Adding Diversity to Classical Heuristic Planning

Carlos Linares López and **Daniel Borrajo**
Departamento de Informtica
Universidad Carlos III de Madrid
28911 - Leganés (Madrid), Spain
e-mail:{carlos.linares, daniel.borrajo}@uc3m.es

Abstract

State spaces in classical planning domains are usually quite large and can easily be extended to larger unmanageable sizes that do often exceed the capacity of many solvers. In this context, heuristic planning provides a powerful mean for solving rather difficult instances. However, it has been empirically observed that the performance significantly drops in the most difficult domains where the heuristic function induces large plateaus or unrecognized dead-ends. Therefore, a large effort has been invested in improving heuristic functions while only a few contributions have been made to the selection of the search algorithm. Thus, we review the choice of the search algorithm and show that some improvements are still feasible for various kinds of domains, especially the harder ones. In order to do so, we add diversity to the heuristic search in the hope that severe plateaus and/or dead-ends will be avoided or that shorter plans will be found. Also, with the aim of making the reporting of results more clear, a technique based on cumulative distribution functions is advocated.

Introduction

Heuristic search has been largely recognized as a general problem-solving technique for more than four decades and has been widely applied to many different domains which suffer from combinatorial explosion. Since planning tasks can be easily formulated like path-finding problems, single-agent search algorithms can also be applied to solve a wide variety of planning tasks. This has been shown at the various International Planning Competitions (IPCs) since 1998, at the IPC2, where HSP was shown to be competitive with state-of-the-art planners at that time (Bonet and Geffner 2001). Two years later, another heuristic planner, FF, showed an outstanding performance in the fully automated track (Hoffmann and Nebel 2001). Since then, heuristic planning has been applied to many other cases, not only classical planning, but also temporal planning, cost-based planning or conformant planning just to cite a few.

Originally, HSP was endowed with a heuristic hill-climbing algorithm. Since hill-climbing is an incomplete algorithm, the authors also tried another version, HSP2, with best-first search in order to overcome this difficulty. As a result, HSP2 was able to solve more problems or to solve them faster or with better quality (Bonet and Geffner 2001). On the other hand, FF introduced a variation in the selection of the local search algorithm, called enforced hill-climbing (EHC for short), which automatically resorts to a best-first search in case of failure. Actually, the enforced hill-climbing approach was recognized by the authors as an important component of the base architecture of FF (Hoffmann and Nebel 2001).

Still, if one compares the current performance of search algorithms in the heuristic search community with those in the heuristic planning community, a large gap is observed. Nevertheless, it is the case that the planning community can significantly profit from the latest advances in heuristic search.

In this work, we focus our attention on another idea explored in the heuristic search community: *diversity*, and discuss its applicability in classical heuristic planning problems. Diversity can be seen as the ability to switch to completely different paths during the search, therefore avoiding large errors in the heuristic function. In this context, improving search through diversity is expected to enhance significantly current state-of-the-art planners.

The paper is arranged as follows: next section briefly discusses different existing approaches to diversity, either depth-first or best-first, and justifies the convenience of a particular choice for solving planning tasks. Next, an experimental evaluation technique founded in statistical notions often seen in other fields is proposed for the reporting of results shown immediately after. The paper ends with some concluding remarks.

Diversity in planning

One of the first attempts to fight heuristic errors shall be acknowledged to Harvey and Ginsberg who introduced the Limited Discrepancy Search (LDS) (Harvey and Ginsberg 1995). The key observation was that, in most cases, failing to find a goal node can be explained by a small number of erroneous decisions along the current path. Therefore, LDS explores the left-most child at a given depth as suggested by the current heuristic function, i.e., with no discrepancies. If no solution is found, all leaf nodes which differ in just one decision point from the advice of the heuristic function are tried. The algorithm proceeds iteratively with increasing larger discrepancies until a solution is found. How-

ever, the original LDS reexamines the nodes considered by the previous iterations. Korf proved that this is unnecesary and showed how to avoid that many re-expansions introducing an improved version of the same algorithm known as Improved Local Discrepancy Search (ILDS) (Korf 1996). However, both LDS and ILDS have only been applied to either boolean problems or the number partitioning problem with two descendants per node, where a right and wrong choice can be easily identified. Further research is still needed to apply the same idea to a more general numerical setting with an arbitrary number of descendants.

Another interesting idea is Interleaved Depth-First Search (IDFS) (Meseguer 1997). It searches a path until a leaf node is found and if it is not a goal node, it starts another probe from the root node but following a different path, thus ensuring that different trajectories are traversed in every iteration.

However, preliminary experiments with the Improved Local Discrepancy Search algorithm in our research yielded discouraging results and, in general, it seems to us that algorithms adding diversity in a depth-first fashion are rather difficult to be applied to solve planning tasks efficiently.

More recently, K-Best-First Search (k) (KBFS) (Felner, Kraus, and Korf 2003) has been introduced and described as a generalization of Best-First Search. KBFS (k) expands the first k nodes in the OPEN list and only after merging the descendants in increasing order of the cost function into the OPEN list, the next iteration is started, until a solution is found. This algorithm provides a simple way to add diversity since at every iteration many nodes on completely different paths will be simultaneously expanded: the larger k, the more paths will be simultaneously considered. On the other hand, it does not impose any significant overhead and the processing time per node will remain the same. It shall be easy to notice that KBFS (1) = BFS and that KBFS (∞) results in the breadth-first search algorithm.

On the other hand, considering one of the most spread search algorithms in heuristic planning, EHC (also known as *First Iterative Improvement* (FII) in the specialized bibliography in Stochastic Local Search algorithms (Hoos and Stützle 2005)), its main drawback is that it deterministically pursues the same paths, trying to escape from local minima using a breadth-first search. An alternative consists of dynamically restarting the search, as in LPG (Gerevini and Serina 2002) or IdentIdem (Coles, Fox, and Smith 2007), but once again these algorithms implement irrevocable policies, as EHC —i.e., once it finds a successor which is better than the current node, it commits to it and it will never review its decision in the current search. Another option, implemented in KBFS (k), consists of simultaneously traversing different paths and switching from one to the other when a local minimum is encountered —recall that only when the first k nodes have been expanded and merged in the OPEN list, a new iteration is started.

In most domains, diversity with admissible heuristic functions would not be expected to provide any significant improvements since after expanding the best nodes, a minimum value of the heuristic function would be reached, so that chronological backtracking would eventually return to the descendants of the first node lying on the optimal path. In other words, the order in which nodes are expanded may not be relevant in those cases. However, this is true in domains like the sliding-tile puzzle or the Rubik's cube which have a unique goal state and are solved using admissible heuristic functions. In contrast:

- When solving planning tasks, heuristics are often based upon Relaxed Planning Graphs (RPGs), an idea originally introduced in FF (Hoffmann and Nebel 2001), which can result very easily in non-admissible estimates.
- Planning tasks can be realized by reaching any state among an arbitrarily large set of goal states, so that diversity serves for exploring simultaneously different paths to solve the same problem with different goal states.

Moreover, in any case, KBFS (k) with a sufficiently large width k can avoid traversing large plateaus: everytime a path reaches a plateau, its heuristic value will not decrease for an arbitrarily large number of iterations, until it finds an exit. Other paths currently stored within the best k in OPEN can use these iterations to try other alternatives, eventually displacing the paths that led to the plateau behind them in the OPEN list. This observation does not only apply to local minima, but also to unrecognized dead-ends since they always have to end in a valley.

In summary, it is not only true that KBFS (k) turns out to be a natural means for ranging from breadth-first search to BFS, but it also provides a simple scheme for pursuing different paths simultaneously. With a suitable parameter, KBFS (k) is expected to behave better than BFS, especially in the most difficult domains where the heuristic functions provide a rather poor assessment, leading to either local minima or, even worse, unrecognized dead-ends.

Experimental evaluation

The easiest class of search algorithms to compare against each other are deterministic *admissible* search algorithms (Pearl 1984), since in this case all solutions are known to have the same cost, being optimal. In addition, since admissibility necessarily implies completeness they are all expected to find a solution in a finite amount of time. Therefore, it usually suffices to compare run times for all problems.

However, if the algorithms to compare are either nondeterministic (so that different runs of the same algorithm on the same instance usually result in different solutions with different run times) or inadmissible (so that the algorithms are likely to find solutions with different quality with respect to the optimal cost), comparisons become far more difficult. The solution usually adopted in the first case (which is the typical scenario of the stochastic local search algorithms community) consists of computing empirical run-time and solution-quality distributions. When the algorithms to compare are inadmissible, as it usually happens in the planning community, most results consist of pictures with raw data (i.e., figures with run-time and/or solution-quality per solved instance) and some descriptive statistics including the number of solved problems, mean, median and standard deviation occasionally accompanied by the first and third quar-

tiles. However this technique cannot be deemed as very useful when comparing many search algorithms because:

- Different algorithms often solve a different number of instances. Taking only the subset of problems being simultaneously solved by the algorithms under consideration clearly lacks of a key observation: what problems are more difficult for what algorithms. Precisely those not being simultaneously solved make the real difference.
- Algorithms being compared always behave differently for different sorts of instances —distinguished by their difficulty or other parameters related to the problem size. Certainly, these observations cannot be easily derived from lines that mix up in the same picture and statistics that bind all samples.

In contrast, the approach followed in the study of stochastic local search seems more methodological and follows up more easily to other potentially interesting questions or for identifying general trends in the behaviour of the algorithms being compared. The discussion introduced herein is aimed at filling this goal.

While in the stochastic local search algorithms community, algorithms are run many times for solving the same instance, this does not make sense when applying deterministic algorithms like the ones considered in this paper for solving planning tasks. Thus, the following definition, slightly modified from that suggested by Hoos and Stützle (2005), is proposed for the comparison of heuristic search algorithms in planning:

Definition 1 *Consider a heuristic algorithm A for solving a finite and known set of problems in the planning domain $\mathcal{D}$, and let $P(RT_{A,\mathcal{D}} \leq t)$ denote the probability that A finds a solution for one of these instances in time less or equal than t. The Run-Time Distribution (or RTD, for short) of A on $\mathcal{D}$ is the probability distribution of the random variable $RT_{A,\mathcal{D}}$, which is characterized by the Run-Time Distribution function $rtd : \mathbb{R}^+ \mapsto [0,1]$ defined as $rtd(t) = P(RT_{A,\mathcal{D}} \leq t)$*[1]

These distributions are defined with respect to some resource bounds. Other variables might be memory usage, heuristic evaluations ... As a matter of fact, Plan-Length Distributions, $pld(t)$ will be also considered in the next section. Similar figures have been provided by other researchers —e.g., (Haslum, Bonet, and Geffner 2005)

Now, plotting one cumulative distribution function (abbreviated as *cdf*), for one algorithm against another, with respect to the same random variable, gives a more clear picture of the relative performance of each one. For example, consider Figure 1 which shows the Run-Time distributions, $rtd(t)$, of EHC and EKBFS (5) —to be discussed later. Firstly, instead of providing observed values of the random variable (e.g., time or plan length) per instance, the general behaviour is characterized by showing how the probability accumulates for given bounds. For instance, Figure 1 shows

[1]In short, while Hoos and Stützle consider the RTD for solving a single instance, an attempt is made here to characterize the RTD for a whole planning domain.

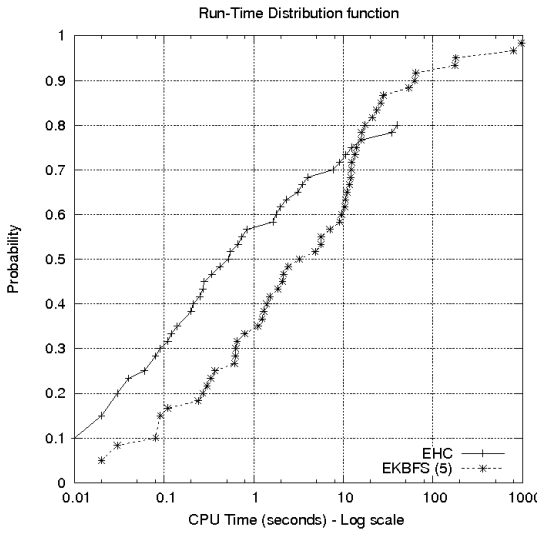

Figure 1: Run-Time Distribution of EHC and EKBFS (5) in *Freecell*

a steep increment in the number of problems solved around $t = 10$ for EKBFS (5). Secondly, the procedure is sensitive to the number of problems solved by each algorithm, since the probabilities are computed with respect to the size of the testset but it does not strictly rely on the number of problems solved by each algorithm, but how fast the random variable progresses with the instances being solved. In the preceding figure, a noticeable observation is that EKBFS (5) clearly solves more problems than EHC, since it gets far closer to 1. Thirdly, and more remarkably, two cumulative distribution functions can be compared one against each other by computing the intervals of the random variable where one *cdf* dominates the other, i.e., documenting the cross-over points where one is more likely to provide better results than the other with respect to the independent random variable been observed. Once again, the previous figure shows that the cross-over point happens somewhere near 10 seconds —more exactly, 15.85 seconds as shown in Table 4, page 5. Since EHC dominates the RTD of EKBFS (5) before this point, it can be concluded that the former is more likely to find solutions for those problems that do not take much time. Once this time has been elapsed, EKBFS (5) is more likely to find solutions and, in the end, it solves more problems.

In short, the experimental evaluation technique followed in the next section consists of the following steps:

1. Measure the independent variable of interest for each algorithm A under study for a bunch of problems in the same planning domain, $\mathcal{D}$. Since the algorithms considered herein are all deterministic, one run shall suffice.
2. Compute the probability distribution function, $P(X_{A,\mathcal{D}} = x)$, where $X_{A,\mathcal{D}}$ is the random variable under study and x are the values observed, so that $P(X_{A,\mathcal{D}} = x) = \frac{n_x}{n}$, where n_x is the number of times that $X = x$ was observed and n the overall number of

samples.

3. Accumulate the probability function to obtain the cumulative distribution function, $P(X_{A,\mathcal{D}} \leq x) = \sum_{y \leq x} P(X_{A,\mathcal{D}} = y)$.
4. Plot both cumulative distribution functions and compute the intervals within the range of the observed variable where either one *cdf* dominates the other or only one of them exists.

Nevertheless, the following shall be noted:

- It might happen that a significant variation in which problems are solved by different algorithms does not show up in the resulting distributions. Indeed, two different algorithms that solve different instances in exactly the same running time would have identical distributions.
- An underlying assumption is that the instances selected for making the evaluation are randomly distributed over the set of all instances of the domain.

Thus, special care shall be taken when selecting problems to guarantee that they are all of increasing difficulty and *essentially* different.

Results

In this section, KBFS (k) is compared with the current search strategies implemented in FF, namely: Enforced Hill-Climbing and Best-First Search. Although FF is not anymore the current state-of-the-art, it is a good representative, usually employed as the baseline planner at different IPCs, serving often as a starting point for building new planners. In FF, the EHC algorithm is accompanied by a series of important enhancements, mainly a *goal agenda* which decides, for every planning task, the order in which the goals shall be fulfilled, and *helpful actions*, a powerful forward pruning technique which aims at focusing in the relevant actions, ignoring the rest. In case that EHC with these enhancements fails to find a plausible plan, FF resorts to BFS turning off all these add-ons. Therefore, two versions of K-Best-First Search have been coded to make the comparisons: one where all these enhancements are turned off and another one which counts with the goal agenda and helpful actions. The first one is just termed KBFS (k) whereas the second one will be denoted as Enhanced KBFS (k) or EKBFS (k) for short. While EKBFS shall be compared with EHC, comparisons of KBFS will be done with respect to BFS. No diversified version of EHC is provided, since the goal of this research is not to test diversifying by itself but reviewing the search strategy implemented in FF with the aim of overcoming its main difficulties.

For making the experiments as meaningful as possible, various domains with rather different topologies have been selected. According to Hoffmann (2005) the complexity of various planning domains can be distinguished according to various structural properties in two directions: on one hand, domains where the heuristic function is likely to have local minima are proven to be more than a challenge in comparison with those where the maximal exit distance (roughly, the maximal number of steps to exit from a plateau) is bounded by some constant c; on the other hand, domains are classified in decreasing difficulty as follows: with unrecognized dead-ends, recognized dead-ends, harmless and finally undirected. We have selected domains falling in each case.

	Blocksworld (61)	Freecell (60)	Pipesworld (50)	Rovers (40)
EHC	51	48	4	**40**
EKBFS (5)	**58**/51	59/48	**23**/4	**40**/40
EKBFS (10)	57/50	59/48	21/4	**40**/40
EKBFS (50)	57/50	**60**/48	22/4	35/35
EKBFS (100)	56/49	59/48	**23**/4	34/34
BFS	45	**59**	**20**	**21**
KBFS (5)	44/41	58/57	**20**/19	20/20
KBFS (10)	47/43	58/57	**20**/19	20/20
KBFS (50)	50/45	**59**/58	18/17	**21**/20
KBFS (100)	**52**/45	**59**/58	17/17	**21**/20

Table 1: Number of problems solved (absolute/wrt) – hardest domains.

Since we are mainly interested in the hardest part of this taxonomy with regard to exit distance, the following domains where the FF heuristic estimation function is known to induce local minima have been selected: *Blocksworld-arm*, *Pipesworld*, *Rovers* and *Freecell*, everyone belonging to a different class, namely: undirected, harmless, recognized dead-ends and unrecognized dead-ends respectively.

Also, we examine whether E/KBFS (k) is still competitive in the easiest cases with regard to exit distance, where FF has previously shown an outstanding performance: *Logistics* and *Schedule*. The first domain falls into the category of undirected graphs and the latter has been tagged as having recognized dead-ends.

On the other hand, KBFS (k) and EKBFS (k) were tested with different widths, namely $k = \{5, 10, 50, 100\}$. All algorithms were given 30 minutes per instance and were allowed to use up to 1,8 Gb of RAM. Experiments were performed with a monoprocessor Intel Pentium 2.9 GHz running Linux. The code running EHC and BFS is FF 2.3 which has been compiled with `gcc` and the optimization flags turned on. KBFS and EKBFS were also programmed on the same version of FF, just replacing the search algorithms in `search.c` and leaving all the rest untouched.

Table 1 shows the number of problems solved in the more difficult domains selected with every search algorithm. The numbers shown for EHC and BFS denote the number of problems solved in each domain. However, for EKBFS (k) and KBFS (k) two numbers are given: the first one is the absolute number of problems solved by the algorithm specified whereas the second is the number of problems solved also by either EHC or BFS. Recall that while EKBFS shall be compared with EHC, comparisons of KBFS will be done with respect to BFS. Bold face typesetting has been selected to highlight the best results.

Blocksworld

61 problems have been taken from the IPC2 (2000), Track 1 (Typed). Figure 2 shows the RTDs of EHC vs EKBFS (5). Alternatively, Table 2 shows the intervals where either one

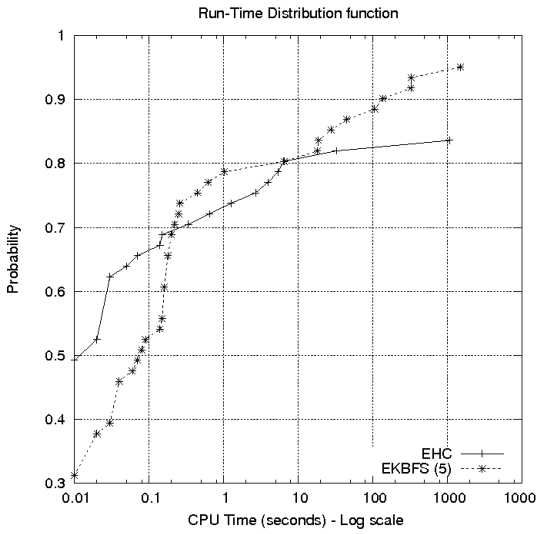

Figure 2: Run-Time Distribution of EHC and EKBFS (5) in *Blocksworld*

Interval	EHC	EKBFS (5)
[0.00 , 0.20]	(**42**, **0.02** , **0.03**)	(42, 0.06 , 0.07)
[0.22 , 5.41]	(6 , 2.39 , 2.01)	(**6**, **0.47** , **0.31**)
[6.43 , 6.55]	(1 , 6.43 , –)	(1 , 6.55 , –)
[18.25 ,1075.60]	(2 ,554.16,737.42)	(**8**, **126.39**, **131.97**)
[1510.05,1510.05]	(0 , – , –)	(**1**,**1510.05**, –)

Table 2: Blocksworld – Run-Time Distribution $(\#, \mu, \sigma)$

$rtd(t)$ dominates the other or only one exists. In all tables, data is shown as a triplet specifying the number of problems solved (#), mean (μ) and sampled standard deviation (σ) of the random variable under study. Once again, bold face typesetting has been used to highlight the dominating distribution.

As it can be seen, after solving the easiest instances, EHC is expected to be about 5 times slower than EKBFS (5). More remarkably, EHC is only able to solve one problem beyond 32 seconds and it takes it 1075 seconds. Beyond that point, EHC gets stagnated and does not solve more problems. In contrast, EKBFS clearly dominates EHC and solves problems far faster for those cases that take more than 18.25 seconds: whereas EHC solves only the two aforementioned cases, EKBFS (5) succeeds at solving 8 cases four times faster. Moreover, it does still succeed at solving another very difficult instance five minutes from the time bound, which was unsolvable for EHC.

Of the 10 problems not solved by EHC, only 2 failed because the time bound was reached. The other 8 were terminated because they took all the available memory, entirely different to EKBFS (5) who never took all the memory but failed on time. In other words, EHC suffers from severe stagnation in this domain, so that the algorithm never resorted to BFS for solving these problems. However, even in that case, a diversified version, KBFS (100) is better than

Interval	BFS	KBFS (100)
[0.00 , 1.58]	(**30**, **0.12** , **0.26**)	(30, 0.30 , 0.37)
[1.62 ,1021.17]	(13, 220.78 ,288.31)	(**22**, **140.62**, **270.90**)
[1282.22,1741.04]	(**2**,**1511.63**, **324.43**)	(0 , – , –)
[6.00 , 12.00]	(7 , 9.43 , 2.51)	(7 , 9.43 , 2.51)
[16.00 , 258.00]	(30, 98.73 , 60.99)	(**45**, **84.93** , **65.34**)
[266.00 , 348.00]	(**8** , **294.00** , **28.43**)	(0 , – , –)

Table 3: Blocksworld – Run-Time Distribution (above) and Plan-Length Distribution (below) $(\#, \mu, \sigma)$

Interval	EKBFS (5)	EHC
[0.00 , 0.01]	(0 , – , –)	(**6** , **0.01** , **0.01**)
[0.02 , 15.85]	(46, 4.12 , 5.02)	(**40**, **2.01** , **3.77**)
[16.01 , 40.00]	(**6** , **22.11** , **4.77**)	(2 , 37.52 , 3.51)
[54.39 ,976.23]	(**7** , **329.74**, **386.57**)	(0 , – , –)
	EKBFS(50)	EHC
[8.00 ,105.00]	(**56**, **40.86** , **24.97**)	(48, 42.67 ,26.68)
[106.00,132.00]	(**4** , **119.50**, **14.46**)	(0 , – , –)
	KBFS(100)	BFS
[8.00 , 9.00]	(5 , 8.40 , 0.55)	(5 , 8.40 , 0.55)
[12.00 ,111.00]	(**54**, **47.19** , **27.10**)	(50, 50.62 ,27.94)
[114.00,152.00]	(0 , – , –)	(**4** , **129.75**, **15.97**)

Table 4: Freecell – Run-Time Distribution (above), Plan-Length Distribution (middle and below) $(\#, \mu, \sigma)$

BFS in the number of problems solved (52 vs. 45) as shown in Table 1 and also in run time and plan length as shown in Table 3.

Freecell

The dataset with 60 problems from the IPC2 (2000, Typed) was used. In this case, EHC fails at solving 20% of the problems, as shown in Table 4. However, in all cases but one, EHC exits with no solution instead of getting stagnated. Hence, BFS is given an opportunity for solving those problems. Although EKBFS (5) is the fastest algorithm for the most difficult instances (as shown in Figure 1, page 3), if EKBFS (50) would have been employed (since this is the one providing the best solution quality, as shown in the mid-area of Table 4), it turns out that EKBFS (50) would have taken 1.4 times, in the mean, more time than the combination EHC–BFS, just for solving those problems[2]. But, as Table 4 shows, EKBFS (50) finds plans which are far shorter than those found by either EHC or BFS.

In short, using just one algorithm, EKBFS (50), all problems are solved usually faster than its counterpart, EHC, and yielding shorter plans. On the other hand, instead of resorting to BFS, resorting to KBFS (100) seems far more beneficial as shown in Table 4. Indeed, for plans having more than 12 operators, the mean length found by BFS is $(50 \times 50.62 + 4 \times 129.75)/54 = 56.48$, almost 10 steps larger than those found by KBFS in the same interval, 47.19.

[2]This data is not shown in the tables and has been computed by hand solely for those problems where EHC resorted to BFS.

Interval	BFS	EKBFS (5)
[0.00 , 0.01]	(**2**, **0.00** , **0.00**)	(2 , 0.01 , 0.01)
[0.02 , 0.02]	(1 , 0.02 , –)	(**2**, **0.02** , **0.00**)
[0.03 , 0.03]	(2 , 0.03 , 0.00)	(1 , 0.03 , –)
[0.04 , 0.05]	(1 , 0.04 , –)	(1 , 0.05 , –)
[0.09 ,1054.06]	(14,214.63,327.94)	(**16**, **37.06** , **44.43**)
[1484.92,1484.92]	(0 , – , –)	(**1**,**1484.92**, –)
	BFS	EKBFS (100)
[5.00 , 5.00]	(1 , 5.00 , –)	(1 , 5.00 , –)
[8.00 , 10.00]	(3 , 9.00 , 0.00)	(**4**, **8.75** , **0.71**)
[11.00 , 110.00]	(16, 36.80 , 23.23)	(**18**, **36.39** , **24.55**)

Table 5: Pipesworld – Run-Time Distribution (above) and Plan-Length Distribution (below) $(\#, \mu, \sigma)$

Pipesworld

For making experiments in this domain, the propositional dataset from IPC5 (2006) that contains 50 problems was used.

In this domain, both EKBFS (5) and EKBFS (100) are the algorithms, amongst all, that solve most problems, 23 in total, as shown in Table 1. More remarkably, they almost solve 6 times more problems than EHC. However, it is worth noting that in this domain, EHC usually exits almost immediately without finding any solution (most of the times in less than a second, though a few times it takes more than a minute and in one case, almost 25 minutes) thus, giving BFS an opportunity for solving those problems. In other words, EHC never experiences stagnation. This is not the case, however, for BFS that is only able to solve 20 cases, this is, 40% of the problems. It has been observed that the problems solved by EHC are also solvable by BFS, so that the final number of problems that can be solved with the combination EHC–BFS is exactly 20. Hence, just using one algorithm, either EKBFS (5) or EKBFS (100), 3 additional problems can be solved. Since EHC solves so few problems, a direct comparison is made between an EKBFS (k) algorithm and the combination EHC–BFS.

Figure 3 shows the Run-Time Distribution of BFS and EKBFS (5). As it can be seen, EKBFS (5) can be expected to be immediately faster (i.e., even for low values of the run time) than BFS. Table 5 shows the RTD and PLD of BFS versus EKBFS (5) and EKBFS (100) respectively. The vast majority of problems solved by BFS and EKBFS (5) fall within the interval of time $[0.09, 1054.06]$ seconds. Before that, the time differences could not be considered significant (because they happen before the first 0.1 second) and within the aforementioned interval, EKBFS (5) is almost 6 times faster than BFS. Also, EKBFS (100) provides solution lengths which are slightly better than those provided by BFS.

Rovers

In this case, the experiments were performed with the propositional dataset of the IPC5 (2006) that is made of 40 different planning tasks.

It turned out that in this domain, EHC succeeds at solving all problems, as shown in Table 1. Therefore, BFS is never invoked over any planning task[3]. Thus, a comparison is performed solely between EHC and EKBFS (k). Figure 4 shows how the difference between EHC and EKBFS (5) diminishes with time. The cross-over point happens at $t = 57.45$ seconds. From that point on, EKBFS (5) is able to solve the remaining cases (which are the most difficult ones) in 381.86 in the mean. However, EHC takes, just for solving these problems $(3 \times 279.22 + 4 \times 1088.26)/7 = 741.52$. In other words, while EHC seems better suited for those problems that are expected to take less than a minute, EKBFS is far better suited for solving the most difficult instances. Besides, EKBFS (10) is able to get plans which are slightly better in all cases as shown in Table 6.

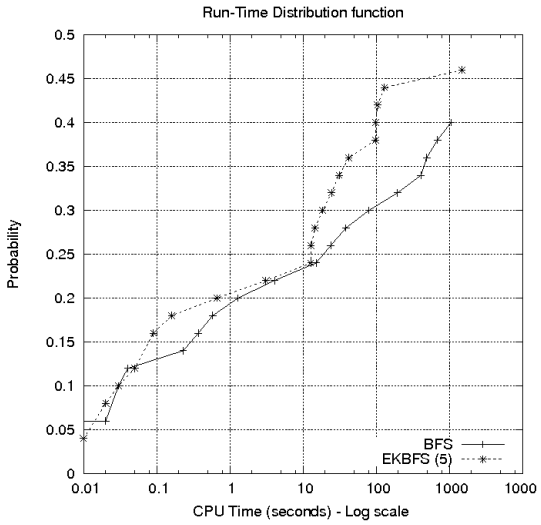

Figure 3: Run-Time Distribution of BFS and EKBFS (5) in *Pipesworld*

Interval	EKBFS(5)	EHC
[0.00 , 57.45]	(33, 4.71 , 12.30)	(**33**, **3.39** , **9.97**)
[73.75 , 652.65]	(**7** , **381.86** , **245.23**)	(3 , 279.22 , 281.27)
[784.77,1397.37]	(0 , – , –)	(**4** , **1088.26**, **264.36**)
	EKBFS(10)	EHC
[8.00 , 10.00]	(3 , 8.67 , 1.15)	(3 , 8.67 , 1.15)
[11.00 , 22.00]	(**4** , **17.50** , **4.65**)	(4 , 18.00 , 3.74)
[26.00 , 28.00]	(**2**, **27.00** , 1.41)	(2 , 28.00 , 0.00)
[31.00 , 41.00]	(**6** , **34.33** , **3.88**)	(5 , 35.40 , 2.70)
[42.00 , 380.00]	(**25**, **134.40**, **97.23**)	(26, 141.84 , 107.84)

Table 6: Rovers – Run-Time Distribution (above) and Plan-Length Distribution (below) $(\#, \mu, \sigma)$

Logistics

The testset used in Track 1 in the IPC2 (2000), Typed, consisting of 79 problems has been used. Table 7 shows the

[3]Nevertheless, the results shown in Table 1 indicate that KBFS (k) is still competitive with BFS. In addition, KBFS (5) solves problems at a similar pace than BFS while KBFS (100) can find plans which are moderately shorter.

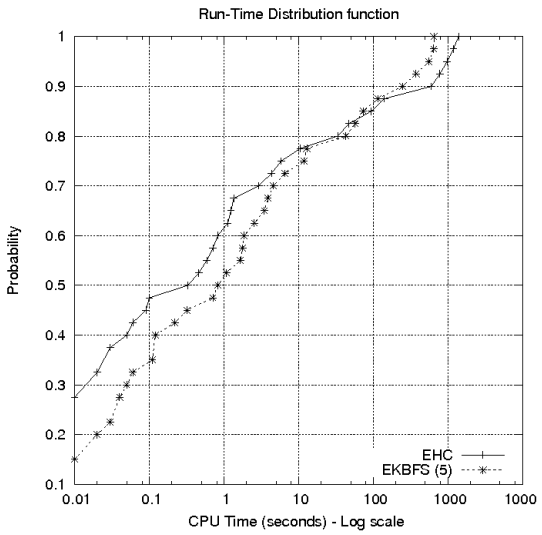

Figure 4: Run-Time Distribution of EHC and EKBFS (5) in *Rovers*

	Schedule (84)	Logistics (79)
EHC	**84**	**79**
EKBFS (5)	**84**/84	77/77
EKBFS (10)	**84**/84	77/77
EKBFS (50)	**84**/84	78/78
EKBFS (100)	**84**/84	75/75
BFS	32	58
KBFS (5)	35/32	60/58
KBFS (10)	**36**/32	**63**/58
KBFS (50)	34/26	62/54
KBFS (100)	33/24	55/49

Table 7: Number of solved problems (absolute/wrt) – easiest domains

number of problems solved in the easiest domains of the taxonomy suggested by Hoffmann (Hoffmann 2005). The data are summarized in the same way as in Table 1.

As it can be seen, EHC suffices to solve all problems in the Logistics domain, so that the results with BFS are given only for the sake of completeness. Moreover, the algorithm is far faster than any EKBFS (k) variation. It has been observed that the larger k, the slower the resulting execution of EKBFS (k). Table 8 shows the differences in run time with the fastest version of EKBFS (k), EKBFS (5). Clearly, EKBFS (k) behaves worse for the most difficult instances and takes longer for solving them. In fact, the difference in time does not pay off for the improvements on quality. Table 8 shows that the differences in solution quality are moderate though EKBFS (50) shows a clear trend to improve on the harder problems. Indeed, it has been observed that in the last intervals, EKBFS (50) is able to output plans which are a few tens shorter.

An explanation to this phenomenon is that while the FF

Interval	EHC	EKBFS (5)
[0.00 , 15.80]	(**79**, **1.20** , **2.43**)	(58, 2.04 , 3.18)
[16.35 ,558.60]	(0 , – , –)	(**19**, **105.50**, **141.62**)
	EHC	EKBFS (50)
[8.00 , 42.00]	(16, 25.69 ,10.04)	(16, 25.69 , 10.04)
[44.00 , 45.00]	(**2**, **44.00** , **0.00**)	(2 , 44.50 , 0.71)
[46.00 ,207.00]	(48, 132.77 , 48.31)	(**48**, **127.92**, **46.21**)
[209.00,211.00]	(1 , 209.00 , –)	(1 , 211.00 , –)
[213.00,223.00]	(3 , 221.00 , 2.00)	(**3** , **213.67** , **0.58**)
[224.00,226.00]	(1 , 224.00 , –)	(1 , 226.00 , –)
[228.00,250.00]	(6 , 241.50 , 6.28)	(**7** , **235.71** , **7.80**)
[251.00,281.00]	(**2** , **266.00**, **21.21**)	(0 , – , –)

Table 8: Logistics – Run-Time Distribution (above) and Plan-Length Distribution (below) $(\#, \mu, \sigma)$

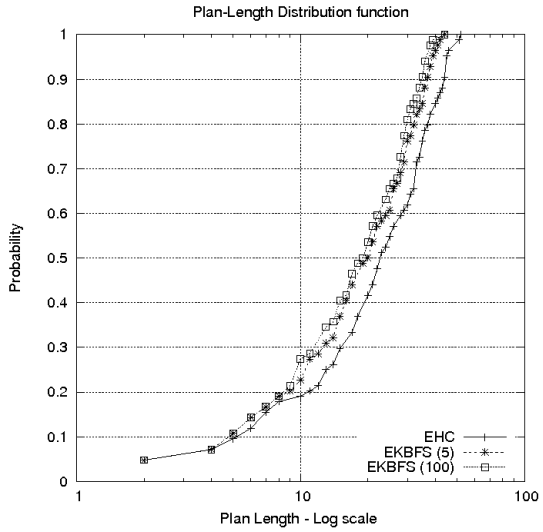

Figure 5: Plan-Length Distribution of EHC and EKBFS (50) in *Schedule*

heuristic is very well-informed in this domain (so well that there are no significant big plateaus and no dead-end at all), the EKBFS (k) algorithm is often losing sight of the goal switching constantly among paths. This effect is similar to *shuffling* the OPEN list and results in a poorer performance. Nevertheless, the problems that remained unsolved failed on time. This is, the OPEN list never grew as to exceed the available memory.

Schedule

The Typed dataset from IPC2 (2000) with 84 problems has been used in the last set of experiments. As it is shown in Table 7, EHC and all EKBFS (k) variations succeed at solving all cases. Hence, FF would have never resorted to BFS to solve any planning task on this domain. Nevertheless, the results on the number of problems solved by BFS and KBFS (k) are given for the sake of completeness —note that KBFS (k) do always solve more problems than BFS because Schedule has dead ends so that diversity helps.

As it was observed in the preceding domain, EHC turned

out to be the fastest algorithm in this domain. While EHC solves all problems in 0.07 seconds in the mean, EKBFS (5) takes 0.43 seconds also in the mean. However, the same version of EKBFS significantly and consistently improves the solution quality of all plans found by EHC. Indeed, this is true for all values of k tested. Figure 5 shows the Plan-Length Distribution for $k = 5$ and 100. Other values of k fell between these two curves.

Conclusions and future work

The search strategy implemented in FF starts by using EHC. While this algorithm is known to have an outstanding behaviour in the easiest domains, it usually either gets stagnated or it just exits with no solution in the most difficult domains and, in case it finds any solution, quality can easily be improved many times. The explanation for this behaviour can be succintly summarized as follows: first, because the only way for EHC to escape from a plateau consists of fully traversing it so that stagnation can easily happen; and secondly, because it implements irrevocable selection policies so that once a node has been chosen the decision is never reviewed though any of the siblings might have led to a better solution. Other researchers have tried to overcome these difficulties suggesting the use of random restarts along with other ideas (e.g., LPG (Gerevini and Serina 2002) and IdentIdem (Coles, Fox, and Smith 2007)). In this paper, diversity is proposed, instead, as a natural mean for overcoming the same problems. After considering the possibility of using depth-first algorithms that implement the idea of diversity, they have been discarded either because further research is still needed or because our preliminary results were discouraging. However, since it has been empirically observed that memory is not the limiting factor (even with time bounds of 1800 seconds/problem), an approach based on best-first search has been suggested instead.

On the other hand, because deriving conclusions from the raw data generated by the execution of a large number of algorithms (10 different algorithms were compared in this work) on a large set of problems (in total, 374 problems in 6 different domains) would be very cumbersome, or even impossible, a different procedure has been advocated here. Computing cumulative distribution functions and documenting the cross-over points where one *cdf* dominates another seems very useful for deriving conclusions on the general behaviour of the algorithms with regard to a wide variety of random variables with respect to some resource bounds.

In the experiments, it has been observed that while both EHC and BFS can suffer from severe stagnation (e.g., the Blocksworld domain), EKBFS (k) and KBFS (k) do solve more problems. In the hardest domains, there are always values of k that significantly improve on the run time and/or the plan length. More specifically, low values of k seem better suited for finding solutions faster though the higher values of k usually led to better solution quality. With regard to the easiest domains (where the minimal exit distance is bounded), EHC clearly wins being so fast that it leaves no room for improvement, whereas the presence of dead-ends in the Schedule domain served for significantly and consistenly improving the solution quality of almost all problems tested.

Lines currently undergoing research include automatic ways to dynamically adjust the value of k and to experiment with other planners, more remarkably with LAMA (Richter and Westphal 2008).

Acknowledgments

The reviewers of this paper significantly improved the quality of this work by making an outstanding review both in terms of quality and quantity. This work has been partially supported by the Spanish MICINN under project TIN2008-06701-C03-03

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Coles, A.; Fox, M.; and Smith, A. 2007. A new local-search algorithm for forward-chaining planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS-07)*, 89–96.

Felner, A.; Kraus, S.; and Korf, R. E. 2003. KBFS: K-best-first search. *Annals of Mathematics and Artificial Intelligence* 39:19–39.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, 13–22.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 607–613.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 1163–1168.

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. 2005. Where "ignoring delete lists" works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Hoos, H. H., and Stützle, T. 2005. *Stochastic Local Search: Foundations and Applications*. San Francisco, CA, USA: Morgan Kaufmann.

Korf, R. E. 1996. Improved limited discrepancy search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 286–291.

Meseguer, P. 1997. Interleaved depth-first search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1382–1387.

Pearl, J. 1984. *Heuristics*. Reading MA: Addison-Wesley.

Richter, S., and Westphal, M. 2008. The LAMA planner using landmark counting in heuristic search. In *The International Planning Competition (IPC-08)*.