An Empirical Analysis of Some Heuristic Features for Planning with Local Search in LPG*

Alfonso E. Gerevini[1], Alessandro Saetti[1], and Ivan Serina[2]

[1] DEA, Università degli Studi di Brescia, via Branze 38, 25123 Brescia, Italy
{gerevini,saetti}@ing.unibs.it

[2] Free University of Bozen – Bolzano, Viale Stazione 16, 39042 Bressanone, Italy
ivan.serina@unibz.it

Abstract

LPG is a planner that performed very well in the third and fourth International planning competitions. The system is based on a stochastic local search procedure, and it incorporates several heuristic features. In this paper we experimentally analyze the most important of them with the goal of understanding and evaluating their impact on the performance of the planner. In particular, we examine three heuristic functions for evaluating the search neighborhood and some settings of the "noise" parameter, that randomizes the next search step for escaping from local minima. Moreover, we present and analyze additional heuristic techniques for restricting the search neighborhood and for selecting the next inconsistency to handle. The experimental results show that the use of such techniques significantly improves the performance of the planner.

1 Introduction

The results of the 3rd and 4th International planning competitions [13, 12] showed that LPG is an efficient planner for PDDL2.2 domains [7, 8, 9]. The system is based on a stochastic local search procedure, called Walkplan, that is similar to the well-known Walksat procedure for solving SAT problems [15].

As in any local search scheme, the definition of the search neighborhood (the set of possible successor states) and the heuristic function for evaluating its elements are crucial features for the effectiveness of Walkplan. When the number of the elements in the neighborhood is high, its evaluation can be computationally expensive, and a technique for pruning some elements can be very effective. Moreover, in an iterative-repair approach, the strategy to select the next flaw to handle (inconsistency in LPG, unsatisfied clause in Walksat) may also affect the performance of the search.

In order to escape from local minima, in Walkplan as in Walksat, if every element in the neighborhood is worse than the current state (according to an heuristic function), then with some probability (called "noise") an element of the neighborhood is randomly chosen, instead of selecting the best one. In general, the value of the noise can significantly affect the performance of the search. In LPG the noise value can be either statically set by the user, or automatically set to an initial default value that is dynamically changed during search.

This paper has two main contributions:

*This paper has been already published in the Proceedings of the Fourteenth International Conference on Automated Planning & Scheduling (ICAPS-04).

Proceedings of the 16th International RCRA workshop (RCRA 2009):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Reggio Emilia, Italy, 12 December 2009

- we propose some techniques for effectively restricting the search neighborhood of Walkplan, and for selecting the next inconsistency to handle;
- we experimentally analyze the main heuristic features for local search in LPG with the goal of understanding and evaluating their impact on the performance of the planner.

In addition to the techniques for neighborhood restriction and inconsistency selection, we analyze three heuristic functions for evaluating the neighborhood elements, that we introduced in previous work [6, 7, 8], and the noise setting. We focus our analysis on simple STRIPS domains.

The second section gives the necessary background on LPG and Walkplan; the third section presents the techniques for the neighborhood restriction and the inconsistency selection; the fourth section presents and discusses the results of our experimental analysis; finally the last section gives the conclusions.

2 Background on LPG

In this section, we give an overview of LPG's plan representation, search space, search algorithm and main heuristic features [6, 7, 8].

2.1 Plan Representation: Linear Action Graphs

In our framework, plans are represented through *action graphs* [6] that are particular subgraphs of planning graphs [1]. Given a planning graph $\mathcal{G}$ for a planning problem Π, an *action graph* (A-graph) for Π is a subgraph $\mathcal{A}$ of $\mathcal{G}$ such that, if a is an action node of $\mathcal{G}$ in $\mathcal{A}$, then also the fact nodes of $\mathcal{G}$ corresponding to the preconditions and positive effects of a are in $\mathcal{A}$, together with the edges connecting them to a. An action graph can contain some *inconsistencies*, i.e., action preconditions that are not *supported*, or pairs of action nodes involved in *mutex relations*.[1] A precondition node q at a level i is supported in an action graph $\mathcal{A}$ of $\mathcal{G}$ if in $\mathcal{A}$ there is an action node (or a no-op node) at level $i-1$ representing an action with a positive effect q.[2] An action graph without inconsistencies represents a valid plan that we call a *solution graph*.

In the current version of LPG, plans for STRIPS problems are represented through a subclass of A-graphs called *linear action graph with no-ops propagation (LA-graphs)*.[3] A linear action graph $\mathcal{A}$ is an A-graph in which each action level contains at most one action node and any number of no-op nodes. Moreover, if a is an action node of $\mathcal{A}$ at a level l, then, for any positive effect e of a and any level $l' > l$ of $\mathcal{A}$, the no-op node of e at a level l' is in $\mathcal{A}$, unless there is another action node at a level l'' ($l < l'' \leq l'$) that is mutex with the no-op node. In any LA-graph, the only inconsistencies that the search procedure needs to manage explicitly are the unsupported preconditions. Although an LA-graph cannot contain more than one action

[1] LPG considers only pairs of actions that are *globally mutex*, i.e. that hold at every level of $\mathcal{G}$ [8].

[2] We assume that the goal nodes of $\mathcal{G}$ (i.e., the problem goals) represent the preconditions of a special action a_{end}, which is the last action in any valid plan; the fact nodes of the first level of $\mathcal{G}$ (i.e., the initial facts of the problem) represent the effects of a special action a_{start}, which is the first action in any valid plan. a_{start} and a_{end} belong to any A-graph of $\mathcal{G}$.

[3] In order to handle domains involving time and numerical variables (levels 2 and 3 of PDDL2.1), LPG uses some extensions of LA-graphs that, however, will not be considered in this paper.

per level, a plan with parallel actions (a partial order plan) can be derived from a solution graph by considering the causal dependencies and mutex relations between the actions in the graph. LA-graphs offer some advantages with respect to A-graph that are discussed in [8]. All experiments presented in this paper were conducted using this representation.

2.2 Stochastic Local Search: Walkplan

Given a planning problem Π, LPG uses local search for searching a solution LA-graph in the space of the LA-graphs for Π. The general scheme consists of two main steps. First we construct an initial LA-graph; then we iteratively apply some graph modifications to transform the initial LA-graph into a solution graph. In the current version of LPG, the default initial graph is the empty LA-graph with the "fixed-point level" of the underlying planning graph as the last level.

At each search step, LPG selects an inconsistency (unsupported precondition) in the current LA-graph. As will be shown, the strategy for selecting the next inconsistency to handle can have a significant impact on the overall performance. In the next sections we will present and analyze some possible strategies that are implemented in LPG.

In order to resolve the selected inconsistency, we can either add an action node that supports it, or we can remove an action node that is connected to that fact node by a precondition edge. When we add an action node to a level l, the LA-graph is extended by one level, all action nodes from l are shifted forward by one level, and the new action is inserted at level l (a more detailed description is given in [8]).

Given a linear action graph $\mathcal{A}$ and an inconsistency σ in $\mathcal{A}$, the *neighborhood* $N(\sigma, \mathcal{A})$ of $\mathcal{A}$ for σ is the set of LA-graphs obtained from $\mathcal{A}$ by applying a graph modification that resolves σ. At each step of the local search scheme, the elements of the neighborhood are evaluated according to an heuristic function estimating their quality, and an element with the best quality is then chosen as the next possible LA-graph (search state). Evaluating all elements in the neighborhood can be computationally very expensive, because the neighborhood could contain many LA-graphs and an accurate evaluation of each of them could require significant CPU-time. For this reason, as we will show, it is important that the evaluation of the neighborhood elements is combined with a technique that reduces its size.

The default search strategy used by LPG is called Walkplan. Walkplan is similar to Walksat, a well-known local search method for solving propositional satisfiability problems [15]. In Walkplan the best element in the neighborhood is the LA-graph which has the *lowest decrease of quality* with respect to the current LA-graph, i.e., it does not consider possible improvements. Given an LA-graph $\mathcal{A}$ and an inconsistency σ, if there is a modification for σ that does not decrease the quality of $\mathcal{A}$, then the resulting LA-graph is chosen as the next LA-graph; otherwise, with a probability p one of the graphs in $N(\sigma, \mathcal{A})$ is randomly chosen, and with probability $1 - p$ the next LA-graph is the best element in $N(\sigma, \mathcal{A})$ according to an *action evaluation function* E (an element with the lowest evaluation). As in Walksat, p is called *noise factor*, and its value may have a significant impact on the search. When $N(\sigma, \mathcal{A})$ contains more than one graph with the best evaluation, LPG chooses randomly one of them. Finally, if after a certain number of search steps (*max_steps*) a solution LA-graph is not reached, the current LA-graph is reinitialized, and the search is repeated up to a user-defined maximum number of times (*max_restarts*).

2.3 Heuristic Evaluation of the Search Neighborhood

The neighborhood evaluation function has two parts evaluating: the search cost of an LA-graph in the neighborhood (i.e., the number of search steps required to reach a solution graph); the quality (or execution cost) of the (partial) plan represented by the LA-graph. In this section we focus on the first part. At the end of the section we briefly describe how execution cost is evaluated.

In the design of a neighborhood evaluation function, there is an important tradeoff to consider between accuracy of the evaluation and the computational cost of the evaluation. An accurate evaluation of the elements in the neighborhood could lead to a valid plan within few search steps. However, when the neighborhood contains many elements, the evaluation could slow down the search excessively. On the other hand, a less accurate evaluation of the neighborhood is faster to compute, but since it is less informative it could lead to a valid plan only after many search steps. We developed three evaluation functions trying to identify an appropriate balance between informativeness and efficiency of their computation: E_0, E_H and E_π. In the rest of this section we describe each of them, while in the section about the experimental results we analyze their impact on the performance of Walkplan. For each $E \in \{E_0, E_H, E_\pi\}$, $E(a, \mathcal{A})^i$ is the evaluation of the LA-graph derived from the current graph $\mathcal{A}$ by adding the action a to it (also called the "cost of adding a to $\mathcal{A}$"). Similarly, $E(a, \mathcal{A})^r$ is the cost of removing a from $\mathcal{A}$.

The simplest function that we consider was proposed in [6] and is defined as follows.

Heuristic function E_0:

$$E_0(a, \mathcal{A})^i = |P(a, \mathcal{A})| + |Threats(a, \mathcal{A})|$$

$$E_0(a, \mathcal{A})^r = |Unsup(a, \mathcal{A})|$$

where *Threats*$(a, \mathcal{A})$ is the set of supported precondition facts in $\mathcal{A}$ that become unsupported by adding a to $\mathcal{A}$; $P(a, \mathcal{A})$ is the set of the precondition facts of a that are not supported in $\mathcal{A}$; *Unsup*$(a, \mathcal{A})$ is the set of supported precondition facts in $\mathcal{A}$ that become unsupported by removing a from $\mathcal{A}$.

While computing E_0 in the context of LA-graphs is quite fast, a more accurate evaluation could be more effective. In fact, it can be the case that, although the insertion of a new action a_i leads to fewer new unsupported preconditions than those introduced by an alternative action a_j, the unsupported preconditions of a_i are more difficult to satisfy (support) than those of a_j in the context of the current partial plan. For this reason, subsequently we proposed two alternative, more informative functions, E_H [7] and E_π [8].

E_H is a refinement of E_0 in which, instead of just counting the number of the unsupported preconditions of a and those in *Unsup*$(a, \mathcal{A})$, we estimate the search cost of achieving them. More precisely, E_H is defined as follows.

Heuristic function E_H:

$$E_H(a, \mathcal{A})^i = \underset{f \in Pre(a)}{MAX}\ H(f, \mathcal{A}) + |Threats(a, \mathcal{A})|$$

$$E_H(a, \mathcal{A})^r = \underset{f \in Unsup(a, \mathcal{A})}{MAX}\ H(f, \mathcal{A} - a)$$

where *Pre*(a) is the set of the preconditions of a and $H(f, \mathcal{A})$ is the *heuristic cost of supporting*

f, which is recursively defined in the following way:

$$H(f,\mathcal{A}) = \begin{cases} 0 & \textit{if } f \textit{ is supported} \\ H(f',\mathcal{A}) & \textit{if } a^f \textit{ is no-op with precondition } f' \\ \underset{f' \in Pre(a^f)}{MAX} H(f',\mathcal{A}) + |Threats(a^f,\mathcal{A})| + 1 \end{cases}$$

where

$$a^f = \underset{\{a' \in A_f\}}{ARGMIN} \left\{E_0(a',\mathcal{A})^i\right\}$$

and A_f is the set of action nodes of the underlying planning graph at the levels preceding f that have f as one their effect nodes. Informally, the heuristic cost of an unsupported fact f is determined by considering all the actions at a level preceding f whose addition would support it. Among these actions, we choose the one with the best evaluation (a^f) according to the basic action evaluation function E_0. $H(f,\mathcal{A})$ is recursively computed by summing the highest heuristic cost of supporting a precondition f' of a^f in $\mathcal{A}$ ($H(f',\mathcal{A})$) and the number of supported precondition facts in $\mathcal{A}$ that become unsupported by adding a^f to $\mathcal{A}$ ($|\textit{Threats}(a^f,\mathcal{A})|$). The last term "+1" takes account of the insertion of a^f to support f.

E_π [8] was designed to improve the accuracy of E_H. It was used by LPG in the 3rd and 4th planning competitions. In the evaluation of the addition of a new action a, E_π estimates the search cost of achieving *all* preconditions of a ($\textit{Pre}(a)$) in the context of the current LA-graph, while E_H considers the maximum over the search costs of its preconditions. Moreover, instead of just counting the number of action preconditions in $\mathcal{A}$ that would become unsupported when adding a ($\textit{Threats}(a,\mathcal{A})$), E_π estimates the search cost of re-achieving such conditions after the addition of a. More precisely, in E_π the search costs are estimated by computing a relaxed plan π_r for achieving $\textit{Pre}(a)$ and $\textit{Threats}(a,\mathcal{A})$, and counting the number of actions in π_r. In addition, E_π counts the number of the action preconditions in $\mathcal{A}$ that are subverted by an action a' in π_r ($\textit{Threats}(a',\mathcal{A})$). E_π is formally defined as follows.

Heuristic function E_π:

$$E_\pi(a,\mathcal{A})^i = |\pi(a,\mathcal{A})^i| + \textstyle\sum_{a' \in \pi(a,\mathcal{A})^i} |Threats(a',\mathcal{A})|$$

$$E_\pi(a,\mathcal{A})^r = |\pi(a,\mathcal{A})^r| + \textstyle\sum_{a' \in \pi(a,\mathcal{A})^r} |Threats(a',\mathcal{A})|$$

where

- $\pi(a,\mathcal{A})^i$ is an estimate of a minimal set of actions forming a relaxed plan achieving $\textit{Pre}(a)$ and $\textit{Threats}(a,\mathcal{A})$;
- $\pi(a,\mathcal{A})^r$ is an estimate of a minimal set of actions forming a relaxed plan achieving $\textit{Unsup}(a,\mathcal{A})$.

The plans of E_π are relaxed because their validity do not consider the negative effects of the actions. However, negative effects are considered in the heuristic selection of the actions forming a relaxed plan (more details below). The initial state $I(l)$ of the (relaxed) problem of achieving either $\textit{Pre}(a)$ or $\textit{Unsup}(a,\mathcal{A})$ is the state obtained by applying the actions in $\mathcal{A}$ up

RelaxedPlan(G, $I(l)$, A)

Input: A set of goal facts (G), the set of facts that are true after executing the actions of the current LA-graph up to level l ($I(l)$), a possibly empty set of actions (A);

Output: An estimated minimal set of actions required to achieve G.

1. $G \leftarrow G - I(l)$; $Acts \leftarrow A$;
2. $F \leftarrow \bigcup_{a \in Acts} Add(a)$;
3. **while** $G - F \neq \emptyset$
4. $\quad g \leftarrow$ a fact in $G - F$;
5. $\quad bestact \leftarrow Bestaction(g)$;
6. $\quad Rplan \leftarrow$ RelaxedPlan($Pre(bestact), I(l), Acts$);
7. $\quad Acts \leftarrow Rplan \cup \{bestact\}$;
8. $\quad F \leftarrow \bigcup_{a \in Acts} Add(a)$;
9. **return** $Acts$.

Figure 1: Algorithm for computing a relaxed plan achieving a set of action preconditions from the state $I(l)$.

to level $l-1$ (ordered according to their corresponding levels). The initial state from which we achieve *Threats*$(a, \mathcal{A})$ is the state obtained by applying a to $I(l)$.

$\pi(a, \mathcal{A})^i$ and $\pi(a, \mathcal{A})^r$ are computed by an algorithm called RelaxedPlan (see Figure 1). For $\pi(a, \mathcal{A})^i$, RelaxedPlan is run twice, first to achieve *Pre*(a) and then to achieve *Threats*$(a, \mathcal{A})$. The set of actions identified by the first run is given as input to the second run, so that the relaxed subplan for *Threats*$(a, \mathcal{A})$ can reuse the actions in the subplan for *Pre*(a).

RelaxedPlan constructs a plan through a backward process from the input goal set G to the input initial state. The action chosen to achieve a (sub)goal g, *Bestaction(g)*, is determined by considering for each fact f an estimate of the minimum number of actions required to achieve f from $I(l)$ ($Num\_acts(f, l)$). A detailed description of this reachability information and of its computation is given in [8].
More formally, *Bestaction(g)* is defined as

$$\underset{\{a' \in A_g\}}{\textit{ARGMIN}} \left\{ \underset{p \in Pre(a') - F}{\textit{MAX}} Num\_acts(p, l) + |Threats(a', \mathcal{A})| \right\},$$

where F is the set of positive effects of the actions currently in the relaxed plan (*Acts*), and A_g is the set of actions with the effect g and with all preconditions reachable from the initial state. For a more detailed description of *Bestaction*, RelaxedPlan and E_π, the interested reader may see [8].

Finally, we briefly describe the heuristic evaluation of the execution cost associated with the plan represented by an LA-graph in the search neighborhood. The action evaluation function is a normalized linear combination of the search cost and the execution cost, that for STRIPS domains is defined as the number of actions in the plan. In E_0^i the execution cost is modeled by a "+1" term, while in E_0^r we do not have this term; in E_H it is modeled by a term equal to the maximum depth of the tree of action nodes identified by H; finally, in E_π it is modeled by a term equal to the number of the actions in the relaxed plan [8].

3 Additional Heuristic Features

In this section we propose some additional heuristic features that have a significant impact on the performance of LPG. In particular, we present techniques for restricting the search neighborhood and selecting the next inconsistency to handle.

3.1 Neighborhood Restriction

In general, the effectiveness of a heuristic function evaluating the elements in the search neighborhood can be significantly affected by the size of the neighborhood. If this is too large, an accurate evaluation might require too much time, and a less accurate (but computationally more efficient) function could perform better. Since LPG's basic search neighborhood can be very large, we considered some alternative restricted neighborhoods, and we compared the performance of E_0, E_H and E_π using them. The results of this experiment are presented in the next section. In this section, first we overview the basic neighborhood of LPG, and then we present some techniques to restrict it.

The basic neighborhood $N(p, \mathcal{A})$ of an LA-graph $\mathcal{A}$ for an unsupported precondition node p is the set of the LA-graphs that can be derived from $\mathcal{A}$ by adding an action node supporting p, or by removing the action node with precondition p. Suppose that p is a precondition node at a level l of $\mathcal{A}$. In order to support p we can add an action a with positive effect p to *any* level $l' \leq l$, provided that p can be propagated to l by adding a no-op for p to the levels between l' and l. (The propagation is not possible if at any of such intermediate levels there is an action node that is mutex with the no-op of p.) In the basic definition of $N(p, \mathcal{A})$ for LA-graphs, we have an LA-graph for each of these graph modifications. Moreover, $N(p, \mathcal{A})$ contains the LA-graph obtained by removing the action node of which p is a precondition.[4]

While any restriction of the basic neighborhood makes its evaluation faster, clearly not every restriction can speed up the search of a solution graph. In particular, we would like to remove from the neighborhood only the bad elements (those requiring more search steps to reach a solution graph).

In order to select the elements forming a restricted search neighborhood, LPG pre-evaluates the candidate elements of the basic neighborhood using the same method used in E_π to select the actions forming a relaxed plan. Specifically, if $\mathcal{A}'$ is the LA-graph derived by adding an action a' to a level l of $\mathcal{A}$, the quality of $\mathcal{A}'$ is defined as the maximum over

$$Num\_acts(q, l) + |Threats(a', \mathcal{A})|$$

for every unsupported precondition q of a'. Clearly, this evaluation is much faster to compute than E_π, but it is also less accurate.

We considered four strategies for determining the number of the elements forming the restricted neighborhood: NR_k, $NR_{k\%}$, NR_A, and NR_L. They differ from the basic neighborhood because they contain only a subset of the LA-graphs obtained by adding an action node to the current LA-graph. NR_k, $NR_{k\%}$ and NR_A exploit reachability information that is available

[4] Since at any level there can be at most one action node (plus any number of no-ops), when we remove an action node, the corresponding action level becomes "empty" (i.e., it contains only no-ops). If the LA-graph contains adjacent empty levels, and in order to support p a certain action node can be added to any of these levels, then $N(p, \mathcal{A})$ contains only one of the resulting graphs.

when the heuristic evaluation function is E_π; NR_L is a simpler technique that can be used in combination with any of the heuristic functions that we have described.

In NR_k, $N(p, \mathcal{A})$ contains only the k best pre-evaluated elements. Note that if there are many candidate elements of good quality according to the pre-evaluation, it could be the case that NR_k removes elements that actually have better quality according to the heuristic evaluation function.

In $NR_{k\%}$, the search neighborhood contains only those elements, whose pre-evaluation is worse than the best element by a factor less than or equal to $k\%$ (k is an input parameter for both NR_k and $NR_{k\%}$). $NR_{k\%}$ was designed to remove from the neighborhood only those elements that the pre-evaluation considers significantly bad, and so to reduce the probability of erroneously removing elements that the heuristic evaluation function would consider of good quality.

In NR_A, for each action a with effect p, $N(p, \mathcal{A})$ contains only one LA-graph (while the basic neighborhood contains an LA-graph for any level where a can be added and p can be propagated up to the level of the inconsistency). The best level where adding a is decided according to the pre-evaluation above. If there is more than one element with the best evaluation, NR_A chooses the one where a is put at the highest level. NR_A is the restriction used by LPG in the third and fourth IPCs.

NR_L is a simplified version of NR_A in which the best level where we add a is just the *last* level where a can be added, i.e. the same level of the unsupported precondition p.

Finally, we considered a fifth strategy in which we combine all three restriction techniques that use reachability information. We call such a strategy NR_{all}.

3.2 Inconsistency Selection

In partial-order causal-link (POCL) planning, the choice of the next inconsistency to repair at each search step can significantly affect the performance of the search process (e.g., [5, 14]). Although the search methods of LPG and POCL-planners are radically different (and hence we cannot directly import results from POCL planning into our framework), preliminary experimental tests showed that this is the case also for LPG: if we change the basic random selection strategy of Walkplan, the performance of the planner can be significantly different.[5] Thus, we designed some alternative strategies with the aim of improving the performance of the random strategy (indicated with Σ_{rand}).

In this section we introduce two of them, Σ_{lifo} and Σ_{lev-}, and in the next section we evaluate their performance with respect to the default random strategy.

Σ_{lifo} is a simple standard strategy that handles the inconsistencies according to a last-in first-out discipline. Σ_{lev-} prefers the inconsistency at the *lowest level* t of the current LA-graph; if more than one of such inconsistencies are present at level t, then Σ_{lev-} chooses randomly one of them. The rationale of Σ_{lev-} is related to the definition of the evaluation function E_π, for which it was initially designed. As we have seen, $E_\pi(a, \mathcal{A})^i$ is defined using a relaxed plan π_r for achieving the preconditions of a from the state $I(l)$, where $I(l)$ is the state obtained by executing the actions at the levels preceding the level l of the selected inconsistency starting from the initial state. Moreover, the actions forming π_r are selected using reachability

[5] The random selection of the next inconsistency to handle in Walkplan is the analogue of the random selection of the next unsatisfied clause to handle in Walksat [15].

information for their preconditions that are dynamically computed from $I(l)$. By removing the inconsistency at the earliest level l of the current LA-graph, we guarantee that $I(l)$ is the state s reached by the actions at the levels preceding l. On the contrary, if we randomly select an inconsistency at any level l, and there are other inconsistencies before l, then $I(l)$ can be only an *estimate* of s. Such an estimate could change in the next LA-graphs of the search (because some actions are added/removed to deal with inconsistencies at levels before l). Hence, the evaluation of $E_\pi(a, \mathcal{A})^i$ at a certain search step could change radically at a next search step. This could lead E_π combined with Σ_{rand} to make incorrect evaluations more often than when E_π is combined with Σ_{lev-}. For similar reasons, we conjecture that also E_0 and E_H can benefit from the use of Σ_{lev-}. In this paper we will restrict the experimental analysis of Σ_{lev-} by considering only E_π. Σ_{lev-} is the inconsistency selection strategy used by LPG in the third IPC.

4 Experimental Results

In this section we present some experimental results illustrating the performance of the heuristic features for LPG described in the previous sections. We will use the test problems of the 3rd IPC [13]. For lack of space we will show the experimental results for only the 102 problems of the STRIPS domain variants (`Depots`, `DriverLog`, `Rovers`, `Satellite` and `Zenotravel`).[6] Similar experimental results were obtained for the temporal domain variants of the 3rd IPC.

LPG is an incremental planner, in the sense that it produces a sequence of valid plans, each of which improves the quality of the previous ones. We tested LPG in terms of both the CPU-time required to find a solution (LPG-speed) and the quality of the best plan computed by the incremental process using five CPU-minutes (LPG-quality).[7]

We compare different options of an heuristic feature using Friedman's statistical test [4]. Note that when there are more than two options to compare, Friedman's test is more accurate than Wilcoxon's test [17], that Long and Fox used to compare the performance of *pairs* of planners [13].

The first observation that we derived from our experimental results is that the assumptions of the Friedman's test procedure [4] are satisfied for every heuristic feature analyzed in this paper.

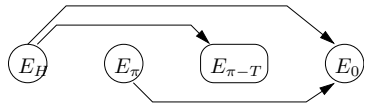
Observation 1. *The noise value, the heuristic evaluation of the search neighborhood, the neighborhood restriction and the inconsistency selection are statistically meaningful features for the performance of* LPG.

[6] These domains are described at `www.dur.ac.uk/ d.p.long/competition.html`. We have not considered the `Freecell` domain because currently LPG solves the problems of this domain only by using the alternative best-search mode [8].

[7] Since Walkplan is a stochastic procedure, we ran LPG five times for problem tested. The tests were conducted on a PIII Intel 866 MHz with 512 Mbytes of RAM. In every run the *max_steps* parameter of Walkplan was set to 500, and it was automatically increased by 10% at each search restart.

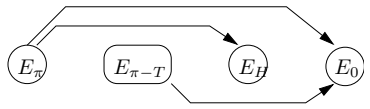
Results using *NoNR*	E_0	E_H	E_π	$E_{\pi-T}$
Problems Solved	72.5%	86.3%	99.0%	87.3%
Speed Mean Rank	12.5	8.4	10.2	11.0
Quality Mean Rank	12.1	11.0	8.9	10.0

CPU-time: $Friedman(E_0, E_H, E_\pi, E_{\pi-T})$



$D = 1.9$

Plan Quality: $Friedman(E_0, E_H, E_\pi, E_{\pi-T})$



$D = 1.3$

Table 1: Results of Friedman's test on the performance of Walkplan using four evaluation functions (E_0, E_H, E_π, and $E_{\pi-T}$) without neighborhood restriction: percentage of problems solved, CPU-time and plan quality ranks.

4.1 Heuristic Evaluation of the Search Neighborhood

In this section we comment on the performance of LPG with different heuristic evaluations of the search neighborhood: E_0, E_H, E_π, and E_π without considering the terms counting the threats ($E_{\pi-T}$). $E_{\pi-T}$ is tested in order to show the importance considering the threats in the evaluation of E_π. Note that this is an important difference in the construction of the relaxed plans computed by LPG and those computed by FF [10] and the first version of SAPA [2].[8]

E_H and E_π are more accurate than E_0. On the other hand, as expected, we observed that E_0 is the fastest to compute, E_H is slower than E_0, and E_π is the slowest one.

Table 1 shows the performance of different neighborhood evaluation functions in terms of both CPU-time and plan quality when no restriction is imposed on the neighborhood. The graphs in each table should be interpreted as follows. An edge connecting an option $\mathcal{T}$ to another option $\mathcal{T}'$ indicates that $\mathcal{T}$ performs statistically better than $\mathcal{T}'$. For instance, in Table 1 E_H performs better than E_0 in terms of CPU-time. If an edge connects a cluster of options to a certain option $\mathcal{T}$, as in Table 3, it means that any option in the cluster is statistically better than $\mathcal{T}$. Each graph is annotated with the corresponding D-value.

Observation 2. *Without any restriction of the search neighborhood, in terms of CPU-time, E_H is statistically better than both E_0 and $E_{\pi-T}$, and E_π is statistically better only than E_0.*

The fact that E_π does not perform better than E_H is somewhat disappointing. It shows that without restricting the search neighborhood, a more accurate but computationally more expensive function does not pay off.

Observation 3. *A complete examination of the search neighborhood using E_π can be too expensive.*

[8] The last version of SAPA [3] considers static mutex relations in order to improve the makespan of the relaxed plan, and to check if the relaxed plan is a valid plan for the planning problem.

Heuristic Function	Speed Mean Rank		Quality Mean Rank	
	NoNR	NR_L	*NoNR*	NR_L
E_0	25.2	27.1	22.7	25.2
E_H	17.7	20.6	20.3	25.1
E_π	21.5	**13.5**	16.3	17.4
$E_{\pi-T}$	23.3	15.2	18.5	18.7

Table 2: Mean ranks of CPU-time and plan quality using different heuristic evaluations of the search neighborhood with/out neighborhood restriction (NR_L/*NoNR*).

The previous observation suggests that the use of an accurate evaluation should be combined with a restriction of the search neighborhood. In fact, if we combine the use of E_π with a restriction of the search neighborhood, we obtain a different picture. In particular, if we use the simple restriction NR_L introduced in the previous section, we have that Walkplan with E_π performs more efficiently than with any other evaluation function (see Table 2).

Regarding plan quality, as expected, the most accurate heuristic evaluation functions (E_π and $E_{\pi-T}$) performs better than the other evaluation functions. Moreover, the results of Friedman's test show that $E_{\pi-T}$ is worse than E_π (see the mean ranks of tables 1 and 2), which demonstrates the importance of taking threats into account.

Observation 4. *Without any restriction of the search neighborhood, in terms of plan quality, E_π is statistically better than both E_H and E_0, while $E_{\pi-T}$ is better than E_0.*

4.2 Noise Value for Walkplan

It is well known that the performance of a stochastic local search procedure (like Walkplan) can depend on the value of its noise parameter, that is used for escaping from local minima [15]. In this section we empirically analyze the performance of Walkplan using different noise values, including the special case in which the noise is set to zero. Since the impact of the noise value may also depend on which heuristic function is used to evaluate the neighborhood, the experimental analysis for the noise parameter was conducted using two heuristic functions: the simplest function (E_0) and the most accurate one (E_π).

Tables 3 and 4 show the results of Friedman's test for E_0 and E_π, respectively, using different static values for the noise, as well as a *dynamic value* (N_{dyn}) that is defined in the following way. The search process starts with a low noise value and it considers the variance of the number of inconsistencies in the last t visited LA-graphs. The idea is that a low variance indicates the presence of a local minimum; in this case, the noise value is increased to facilities escaping from the minimum. The variance is checked every t search step (in our tests we used $t = 25$). If it is less than one, the noise value is increased by a certain factor (we used 1.5). However, it can increase only up to a maximum value that depends on which neighborhood evaluation function is used.[9] If the variance is greater than one, the noise value is set to its initial default value.

Observation 5. *Using E_0, the dynamic noise performs statistically better than low noise values, and slightly better than high noise values, in terms of both CPU-time and plan quality.*

[9] Such maximum values were empirically determined by observing the performance of each evaluation function using various noise values for all test problems considered in this paper.

Results using E_0	N_0	N_{10}	N_{20}	N_{50}	N_{dyn}
Problems Solved	2%	59.8%	68.6%	64.7%	72.5%
CPU-time Mean Rank	19.1	14.5	11.9	10.0	9.5
Quality Mean Rank	19.1	12.9	11.5	11.2	10.3

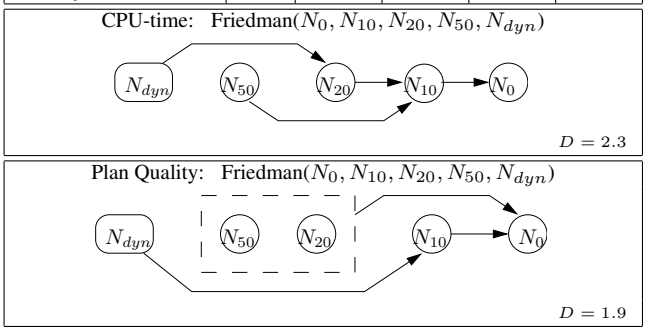


Table 3: Results of Friedman's test on the performance of Walkplan using E_0 with different noise settings (0%, 10%, 20%, 50% and dynamic noise): percentage of problems solved, CPU-time and plan quality ranks.

Results using E_π	N_0	N_{10}	N_{20}	N_{50}	N_{dyn}
Problems Solved	93.1%	97.1%	94.1%	71.6%	98.0%
CPU-time Rank	9.7	10.5	13.7	20.9	10.2
Quality Rank	11.3	11.2	13.1	18.9	10.5

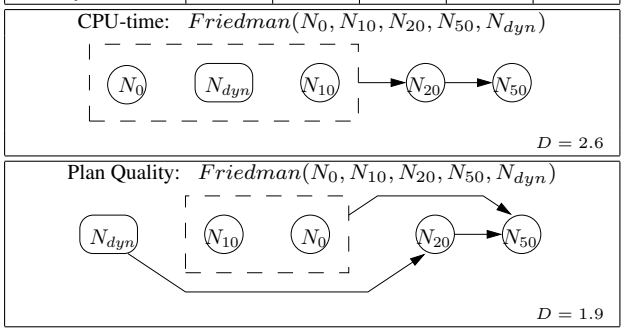


Table 4: Results of Friedman's test on the performance of Walkplan using E_π with different noise settings (0%, 10%, 20%, 50% and dynamic): percentage of problems solved, CPU-time and plan quality ranks.

Observation 6. *Using E_π, low noise values and the dynamic noise perform statistically better than high noise values, in terms of both CPU-time and plan quality.*

The reason why the dynamic noise performs better than with the other static noise settings can be intuitively explained by the fact that this method tends to make random choices only when needed, i.e., only when the search has reached (or is near to) a local minimum. A high value of the noise setting facilitates escaping from local minima, but obviously it could affect negatively the performance by introducing unsupported preconditions that are difficult to achieve in the context of the current LA-graph, as well as many threats. This determines an increment of the total CPU-time and slows down the incremental process for finding good quality plans.

Results using E_π	*NoNR*	NR_L	NR_A	NR_k	$NR_{k\%}$	NR_{all}
Problems Solved	92.2%	99%	98%	96.1%	98.0%	99.0%
CPU-time Mean Rank	21.3	12.9	13.2	14.8	17.7	13.0
Quality Mean Rank	17.1	18.6	14.9	13.9	16.0	12.5

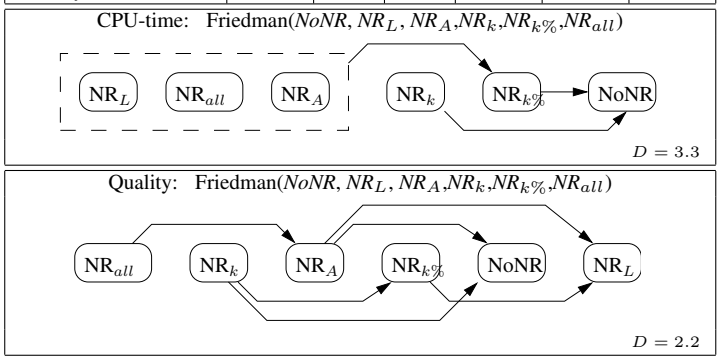


Table 5: Results of Friedman's test on the performance of Walkplan using E_π with the neighborhood restriction techniques (*NoNR*, NR_L, NR_A, NR_k, $NR_k\%$ and NR_{all}): percentage of problems solved, CPU-time and plan quality ranks.

Observation 7. *The performance of* Walkplan *using E_0 with 0% of noise is statistically worse than with the other noise values analyzed, in terms of both CPU-time and plan quality.*

Observation 8. *The performance of* Walkplan *using E_π with 0% of noise is not statistically different from the performance with low noise values, in terms of both CPU-time and plan quality.*

The reason why Walkplan using E_π performs better with low noise values seems mainly related to the very good accuracy of the neighborhood evaluation of E_π. In particular, we experimentally observed that, while E_0 can often lead to local minima, E_π rarely does so.[10] Hence, a high value of the noise when using E_π can often "destruct" the search towards the solution graph. On the contrary, when using E_0, performing random choices among the elements of the neighborhood is more often useful to abandon portions of the search space containing local minima.

4.3 Neighborhood Restriction

As we have seen, the effectiveness of E_π can be improved by restricting the search neighborhood using NR_L. In this section we focus on E_π, and we analyze the relative importance of the neighborhood restriction options that we introduced in the previous section.

Table 5 shows the results of Friedman's test for Walkplan with no neighborhood restriction (*NoNR*), NR_L, NR_A, NR_k, $NR_{k\%}$, and NR_{all}. For NR_k we used $k = 10$, and for $NR_{k\%}$ we used $k = 300$.

Observation 9. *In terms of CPU-time,* Walkplan *with any of the neighborhood restrictions analyzed performs statistically better than with no restriction.*

[10] Hoffmann observed a similar behavior for the relaxed-plan heuristic used by FF, that he tested on different domains [11]. Moreover, the heuristics of LPG and FF, as well as their search spaces, are different.

Results (E_π with NR_A)	Σ_{rand}	Σ_{lev^-}	Σ_{lifo}
Problems Solved	97.1%	98.0%	94.1%
Speed Mean Rank	9.2	6.0	8.8
Quality Mean Rank	8.4	6.3	9.3

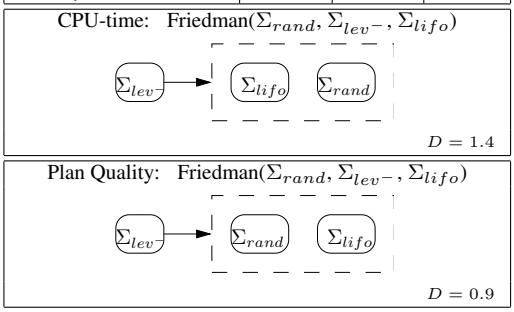


Table 6: Results of Friedman's test on the performance of Walkplan using E_π with NR_A and one of the inconsistency selection strategies Σ_{rand}, Σ_{lev^-} and Σ_{lifo}: percentage of problems solved, CPU-time and plan quality ranks.

An accurate evaluation of every neighborhood element using E_π is computationally expensive. For this reason, Walkplan with a restricted neighborhood visits a portion of the search space that can be significantly larger than the portion visited with no restriction in the same amount of CPU-time. This makes E_π more effective in terms of both CPU-time and plan quality.

Observation 10. *NR_{all} is statistically the best neighborhood restriction among those considered, in terms of both CPU-time and plan quality.*

The removal of some neighborhood elements can be harmful for the local search process, because it can eliminate the only elements that could take the search away from a local minimum in which it is trapped. NR_k and $NR_{k\%}$ appear to be more sensitive to this negative side effect than the other restrictions considered.

NR_L performs badly in terms of plan quality. We believe that the reason for this behavior is that NR_L does not pre-evaluate the neighborhood elements. The restriction of NR_L is fast to compute, but it can easily exclude LA-graphs from which, within the same CPU-time, could reach a solution of better quality.

4.4 Inconsistency Selection

In this section we present results concerning a comparison of three inconsistency selection strategies for Walkplan. For lack of space we consider only the use of E_π. The results that we obtained using the other neighborhood evaluation functions are similar.

Table 6 shows the results of Friedman's test for Σ_{lev^-}, Σ_{rand}, and Σ_{lifo}, from which we can derive the following observation.

Observation 11. Walkplan *with E_π and Σ_{lev^-} performs statistically better than with E_π and either Σ_{rand}, or Σ_{lifo}, both in terms of CPU-time and plan quality.*

5 Conclusions

We have proposed some techniques for restricting the search neighborhood of Walkplan, and for selecting the next inconsistency to handle. These techniques have been experimentally evaluated together with some options for other important heuristic features of LPG (the noise value and the neighborhood evaluation function).

Our analysis is based on Friedman's statistical test, and it shows that the features investigated are very useful for improving the search in terms of required CPU-time or quality of the solutions. In particular, restricting the search neighborhood using any of the techniques that we have proposed is essential to exploit an accurate neighborhood evaluation function like E_π. Moreover, the analysis identifies options for the noise setting and the inconsistency selection that perform better than others (N_{dyn} and Σ_{lev-}, respectively).

References

[1] Blum, A., and Furst, M. Fast planning through planning graph analysis. *Artificial Intelligence*. 90:281–300. 1997.

[2] Do, M., and Kambhampati, S. Sapa: A domain-independent heuristic metric temporal planner. In *Proc. of ECP-01*. 2001.

[3] Do, M., and Kambhampati, S. Sapa: A scalable multi-objective heuristic metric temporal planner. JAIR 20:155–194. 2003.

[4] Friedman, M. The use of ranks to avoid the assumptions of normality implicit in the analysis of variance. JASA 32:675–701. 1937.

[5] Gerevini, A., and Schubert, L. Accelerating partial-order planners: some techniques for effective search control and pruning. JAIR 5:95–137. 1996.

[6] Gerevini, A., and Serina, I. Fast planning through greedy action graphs. In *Proc. of AAAI-99*. 1999. AAAI Press/MIT Press.

[7] Gerevini, A., and Serina, I. LPG: A planner based on local search for planning graphs with action costs. In *Proc. of AIPS-02*. 2002.

[8] Gerevini, A., Saetti, A., and Serina, I. Planning through stochastic local search and temporal action graphs in LPG. JAIR 20:239–290. 2003.

[9] Gerevini, A., Saetti, A., and Serina, I. An approach to temporal planning and scheduling in domains with predictable exogenous events. *Journal of Artificial Intelligence Research (JAIR)*, 25:187–231, 2006.

[10] Hoffmann, J., and Nebel, B. The FF planning system: Fast plan generation through heuristic search. JAIR:14:253–302. 2001.

[11] Hoffmann, J. Local search topology in planning benchmarks: an empirical analysis. In *Proc. of IJCAI-01*. 2002.

[12] Hoffmann, J., and Edelkamp, S. The deterministic part of IPC-4: An overview. JAIR, 24:519–579, 2005.

[13] Long, D., and Fox, M. The 3rd international planning competition: Results and analysis. JAIR 20:1–59. 2003.

[14] Pollack, M.E. and Joslin, D. and Paolucci, M. Flaw Selection Strategies for Partial-Order Planning. JAIR 6:223–262. 1997.

[15] Selman, B.; Kautz, H.; and Cohen, B. Noise strategies for improving local search. In *Proc. of AAAI-94*. 1994.

[16] Siegel, S.; Castellan J. Nonparametric Statistics for the Behavioral Sciences McGraw Hill. 1988.

[17] Ury, H., K. A comparison of four procedures for multiple comparison among means - pairwise contrast for arbitrary sample sizes. Tecnometrics 18:89–97. 1976.