A New Local-Search Algorithm for Forward-Chaining Planning

Andrew Coles, Maria Fox and Amanda Smith
Department of Computer and Information Sciences,
University of Strathclyde,
26 Richmond Street,
Glasgow, G1 1XH
email: firstname.lastname@cis.strath.ac.uk

Abstract

Forward-chaining heuristic search is a well-established and popular paradigm for domain-independent planning. Its effectiveness relies on the heuristic information provided by a state evaluator, and the search algorithm used with this in order to solve the problem. This paper presents a new stochastic local-search algorithm for forward-chaining planning. The algorithm is used as the basis of a planner in conjunction with FF's Relaxed Planning Graph heuristic. Our approach is unique in that localised restarts are used, returning to the start of plateaux and saddle points, as well as global restarts to the initial state. The majority of the search time when using FF's 'Enforced Hill Climbing' is spent using breadth-first search to escape local minima. Our localised restarts, in conjunction with stochastic search, serve to replace this expensive breadth-first search step. We also describe an extended search neighbourhood incorporating non-helpful actions and the 'lookahead' states used in YAHSP. Making use of non-helpful actions and stochastic search allows us to restart the local-search from the initial state when dead-ends are encountered; rather than resorting to best-first search. We present analyses to demonstrate the effectiveness of our restart strategies, along with results that show the new planning algorithm is effective across a range of domains.

1 Introduction

Two of the most established and successful planners in recent years, FF (Hoffmann & Nebel 2001) and LPG (Gerevini & Serina 2002), use local-search techniques to solve planning problems; albeit using different planning paradigms. FF performs forward-chaining state-space search, whilst LPG searches through a space of plans embedded in Graph-Plan (Blum & Furst 1995) planning graphs. The local search used in each of these planners also differs, with FF using a hill climbing approach relying on exhaustive search to escape local minima. The weakness of FF's hill climbing is that search is deterministic: if it fails to find a solution plan, exhaustive best-first search is performed from the initial state to solve the problem. LPG, however, uses a more sophisticated, stochastic, local search technique that allows the use of restarts.

These two successful planners have different strengths: FF has an intuitive and efficient approach to search, using a comprehensible forward-chaining approach to planning. This has lead to it being used as the basis for a large number of modern planning systems (MIPS-XXL (Edelkamp, Jabbar, & Nazih 2006), Marvin (Coles & Smith 2007), Macro-FF (Botea, Müller, & Schaeffer 2005)) and for the evaluation of many new ideas in planning. LPG, however, uses a more complex search paradigm, which is more difficult to build upon but does make use of more sophisticated local search techniques, thus avoiding the large amount of time spent doing exhaustive search in planning using FF.

In this paper we present a new local search algorithm for forward-chaining planning. This algorithm combines the strengths of FF and LPG: the intuitive and efficient forward-chaining search paradigm, and powerful heuristic of FF; and the principle of using stochastic local-search with restarts from LPG, avoiding the necessity to resort to exhaustive search.The stochastic local-search used in this work differs from that in LPG, both in its application to forward-chaining search and the local-search techniques themselves.

The aim is to produce a flexible framework for local-search forward-chaining planning, demonstrating how additional local search techniques can be applied and pushing the boundaries of current forward-chaining planners. To augment the proposed forward-chaining planning algorithm, we present a novel neighbourhood for use with it, based on applying neighbourhood sampling techniques to the applicable actions. The algorithms described are used as the basis for a planner, 'Identidem'. We present empirical results to show that the algorithms proposed are effective across a range of domains, and how the performance of Identidem is influenced by the use of restarts.

In the next section, an overview of the area of forward-chaining planning on which this work builds is presented. This is followed by details of the kernel of the new local-search algorithm for forward-chaining planning, and how it can be used to escape local minima. Having described this, the neighbourhood used for search with this algorithm is defined, followed by details of how *fail-bounded restarts* can be incorporated into search. Results are then presented indicating the performance of Identidem in a range of domains, and the effect of restarts on search performance. Finally, some concluding remarks are made and potential directions for future work are given.

2 Background

A classical AI planning problem can defined by a tuple $< A, I, G >$, where A is a set of actions with propositional effects and preconditions, I is the set of propositions in the initial state, and G is a set of propositions that hold true in any goal state. A solution plan to a problem in this form is a series of actions chosen from A that when applied transform I into a state of which G is a subset. Several possible search paradigms have been investigated for finding solution plans, one of which is *forward-chaining* planning, popularised in recent years by planners such as FF (Hoffmann & Nebel 2001), HSP (Bonet & Geffner 1995), Downward (Helmert 2006) and YAHSP (Vidal 2004). As in these planners, in this work it is assumed that A is a set of *ground actions*: the parameters of actions have all been bound to variables.

Forward-chaining planning can be seen as searching through a directed graph in which the nodes in the search space are each defined by a tuple $< S, P >$. S is a world state comprising of propositional facts and P is the plan (a series of ordered actions) used to reach S from the initial state I. In forward-chaining planning, search begins from the initial state, corresponding to a tuple $< I, [] >$. Edges between pairs of nodes in the search landscape correspond to applying actions to lead from one state to another. When an action a (chosen from A) is applied to a node $< S, P >$ the node $< S', P' >$ is reached, where S' is determined by the effects of a upon S, and P' is the plan resulting from appending the action A to P. Search proceeds through the search landscape, expanding states to generate successors by applying actions, until one is found in which G is satisfied.

As exhaustive search of this landscape is infeasible in all but the smallest search problems, heuristics are used to evaluate states in the search landscape in order to provide goal distance estimates. The Relaxed Planning Graph (RPG) heuristic is one such heuristic, introduced in FF. The RPG heuristic value is calculated for a given propositional state S by *relaxing* the actions in A by ignoring the delete effects and constructing a GraphPlan planning graph (Blum & Furst 1995) in which S is the initial fact layer. A non-optimal *relaxed plan*, a solution to the relaxed problem, can be extracted from the planning graph in polynomial time. In FF, the length of the relaxed plan is taken as the goal distance estimate; and, also, the actions chosen from the first action layer are used to determine *helpful actions*.

In conjunction with such a heuristic, a search algorithm is needed. FF introduced the Enforced Hill Climbing (EHC) algorithm, using it in conjunction with the RPG heuristic. EHC is an incomplete search algorithm based around classical gradient descent local-search , the key difference being when a local minimum is encountered in the search landscape, breadth-first search is used until a state with a strictly better heuristic value can be found. When such a state is found, it becomes the new global optimum and search continues from there. If breadth-first search terminates without finding a better successor, EHC has failed to find a solution—in these cases, FF restarts search using best-first search instead of EHC, thereby preserving completeness.

In FF, the choices of actions available when performing EHC is restricted to the helpful actions. The helpful actions are defined as any action from A which has at least one add effect in common with the actions chosen from the first layer of the relaxed planning graph. In the vernacular of local search, the helpful actions form the *neighbourhood* of each state. In restricting search to only considering helpful actions, a number of successors from each state are pruned, in practice providing a general reduction in the time taken to find a solution plan. The pruning is not completeness preserving, however, so it is only used in FF with EHC (which is already incomplete) and not with best-first search.

Algorithm 1: Local Search from Local Minima

```
Data: S_min - a local minimum
Result: S' - a state with a strictly better heuristic value
depth_bound ← initial_depth_bound;
for i ← 1 to iterations do
    for probes ← 1 to probes_at_depth do
        S' ← S_min;
        for depth ← 1 to depth_bound do
            S' ← a neighbour of S';
            if h(S') < h(S_min) then
                return S'
            end
        end
    end
    double depth_bound;
end
return abort search
```

3 Local Search for Local Minima

As discussed in Section 2, when a local minimum is encountered in EHC, breadth-first search is used to escape it. In such areas of the search landscape, the heuristic is uninformative: moves to states of equal or worse heuristic value must be made to reach a goal state. In many regards, it can be seen as the 'difficult' part of the search problem under the heuristic used.

As such, the local search planner Identidem developed in this work modifies EHC such that when a local minimum is encountered, local search is used to escape it; rather than relying on breadth-first search. An outline of the algorithm used is presented in Algorithm 1. A series of local search 'probes' from the local minimum forwards are performed, with their length restricted by a *depth bound*. The depth bound is first set according to a parameter, $initial\_depth\_bound$. If a better state cannot be found within the depth bound prescribed, the probe terminates and another starts from S_{min}. As probes of differing depths will be needed to escape different local minima, the depth bound is periodically increased each $probes\_at\_depth$ probes. In doing so, a range of probe depths will be considered. Eventually, if after several probe depths have been considered and a better state has not been found, the plateau is deemed to be inescapable. The number of probe depths to consider is determined by the $iterations$ parameter, and thus the maximum number of probes performed to escape a local minimum is given by $iterations * probes\_at\_depth$.

The choice of neighbour, at line 6, is made using roulette selection. The use of roulette selection allows non-determinism to be introduced into search: if search was deterministic, probes of the same depth bound would explore the same states, and performing repeated probes would not explore different parts of the search landscape. The size of the roulette wheel segments for each neighbour is based on their heuristic value, calculated as follows:

$$W_n = \left(\frac{1}{h_n}\right)^{\beta} \quad (1)$$

where h_n is the heuristic value of the state n, and β is a heuristic-bias parameter. The probability of a given segment n being chosen as the successor is given by:

$$P_n = \frac{W_n}{\sum_{x \in neighbourhood} W_x} \quad (2)$$

The heuristic-bias parameter β controls how much attention search pays to the heuristic values. In Identidem, β is decreased linearly from β_{max} to β_{min} as each of the probes made within the current depth bound are performed. In doing so, a range of heuristic biases are considered, and the diversification of the search landscape explored is increased above what would be the case if β was fixed.

This algorithm can be contrasted to the local search used in HSP (Bonet & Geffner 1995):

- Identidem attempts several probes to escape each local minimum; HSP attempts only one.
- Identidem uses an incrementally increasing depth bound; HSP fixes the depth bound to twice the heuristic value of the initial state.
- Identidem uses roulette selection to choose between *all* successors; HSP always chooses one of the *best* successors, breaking ties randomly.

4 Developing the Neighbourhood

In FF, when performing local search using EHC, the neighbourhood is fixed to contain all the helpful actions (see Section 2 for details). The helpful action neighbourhood has two key weaknesses:

- it can still contain a large number of actions;
- it renders search necessarily incomplete.

Identidem addresses each of these issues by introducing a neighbourhood sampling scheme to reduce its size, and by incorporating some non-helpful actions into the neighbourhood. This section describes how this is performed, and then details how 'lookahead' states are added to the neighbourhood.

4.1 Neighbourhood Sampling

From a given state, several helpful actions may be applicable. If there is a lot of self-similarity in the search space, where the application of different actions leads to states which are effectively equivalent, then considering each of the helpful actions may be detrimental to search performance. In particular, if most of the helpful actions are effectively equivalent and only one leads to an interesting different state, then the chances of this state being chosen by the roulette selection function described in Section 1 are small.

To try and characterise similarity between the helpful actions and consider only a representative sample of the actions, Identidem makes use of an action bucketing scheme. To do so, two criteria are used to characterise actions:

- the action type (e.g. `move`, `pickup`)
- the number of parameters the action has in common with the previous action applied.

The helpful actions are then placed into a square array of buckets, where the x and y coordinates of the bucket into which they are placed is determined by these two criteria. Having bucketed the actions, a neighbourhood of size η actions can be constructed by repeatedly choosing a non-empty bucket (at random) and removing one of the actions from it to add to the neighbourhood; terminating when η actions have been chosen or all the buckets are empty.

By choosing only a sample of size η, the number of successors to a state is reduced and the number of heuristic evaluations necessary when considering successors to a node in the search landscape is reduced. The trade-off, however, is that completeness is reduced. When used in conjunction with the local search algorithm to escape local minima, presented in Algorithm 1, the restarts performed help to address this: if a successor to a state is not considered the first time that state is visited, the successor may be considered if the state is revisited on a later restart.

4.2 Non-Helpful Actions

Whilst in the general case it may be beneficial to consider a neighbourhood containing helpful actions, there are cases where non-helpful actions are necessary to reach a goal state from a local minimum. As such, in Identidem, non-helpful actions are considered if search to escape a local minimum proves difficult. Specifically, with reference to the parameters for the local search algorithm in Algorithm 1:

- for the first $iterations/2$ probes at a given depth bound, only the helpful actions are considered;
- for the other probes at a given depth bound, both helpful and non-helpful actions are considered.

By restricting the first $iterations/2$ probes at a given depth bound to helpful actions, the benefits can still be realised on local minima which can be escaped using just helpful actions. The subsequent probes, with all actions, also allow local minima which require non-helpful actions to be escaped. In Identidem, this choice of whether to consider helpful actions or all actions is made prior to the neighbourhood sampling described in Section 4.1.

4.3 Lookahead

In the planner YAHSP (Vidal 2004), a polynomial-time procedure known as 'lookahead' is used to generate an extra successor to each non-goal state. For each propositional state in the search landscape, S, a relaxed plan R is built,

as in FF. The extra lookahead successor generated for each state S is derived from S by attempting to execute the actions in R in turn. In the general case, when executing R, actions will be encountered that have unsatisfied preconditions. These arise as a result of delete effects ignored in the relaxed problem but present in the original, non-relaxed problem. If a single action can be found to satisfy each unsatisfied precondition of an action, it is executed; otherwise, lookahead terminates and returns the state S_L reached from the execution of as many actions in R as could be applied.

Theoretically, the neighbourhood of a state could be extended to contain not just states reachable by the application of single actions, but states reachable by the application of multiple actions. The size of the neighbourhood would, however, be exponential in the size of the action chains considered, and the effect on search performance would be detrimental. By adding a single lookahead state S_L to the neighbourhood of a state, only one such multiple-action neighbour is included: one which appears to be heuristically useful. In Identidem, the extra state S_L is added to the neighbourhood alongside the η states chosen according to the approach described in Section 4.1.

5 Forward-Chaining Planning with Restarts

The local-search algorithms detailed in this paper so far have been concerned with escaping local minima, replacing the breadth-first search step within EHC. The effectiveness of such an approach relies on the assumption that local minima encountered are escapable, which is not always the case.

In EHC search, earlier action choices along the path to the global best state seen so far are never backtracked over. In directed search spaces, the effects of these actions cause search to be restricted to exploring limited section of the search landscape. Within this limited section of the search landscape, there may be few goal states, if any; making it impossible to find a goal state either due the non-existence of such a state, or due to the search time limit imposed. In the case where local minima are too difficult or impossible to escape, it is necessary to restart search from the initial state in order to find a solution plan.

In general, when performing local search with restarts, it is necessary to use a non-deterministic local search algorithm; otherwise, search will pursue the same path on each restart, and fail to find a solution if the first search attempt could not do so. In LPG (Gerevini & Serina 2002), a planner performing local search within plan graphs, the local search algorithms used are non-deterministic and, as such, LPG can make use of restarts. In FF, however, EHC search is deterministic and is only one attempt is made to use it to find a solution plan: if it fails, best-first search is used.

An alternative to the approach taken in FF is replace EHC with a non-deterministic gradient-descent local search algorithm. As the local search algorithm is non-deterministic, when search restarts from the initial state, it can be used again; rather than resorting to best-first search. This approach has been adopted in Identidem, and the remainder of this section will detail the local search algorithm used.

Algorithm 2: A Planner based on Algorithm 1

```
   Data: a planning problem < A, I, G >
   Result: a solution plan
1  S ← I;
2  while S does not satisfy G do
3  |  call Algorithm 1 to generate S';
4  |  if Algorithm 1 did not abort then
5  |  |  S ← S';
6  |  else
7  |  |  S ← I;
8  |  end
9  end
10 return plan to reach S
```

5.1 Non-deterministic forward-chaining search

Returning to Algorithm 1, it can be seen that in the general case the algorithm takes a state S and attempts to find a state S' with a strictly better heuristic value. This algorithm can be used as the basis for a local-search forward-chaining planning algorithm, presented in Algorithm 2. Algorithm 1 is repeatedly called to generate progressively better states, starting with the initial state I and terminating when one which satisfies G is found.

Each time Algorithm 1 is called, one of two cases arises: either the algorithm finds a strictly better successor; or it does not. In the former of these cases, the state found becomes the new state to search from; in the latter, the state to search from is reset to the initial state, thus triggering a restart. As search is non-deterministic, differing paths through the search landscape will be pursued each time it restarts. By using local search again to search from I, the necessity to resort to best-first search is avoided.

5.2 Fail-bounded restarts

The only criterion for restarting from the initial state considered so far in this paper is when search cannot escape a local minimum. It may be beneficial, however, to consider restarting not only when search has apparently failed, but also when search appears to be making slow progress. When performing forward-chaining planning using local search, decisions made along the path to the current local minimum restrict what portion of the search landscape can subsequently be reached. These decisions can potentially lead search into an area of the search landscape where the RPG heuristic provides little guidance and from which it is difficult to find a state with a better heuristic value. When using Algorithm 1, this is manifested as several probes being needed to escape each local minimum. If search enters such an area, restarting search from the initial state may lead to a solution faster than pursuing search from the current local minimum.

In Identidem, a *fail bound* is used to indicate when a difficult-to-escape area of the search landscape has been entered and a restart from the initial state should be triggered. This can be incorporated into the search algorithm as illustrated in Algorithm 3. Each time a probe to escape a local minimum reaches its depth bound without finding a state with a better heuristic value (line 11), a fail counter is incre-

Algorithm 3: Local Search with Restarts

Data: S_{min} - a local minimum, $failcount$, $failbound$
Result: S' - a state with a strictly better heuristic value, $failcount$ - updated fail count

```
depth_bound ← initial_depth_bound;
for i ← 1 to iterations do
    for probes ← 1 to probes_at_depth do
        S' ← S_min;
        for depth ← 1 to depth_bound do
            S' ← a neighbour of S';
            if h(S') < h(S_min) then
                return S', failcount
            end
        end
        increment failcount;
        if failcount = failbound then
            return abort search
        end
    end
    double depth_bound;
end
return abort search
```

mented. When this reaches a specified fail bound, search to escape the local minimum is aborted. The fail count is then reset to zero, and search restarted from the initial state.

To control the fail bound variable, a *fail sequence* is used (Beck 2006). In Identidem, the fail bound is initially set to 32 and doubled every three restarts; i.e. it follows the sequence $32, 32, 32, 64, 64, 64, 96, 96, 96, \ldots$. This fail bound sequences was chosen after evaluation of several alternatives (Coles 2007), and appears to give good performance in a range of domains.

6 Results

In this section, results are presented to indicate the performance of Identidem across a range of evaluation domains. As a basis for the implementation of Identidem, the code for Marvin (Coles & Smith 2007) was used, but with the features that distinguish it from FF disabled (such as its macro-action machinery).

Results for two configurations of Identidem are presented:

- Identidem (L) - the default configuration of Identidem, using lookahead (see Section 4.3);
- Identidem (NL) - Identidem, no lookahead.

As a benchmark, these are compared against:

- Marvin (L) - Marvin, configured to serve as an analogue of FF, but using lookahead;
- Marvin (NL) - Marvin, configured to serve as an analogue of FF;
- LPG-TD.

The two Marvin configurations serve as an implementation-neutral control for the two Identidem configurations. By having lookahead and no-lookahead configurations, it is possible to distinguish between performance differences due to the use of lookahead and performance differences due to using the modified search algorithm. Results for LPG-TD are included to compare the performance of Identidem to another local-search planner making use of restarts.

To perform the tests, a series of identical Linux machines were used, with 3.0GHz Pentium IV processors and 1Gb of physical memory. Each test was limited to 1800s of CPU time and 800Mb of memory. To control for the effects of the choice of random seed upon the performance of Identidem and LPG-TD, where search is non-determinism, each test was performed with random seeds 1,2,3,4 and 5, and the mean execution time taken from these five runs.

A range of evaluation domains were used, chosen from the benchmark suites for the Third and Fourth International Planning Competitions (Long & Fox 2003; Hoffmann 2005b). Results are presented for four of these, chosen to represent a range of search-space topologies (Hoffmann 2005a):

- Depots—a hybrid of a logistics problem with the classic Blocksworld domain, has an undirected search-space topology.
- Pipesworld-notankage—concerned with moving oil derivatives around a series of connected pipes, has a 'harmlessly directed' search-space topology.
- Pipesworld-tankage—an extension of the pipesworld-notankage domain with further constraints, also has a 'harmlessly directed' search-space topology, but in the practice problems are more difficult.
- FreeCell—the classic FreeCell card game, with a directed search-space, and unrecognised dead-ends under the RPG heuristic.

In the FreeCell domain, 20 full-sized problems were used (52 cards, 4 free cells), and in the other domains the IPC3 and IPC4 competition problem sets were used.

Various parameters of Identidem have been presented in this work. In the experiments performed these have been assigned values found through a series of small-scale experiments. With reference to Algorithm 3: $initial\_depth\_bound = 10$, $iterations = 5$, and $probes\_at\_depth = 60$. For the roulette selection, described in Section 3, $\beta_{max} = 1.5$ and $\beta_{min} = 0.5$. Finally, for the neighbourhood sampling described in Section 4.1, $\eta = 3$. Importantly, these values remain fixed for all experiments.

6.1 Overall Performance

Summaries of the results obtained are presented in Tables 1 and 2. Table 1 indicates how robust the planners are to the choice of random seed, presenting how many invocations of each planner in each domain were successful (out of a possible five per problem per domain). Table 2 indicates how robust the planners are to the choice of problem file, presenting how many of the problems could be solved with any one seed.

In the Depots domain, it can be seen with reference to the tables that several of the configurations perform strongly;

	Depots /110	Pipes notankage /250	Pipes tankage /250	FreeCell /100
Identidem (L)	**110**	**250**	**215**	**96**
Identidem (NL)	101	242	207	93
Marvin (L)	105	190	190	90
Marvin (NL)	90	180	160	75
LPG-TD	**110**	213	149	51

Table 1: Results Across Five Random Seeds

	Depots /22	Pipes notankage /50	Pipes tankage /50	FreeCell /20
Identidem (L)	**22**	**50**	**45**	**20**
Identidem (NL)	**22**	**50**	43	19
Marvin (L)	21	38	38	18
Marvin (NL)	14	36	32	15
LPG-TD	**22**	44	37	12

Table 2: Problems Solved with Any One Seed

particularly LPG-TD, Identidem (L) and Marvin(L). To discriminate between them, Figure 1 illustrates the time taken to solve each problem. As can be seen, LPG-TD solves the problems in the least time, with Identidem (L) approaching its performance on several problem files. Also, the search algorithms used in Identidem give rise to substantially better performance than those used in Marvin (and hence FF); both with and without lookahead. This indicates that the local-search algorithm proposed works well in this domain.

In the Pipesworld-notankage domain, it can be seen that Identidem (L) exhibits strong performance, solving all 50 problems with each of the 5 random seeds. Identidem (NL) exhibits good performance, also, but on the larger problems cannot always find a solution with each seed. The advantage of Identidem over the the Marvin configurations arises here due to the use of local-search to escape local minima. The local minima encountered during search in this domain often have long exit paths, and the local search algorithm considers long exit paths sooner than the breadth-first search used in Marvin. Further experiments reveal that neighbourhood sampling also contributes to the performance: increasing the sample size above 3 gives rise to worse performance.

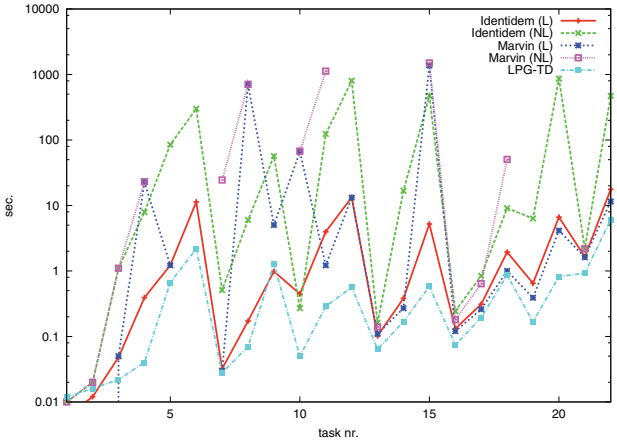

Figure 1: Performance in the Depots domain

In the Pipesworld-tankage domain, Identidem (L) does not perform as strongly as in the previous two domains, but nonetheless performs better than Marvin, indicating the search algorithm used in Identidem improves upon that in Marvin (and hence FF). In the FreeCell domain, Identidem (L) again provides the best performance of all the configurations tested, but does not solve all problems with all five seeds. The search guidance provided by the RPG heuristic in Pipesworld-tankage and Freecell is not as strong as in Pipesworld-notankage and Depots, so these result indicates that the new features of Identidem go some way towards compensating for poor heuristic guidance. In both of these domains, LPG-TD performs relatively poorly: the data indicates the average time taken to solve problems is consistently higher than in the other configurations.

Overall, across all domains, it can be seen that Identidem (L) exhibits strong performance. Furthermore, Identidem (L) and (NL) perform consistently better than Marvin (L) and (NL), respectively. This indicates that local search with restarts provides better performance in these evaluation domains than EHC (resorting to best-first search when that fails).

6.2 Evaluating the Effects of Restarts

So far, all configurations of Identidem tested have made use of restarts when a fail bound has been met (as described in Section 5.2). To evaluate what effect these additional restarts are having on search performance, additional tests were performed with a configuration of Identidem, Identidem (NR), in which the fail bound is set to infinity. The effect of this is that the only restarts that can be performed are those when a local minimum cannot be escaped within 300 probes.

One of the motivations for using restarts with non-deterministic search is to reduce the impact of the choice of random seed on search performance. Without restarts, if an unfortunate random seed is chosen, search will randomly make several decisions detrimental to performance, and take substantially longer to find a solution. Similarly, if a fortunate random seed is chosen, search will randomly make several decisions beneficial to performance, and take substantially less time than expected to find a solution. The effect of introducing restarts is to reduce this sensitivity of performance to the choice of random seed, by restarting search with a new random seed when it appears progress is slow.

To analyse the sensitivity of Identidem to the choice of random seed, a representive problem file was chosen from each of the four evaluation domains presented. The problems chosen were:

- Depots problem 11;
- Pipesworld-notankage problem 48;
- Pipesworld-tankage problem 33;
- FreeCell problem generated with the IPC3 generator: 52 cards, 4 free cells, random seed = 16.

The problems were chosen on the basis of the average time taken to solve the problem across 5 seeds: easier problems were discarded to avoid noise; harder problems were

	Runtime Distribution Statistics (s) (Mean, *Median*)							
	Depots		Pipes notankage		Pipes tankage		FreeCell	
Identidem	103.4	*83.5*	416.9	*188.5*	95.7	*69.4*	98.6	*90.4*
Identidem (NR)	92.6	*50.1*	739.0	*556.5*	177.6	*68.2*	99.5	*75.2*

Table 3: Summary Table: Mean and Median of Runtime Distributions for Identidem and Identidem (NR)

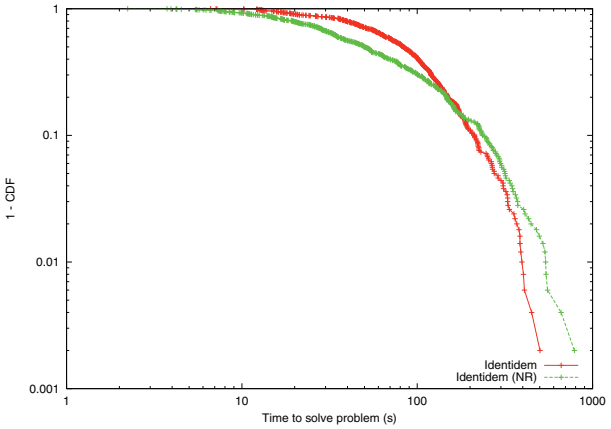

Figure 2: Runtime Distributions in Depots

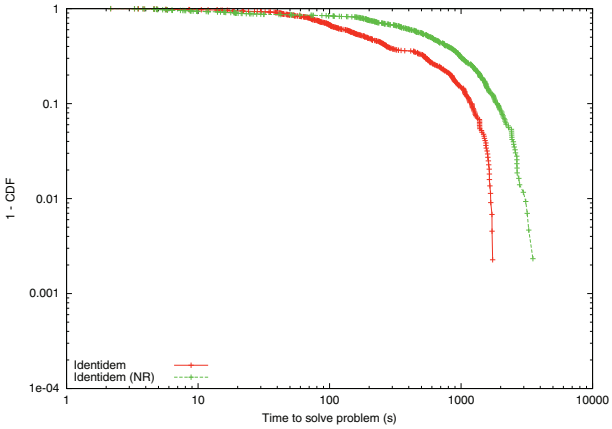

Figure 3: Runtime Distributions in Pipesworld-notankage

discarded to allow the experiments to complete in reasonable time. Having chosen a problem file from each domain, Identidem and Identidem (NR) were called for each problem with 500 random seeds (in the range $[1 \ldots 500]$). A summary table of the results is presented in Table 3.

In the Depots domain, it can be seen that the use of fail-bounded restarts increases the mean runtime of the planner by around 12%. However, the median is substantially closer to the mean. This indicates that the number of high-valued outliers in the the runtime distribution has been decreased, compared to Identidem (NR) which has a low median but proportionally high mean. This is confirmed by Figure 2, a plot of the cumulative distribution of runtimes on log-log axes (plotting 1-CDF against the run time). Here, it can be seen that the tail of the distribution for Identidem finishes at a lower time value than that for Identidem (NR). Also, for the 100 hardest seeds for each configuration (1 - CDF

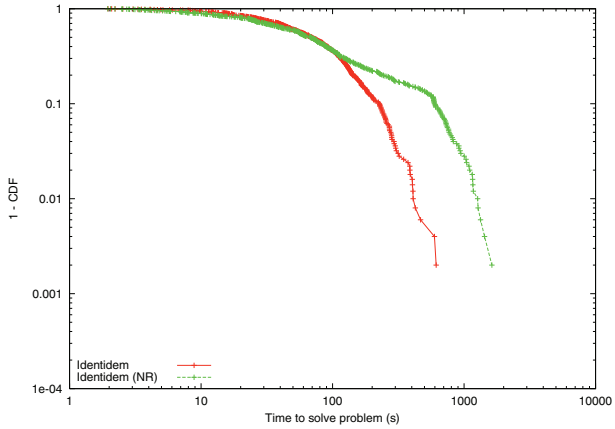

Figure 4: Runtime Distributions in Pipesworld-tankage

< 0.2) the runtimes for Identidem are lower than those for Identidem (NR). As such, it can be concluded that restarts decrease the sensitivity of Identidem to the choice of random seed in this domain, improving worst-case behaviour.

In the Pipesworld-notankage domain, the mean runtime is substantially reduced by the use of fail-bounded restarts. Unlike the Depots domain, the mean is also proportionally further away from the median in Identidem than in Identidem (NR). This suggests the runtime distribution has more high-valued outliers, but the plot of runtimes, shown in Figure 3, indicates this is not the case. The proportionally low median arises due to an irregularity in the runtime distribution curve for Identidem (occurring when (1 - CDF) is around 0.35), the effect of which is to increase the mean in proportion to the median. Overall, it can be concluded that restarts decrease the sensitivity to the random seed in this domain, improving the mean performance and worst-case behaviour.

In the Pipesworld-tankage domain, the median runtime is almost unaffected by the use of fail-bounded restarts, but the mean is substantially reduced. This indicates that fail-bounded restarts provide improved performance in this domain. The plot of the runtime distribution for Identidem (NR), shown in Figure 4 (along with that for Identidem), has an unusual shape with a protracted period of slow descent before sharply falling. Inspection of the output produced during search reveals that the vast majority of the results towards the tail of the Identidem (NR) distribution, after the point at which it begins falling sharply, correspond to where a restart was triggered due to a local minimum being inescapable within 300 probes. If these restarts were not performed, it is likely that the runtime distribution would have a shallower gradient at the tail, and Identidem (NR) would have a higher mean runtime. As such, in this domain, both

fail-bounded restarts and restarts triggered by the 300 probe limit are effective at improving both the mean and the worst-case performance of Identidem.

In the FreeCell domain, the mean runtime is almost unaffected by the use of fail-bounded restarts, with a slight improvement indicated. The median is brought proportionally far closer to the mean, however, indicating much better worst-case performance. This is confirmed by plotting the runtime distributions for Identidem and Identidem (NR), resulting in a graph similar to that for the Depots domain (see Figure 2). As such, the use of restarts can be seen to improve worst-case performance and marginally improve mean performance in this domain.

7 Conclusions

We have presented a local-search algorithm for forward-chaining planning. The idea focusses on the use of local search to escape local minima, repeatedly probing forwards from a local minimum using probes with increasing depth bounds until a strictly better state can be found. Restarts are used with the algorithm, restarting from the initial state when either the current local minimum cannot be escaped after a number of probes, or when the total number of probes performed for all local minima encountered exceeds a *fail bound*.

This local-search algorithm is used in a planner, Identidem, with FF's Relaxed Planning Graph (RPG) heuristic. The local search neighbourhood used differs from that in FF: YAHSP's 'lookahead' algorithm is used to introduce an extra successor to each state; and non-helpful actions are introduced when a number of probes at a given depth bound have failed to find a solution using only helpful actions. The resulting planner serves as a flexible framework, through which techniques from the local search community can be easily applied to forward-chaining planning. The resulting planner serves as a flexible framework, and forms the basis of a prototyping tool which can be used to apply state-of-the-art local-search techniques to forward-chaining planning, bringing together the two areas of research.

Results presented indicate that Identidem, as described, is an effective planner across a range of domains, achieving improved performance over a control planner (an analogue of FF) and, in the majority of domains, providing better performance than LPG-TD. Further analysis of the performance of Identidem with and without fail-bounded restarts indicates that the sensitivity of Identidem to the choice of random seed is reduced by the use of restarts; providing an improvement in worst-case performance in all observed cases, and an improvement in mean performance in most.

8 Future Work

Work on Identidem so far has focussed on reducing the time taken to find a solution plan. We hope to improve the plan quality by using optimisation techniques, finding better quality plans in terms of both the number of actions and the metric specified in the domain. To optimise with respect to a prescribed metric, we are also hoping to incorporate support for numeric state variables and actions with numeric preconditions and effects.

Preliminary work has already been performed investigating the use of Multi-Point Constructive Search (MPCS) (Beck 2006) with the described local-search planning algorithm, using the incomplete solutions found when restarts are triggered as the basis for the elite set (Coles 2007). We hope to extend this further, characterising the domain features that lead to MPCS giving rise to an improvement in search performance.

Acknowledgements

We would like to thank Chris Beck for his helpful discussions and insights concerning this work.

References

Beck, J. C. 2006. An Empirical Study of Multi-Point Constructive Search for Constraint-Based Scheduling. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS-2006)*, 274–283.

Blum, A., and Furst, M. 1995. Fast Planning through Planning Graph Analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Inteligence (IJCAI-95)*, 1636–1642.

Bonet, B., and Geffner, H. 1995. Planning as Heuristic Search: New Results. In *Proceedings of the Fifth European Conference on Planning (ECP-99)*, 359–371.

Botea, A.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *Journal of Artificial Intelligence Research* 24:581–621.

Coles, A. I., and Smith, A. J. 2007. Marvin: A Heuristic Search Planner with Online Macro-Action Learning. *Journal of Artificial Intelligence Research* 28:119–156.

Coles, A. I. 2007. *Heuristics and Metaheuristics in Forward-Chaining Planning*. Ph.D. Dissertation, University of Strathclyde.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-Scale Optimal PDDL3 Planning with MIPS-XXL. IPC5 Booklet, ICAPS 2006.

Gerevini, A., and Serina, I. 2002. LPG: A Planner based on Local Search for Planning Graphs. In *Proceedings of the Sixth International Conference of Artificial Intelligence Planning and Scheduling (AIPS-02)*, 13–22.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2005a. Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Hoffmann, J. 2005b. The Deterministic Part of IPC-4: An Overview. *Journal of Artificial Intelligence Research* 24:519–579.

Long, D., and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research* 20:1–59.

Vidal, V. 2004. A Lookahead Strategy for Heuristic Search Planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-2004)*, 150–159.