain also requires generative elements such as an addition or deletion of actions (of course, not so for approaches that apply maximal structures).

Usually, the constraints will be enriched with further domain-dependent heuristics to exploit domain-specific knowledge. This is an extremely important feature to handle real-world domains. However, we want to show that the approach can even handle general planning tasks, and we will restrict the following to domain-independent planning heuristics.

On top of the constraints, there must be a mechanism that determines which constraint is allowed to select the successor state for a current local-search iteration. This is the job of the so-called *global search control*. For the evaluation of the following section, the selection is done in a very simple way by choosing always the constraint with the highest inconsistency.

3.1 Action Resource Constraint

Violations of an ARC are overlapping ACTION TASKs. To facilitate the handling of inconsistencies, a list of intervals is maintained. The intervals split the time from time 0 to the planning horizon into maximal parts such that for each interval, there is no change in the task assignment for its duration. Task overlaps are mapped as inconsistencies to the respective intervals, the inconsistency value being computed by the number of tasks - 1, multiplied by the interval's length. The ARC's total inconsistency is the sum of the intervals' inconsistencies.

ARC-H1: An obvious repair step is to move one of the tasks that cause an overlap to another position in time. Thus, the basic repair heuristic selects an inconsistent interval, and chooses one of the interval's tasks at random. The task is shifted to the beginning of an interval, the choice probability for the interval being dependent on the task-number improvement with respect to the task's current interval.

ARC-H2: Tasks should obviously be packed quite tightly on a resource such that other tasks can be more easily moved or new tasks be inserted. The heuristic ARC-H2 supports this feature. ARC-H2 is very similar to ARC-H1, but it makes the selection of the second interval in a different way: for the new position of the task, only two shifts are possible, the task either beginning at the beginning of the task's predecessor interval or ending at the end of the task's successor interval. The interval with less tasks is chosen.

ARC-H3: Deleting an action that includes a task causing an overlap is another option. The heuristic ARC-H3 selects an ACTION TASK, the choice probability for an ACTION TASK being proportional to the length of all ARC intervals that include the task and have overlaps. The TASK CONSTRAINT of the selected ACTION TASK is deleted (together with its tasks).

For the evaluation below, the ARC has a static probability distribution of 90 % for choosing ARC-H2, 9 % for ARC-H1 and 1 % for ARC-H3. This distribution is based on job-shop scheduling experiments [11] and was empirically proved, in this context too, to be superior to other ratios.

3.2 State Resource Constraint

We presume a State Resource Constraint with a symbolic state domain and a finite number of possible states. Again, a list of temporal intervals forms the constraint's basic structure. For every linked State Task, a new interval is started. In addition, a state-distance table is maintained, which provides a distance value for every possible transition from one state to another. The distance value is given by the number of actions that need to be executed to realize the transition (using a special value for impossible transitions).

If a Precondition Task's check for a state fails, the state distance from the interval's state to the state requested by the task is added as an inconsistency to the interval of the task. The SRC's total inconsistency is the sum of the intervals' inconsistencies.

SRC-H1: This heuristic tries to add an action that changes the SRC's state in such a way that a Precondition Task becomes less inconsistent. One of the SRC's inconsistent Precondition Tasks is selected. Then, a predecessor interval is chosen, for which an action will be added to reduce the inconsistency of the task. Within this interval, a time point is chosen such that the interference with other tasks is minimized. One state is chosen from among all the states that can be derived by a state transition from the interval's state, a choice probability for a state being proportional to $\frac{1}{d^3}$, where d is the state's state-distance to the state required by the Precondition Task. The action that causes the selected state is the action to be added.

SRC-H2: This heuristic tries to temporally shift a State Task such that a Precondition Task becomes less inconsistent. One of the SRC's inconsistent Precondition Tasks is selected. For the inconsistency improvement of the chosen Precondition Task, a move of the Precondition Task interval's State Task to the previous State Task's interval or to the following State Task's interval is considered. The resulting states at the Precondition Task's time point are computed for both options, and a choice is made with a choice probability for an option that is proportional to $\frac{1}{d^3}$, where d is the state distance from the option's state to the state requested by the Precondition Task, minus the state distance from the state of the task's interval to that requested by the Precondition Task. The time point to which the task is shifted within the selected interval is chosen such that the interference with other tasks is minimized.

SRC-H3: The idea of this heuristic is to temporally shift a Precondition Task such that its inconsistency is decreased. One of the non-goal-related Precondition Tasks is chosen. The new places considered for the chosen Precondition Task are in the task's predecessor interval or successor interval. From these interval options, an interval is chosen with a choice probability for an interval that is proportional to $\frac{1}{d^3}$, where d is the state distance from the interval's state to the state required by the PRECONDITION TASK. The task is moved to the middle of the chosen interval.

SRC-H4: This heuristic tries to improve the inconsistency of a PRECONDITION TASK by deleting a STATE TASK preceding it. One of the SRC's inconsistent PRECONDITION TASKs is selected, and the TASK CONSTRAINT of the interval's STATE TASK is deleted (together with its tasks).

SRC-H5: The idea of this heuristic is to delete an action if it is very hard or impossible to fulfill its precondition. One of the non-goal-related PRECONDITION TASKs is chosen, and the task's TASK CONSTRAINT is deleted (together with its tasks).

The SRC's selection scheme of a heuristic involves a simple kind of learning (see [13] for details). For all five decision alternatives (SRC-H1 to SRC-H5), there is a constraint-internal preference value that represents the preference for this alternative. If a constraint is called to effect an improvement of the plan, it increases an alternative's preference value by one if this alternative was chosen by the constraint's last improvement decision and the constraint's inconsistency value is now better than the last time the constraint was called. If the inconsistency value has deteriorated or is the same as last time, the preference value is decreased by two. Each time there is a consecutive deterioration, the decrease is doubled. However, no preference value can fall below one. Then, the alternative with the highest preference value is chosen.

3.3 Task Constraint

A relation between an action's tasks involves links to two task variables V_1 and V_2, a constant c and a comparator $\bowtie \in \{<, =, >\}$, such that $V_1 \bowtie V_2 + c$. The inconsistency of a relation is given by the minimal shift distance for one of the variables required to satisfy the relation. The TC's total inconsistency is the sum of the relations' inconsistencies.

TC-H1: The TC's heuristic selects an inconsistent relation with a choice probability for a relation that is proportional to its inconsistency. One of the involved variables is selected randomly, and a minimal shift of this variable is performed such that the relation is fulfilled. Then, a recursive repair of all the constraint's relations whose inconsistency has been changed by the improvement is performed (considering only variables that have not already been changed within the improvement step).

4 Empirical Evaluation

Previous experiments (see [12]) have shown that the planning framework can easily be tuned to search in a domain-specific way, e.g., by specialized heuristics. This section sets out to demonstrate that the system is also able to tackle domain-independent tasks and produces decent results even with the simple general heuristics described above. Variations of the Logistics Domain are used for this.

The Logistics Domain specifies the problem of transporting packages between locations. Transportation within a city can be done by a truck. But a truck cannot leave its city. Transportation between locations in different cities must thus be done by airplane. An airplane can only serve one specific location of a city (the airport). A truck and an airplane have actions to load a package, drive/fly to a specific location and unload a package. A problem specification includes a set of locations with information about a location's city and whether the location is an airport, the initial positions of all existing trucks, airports and packages, and the target positions for (not necessarily all) packages.

4.1 Realization

Every package, truck and airplane has an Action Resource, which prevents multiple operations from being executed by an object at the same time (like a truck being driven and simultaneously being loaded with a package). Every package, truck and airplane also has a State Resource, which defines its location. The initial locations are set as the initial states of the State Resources at time 0, and the packages' destinations are specified by a set of corresponding goal-related Precondition Tasks that cannot be manipulated by the heuristics.

4.2 Satisfaction

The benchmark problems of the AIPS-2000 planning competition (track 2) [1] are used in the following. Because of the stochastic nature of the planning system, 1,000 test runs were executed per problem setting. This means that it was only possible to analyze small problems (problems 4 to 10 with variations 0 and 1) because of limited computing power. Instead of computation time, the number of iterations was measured, because the test runs were executed on different types of computers (the average being about 400 iterations per second). A test run was stopped if it failed to find a solution after 100,000 iterations.

Figures 2 and 3 show the number of iterations necessary to find a solution as run-time distributions[1]. The ordering of the problem's solving difficulty equals to that of the planners used in the AIPS competition.

The time taken to solve a problem is closely related to the number of actions necessary to solve the problem (see Figure 4 for the problems of Figure 2). The "steps" in the graphs can be explained by the fact that it does not normally harm a plan to add an action and then add another action to undo the preceding one, e.g., to load a package and then unload it, whereas the addition of a single action can result in a different outcome. Thus, more test runs will yield results with two additional steps instead of one.

[1] It is clearly evident that a restart technique could be applied to greatly improve average-case behavior.

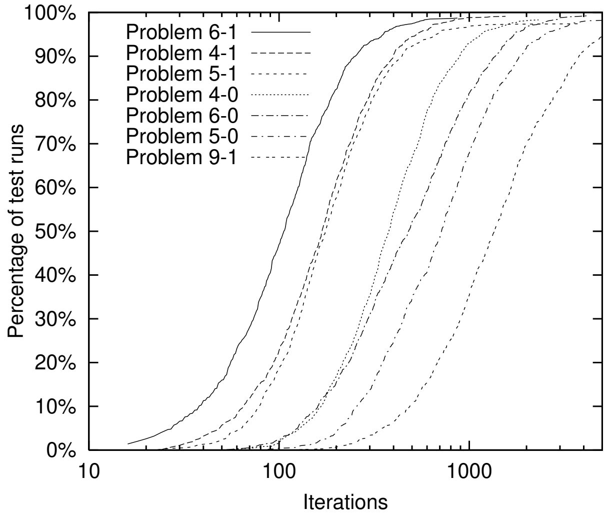

Fig. 2. Runtime Results for the Logistics Domain (I)

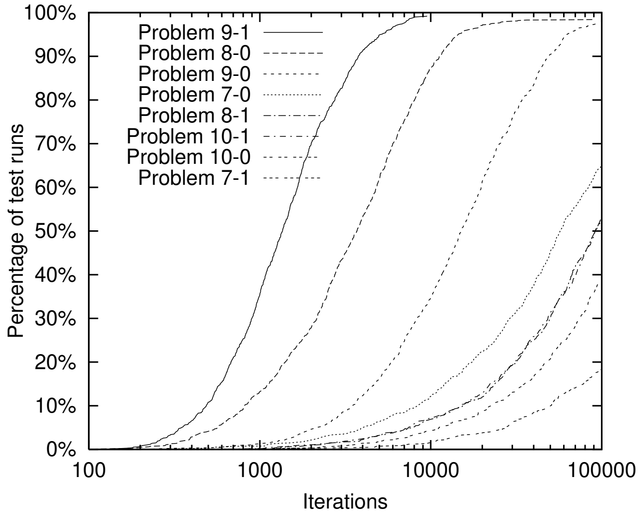

Fig. 3. Runtime Results for the Logistics Domain (II)

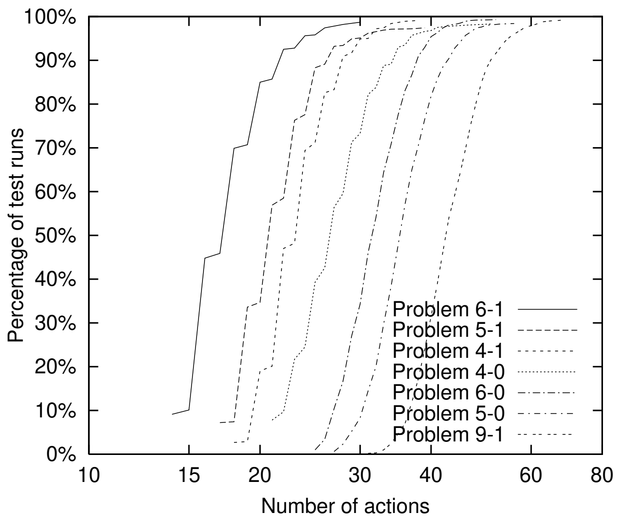

Fig. 4. Plan-Step Results for the Logistics Domain (I)

4.3 Durations & Optimization

The planning system inherently provides a temporal projection. Thus, enhancing the actions by adding a duration is no problem at all. Indeed, the previous problems were already treated as temporal planning problems — each action having a length of one time unit. Of course, this can be changed, and the following test runs involve three different modifications of Problem 6-1, in which every Action Task has a duration of 100 plus a random value of between -99 and +100.

So far, only satisfaction goals have been considered. However, our system could — using *domain knowledge* — actually construct a plan in *linear time* by adding the necessary actions for one package after the other. Even using a very unsophisticated approach, 12 actions at the most would be necessary per package: a truck is driven to the package's location, the package is loaded on the truck, the truck is driven to the airport, the package is unloaded, an airplane is flown to the airport, the package is loaded on the airplane, the airplane is flown to the destination city, the package is unloaded, a truck is driven to the airport, the package is loaded on the truck, the truck is driven to the package's destination, and the package is unloaded.

Thus, enriching the problem by a minimization of the total delivery time (i.e., minimizing the maximal completion time of the single deliveries) seems to be very reasonable. However, moving from a satisfaction to an optimization task produces very different problem characteristics because possible interactions of the single deliveries become very important now.

To implement the minimization, a goal function can be incorporated into the SRC that returns the duration from 0 to the goal-related Precondition

TASKS. An improvement heuristic for this goal function can be implemented very simply: it shifts the PRECONDITION TASKS in the direction of time point 0 until the first time point is reached at which the SRC's inconsistency increases.

The question now is the balance between satisfaction and optimization. Here, the global search control takes a hierarchical approach, i.e., optimization of the goal function is only started if the cost function reaches 0.

The initial time point for the goal-related PRECONDITION TASKS, i.e., a kind of maximal horizon, is not really important as long as it allows enough time for a consistent solution. For the following experiments, the consistency is guaranteed by setting it to $NumberOfPackages \times MaximallyNecessaryActionsPerPackage \times MaximalActionLength$, i.e., for the 6-1 modifications, $6 \times 12 \times 200 = 14400$. Figure 5 presents three sample test runs for Problem 6-1a, in which the development of the duration of consistent solutions can be seen. The runtime was set to 10,000 iterations for these test runs. The dips indicate an application of the SRC's goal heuristic, while the plateaus indicate a search for a new consistent solution.

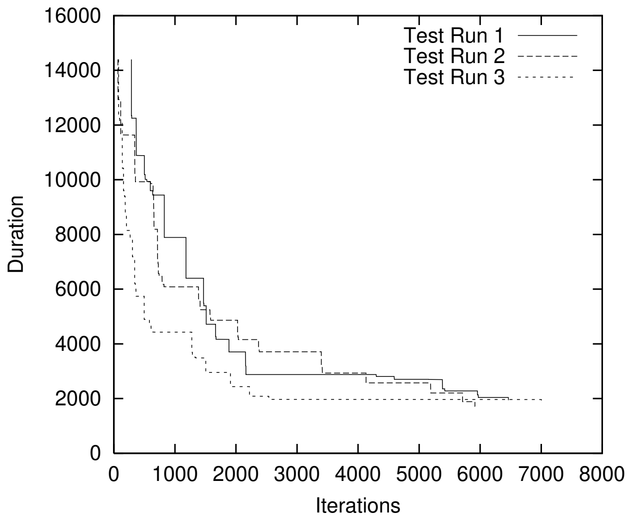

Fig. 5. Three Sample Test Runs for Problem 6-1a

For Problems 6-1a, 6-1b and 6-1c, Figure 6 shows how many test runs found a certain duration for a runtime of 100,000 iterations. The shortest durations found are 1,150 for 6-1a, 1,004 for 6-1b and 1,177 6-1c.

A possibility to improve the result is to apply techniques to leave local minima, such as tabu lists [5]. A tabu list can store the objects affected by changes (e.g., an ACTION TASK moved by ARC-H2). If no new object is stored in the list within an iteration (or in the case of unsuccessful application of an SRC's heuristic), an empty element is stored in the list to guarantee an ongoing replace-

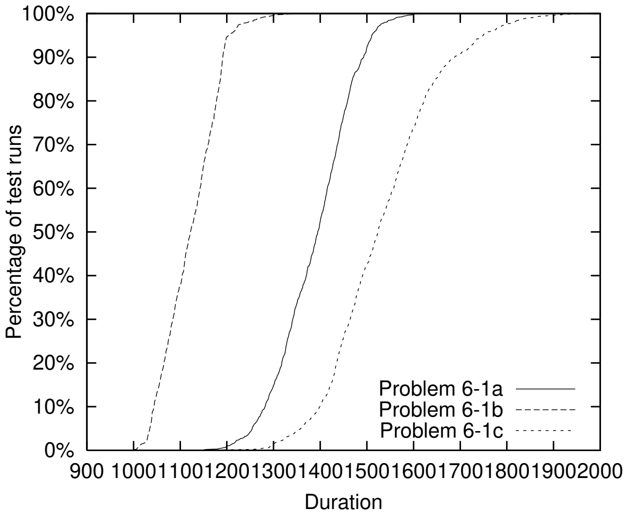

Fig. 6. Durations Found for Problems 6-1a, 6-1b and 6-1c

ment of the elements. A heuristic that tries to modify one of the list's objects fails to do so. Figure 7 shows (for Problem 6-1a) that short tabu lists can yield a significant improvement.

Many other "standard" methods for optimizing local search could be tried, but this is not our main concern here.

4.4 Dynamics

To test the approach's ability to handle dynamics, a random package is canceled every 1,000 iterations (no matter if it is already on the way or has been delivered) and a new package is inserted (its initial location and destination being decided at random). The horizon is expanded by $MaximallyNecessaryActionsPerPackage \times MaximalActionLength + 1$, i.e., for the 6-1 modifications, $12 \times 200 + 1 = 2401$, to grant the existence of a satisfiable plan. To give an example, Figure 8 shows the results between the eighth change and the ninth change. It contrasts the dynamic adaptation with a strategy that recomputes the solution after a change from scratch. It is clear to see that the continuous planning approach yields much better results and can quickly recover after the changes.

5 Conclusion

It was shown that a planning system based on generative local-search techniques is capable of tackling domain-independent planning tasks, even though the underlying techniques were specifically designed to promote a search guided by domain-specific knowledge. The approach was shown to be very appropriate for

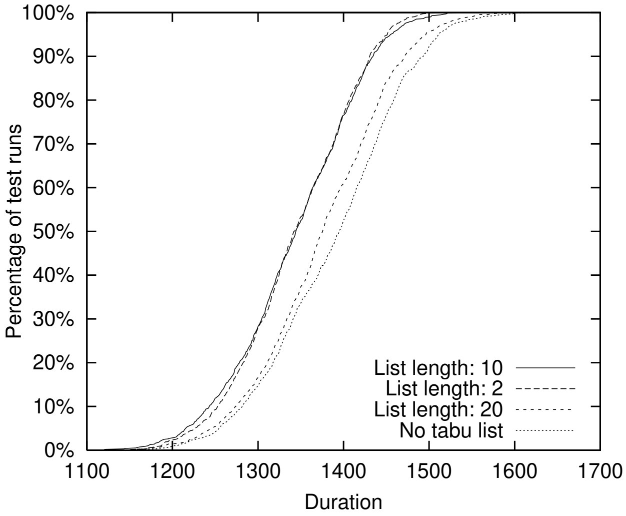

Fig. 7. Durations Found for Problem 6-1a Using Tabu Lists

anytime computations, featuring a steep initial improvement of the inconsistency / goal function. Dynamics were easily handled as well, without the need of a complicated update of a search history — as necessary for (complete) refinement methods. However, only very simple heuristics were used and the results can probably be very much improved with more research on more sophisticated heuristics.

The presented planning system handles combined planning and scheduling and can be extended to handle various types of state domains. Only a simple version of a State Resource Constraint featuring a symbolic state domain was presented. Much more complex SRCs are necessary to tackle more sophisticated problems, e.g., SRCs with state domains of integer numbers, real numbers or even sets, SRCs with enhanced temporal projections like synergistic events and continuous change, and SRCs with enhanced support of precondition checks like state-related or temporal ranges. These extensions are beyond the scope of this paper, but the modular constraint-based framework makes it easy to extend and modify the system.

The system's approach of looking at the problem from a simplified perspective (of one constraint) to choose a heuristic to improve the overall problem is similar to the way in which some other powerful planning approaches proceed, e.g., the approach of Ephrati, Pollack and Milshtein [3], HSP [2] and Fast-Forward [6]. It has actually proved to be a good approach for solving numerous other problems (see also [9]).

In contrast to other local-search approaches to planning, a generative approach was applied. Instead of changing values in pre-defined structures, the structures are generated by the local-search techniques as well. Dispensing with

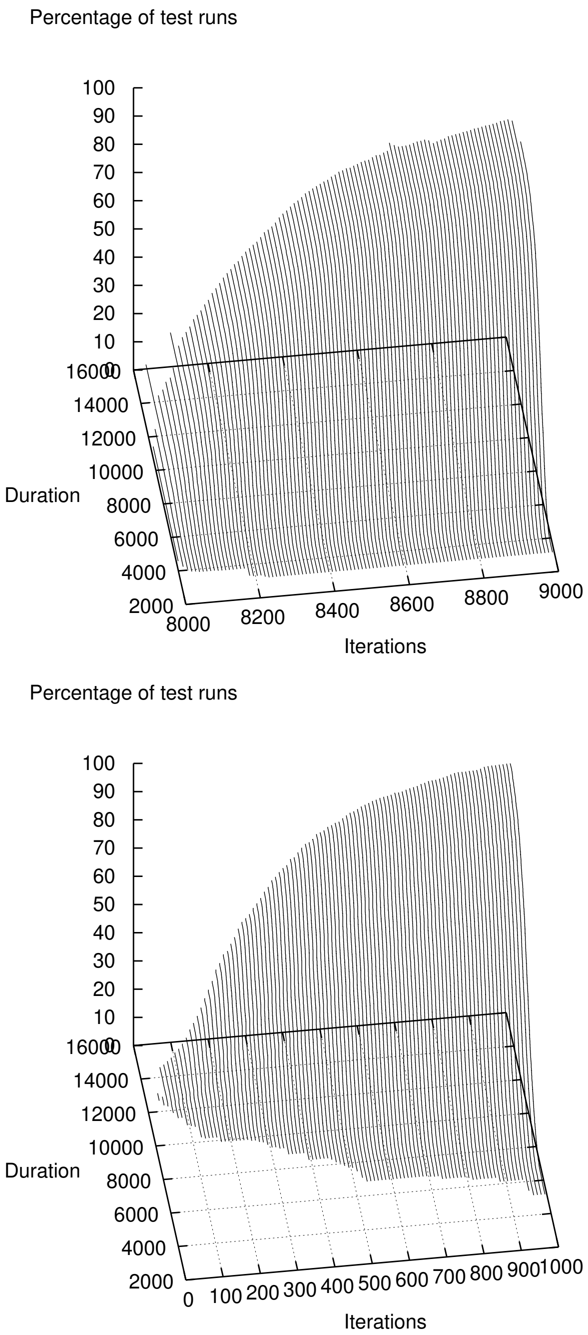

Fig. 8. Dynamic Adaptation (Upper) Vs. Recomputation (Lower) – After Eighth Change

maximal structures has the advantage of being able to handle larger problems, and to search in a more informed way without switching between relatively unrelated problem expansion and search phases. Furthermore, dealing with dynamic environments pose very hard problems for approaches that use maximal structures because these structures have to be extensively manipulated.

Similar work has been done in ASPEN [14] and GERRY [15]. The ASPEN system also deals with generative planning, and although it is tuned to specific domains, the applied heuristics appear to be very similar. GERRY's improvement heuristics also take the same course, although only the subclass of scheduling problems is handled. For future research, it will be very interesting to compare different types of heuristics, and to investigate which combination works for what kind of problems best.

More information on the underlying EXCALIBUR project is available at:

`http://www.ai-center.com/projects/excalibur/`

References

1. Planning Competition of the Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems (AIPS-2000). `http://www.cs.toronto.edu/aips2000/`
2. Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A Robust and Fast Action Selection Mechanism for Planning. In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), 714–719.
3. Ephrati, E.; Pollack, M. E.; and Milshtein, M. 1996. A Cost-Directed Planner: Preliminary Report. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), 1223–1228.
4. Gerevini, A., and Serina, I. 1999. Fast Planning through Greedy Action Graphs. In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), 503–510.
5. Glover, F. 1989. Tabu Search – Part I. *ORSA Journal on Computing* 1(3): 190–206.
6. Hoffmann, J. 2000. A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-climbing Algorithm. In Proceedings of the Twelfth International Symposium on Methodologies for Intelligent Systems.
7. Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92), 359–363.
8. Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), 1194–1201.
9. Korf, R. E. 2000. Recent Progress in the Design and Analysis of Admissible Heuristic Functions. In Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), 1165–1170.
10. Nareyek, A. 2000. Open World Planning as SCSP. In Papers from the AAAI-2000 Workshop on Constraints and AI Planning, Technical Report, WS-00-02, 35–46. AAAI Press, Menlo Park, California. Available via: `http://www.ai-center.com/projects/excalibur/publications.html`
11. Nareyek, A. 2001. Using Global Constraints for Local Search. In Freuder, E. C., and Wallace, R. J. (eds.), *Constraint Programming and Large Scale Discrete Optimization*,

American Mathematical Society Publications, DIMACS Volume 57, 9-28. Available via: `http://www.ai-center.com/projects/excalibur/publications.html`
12. Nareyek, A. 2001. Beyond the Plan-Length Criterion. In *Local Search for Planning and Scheduling*, Springer LNAI 2148. To appear. Available via: `http://www.ai-center.com/projects/excalibur/publications.html`
13. Nareyek, A. 2001. An Empirical Analysis of Weight-Adaptation Strategies for Neighborhoods of Heuristics. In Proceedings of the Fourth Metaheuristics International Conference (MIC'2001). Available via: `http://www.ai-center.com/projects/excalibur/publications.html`
14. Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; and Govindjee, A. 1999. Iterative Repair Planning for Spacecraft Operations in the ASPEN System. International Symposium on Artificial Intelligence Robotics and Automation in Space (iSAIRAS 99).
15. Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and Rescheduling with Iterative Repair. In Zweben, M., and Fox, M. S. (eds.), *Intelligent Scheduling*, Morgan Kaufmann, 241–255.