HAL
open science

Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)

Joerg Hoffmann

To cite this version:

Joerg Hoffmann. Everything You Always Wanted to Know About Planning (But Were Afraid to Ask). 34th Annual German Conference on Artificial Intelligence - KI'11, Oct 2011, Berlin, Germany. pp.1-13, 10.1007/978-3-642-24455-1_1 . hal-00641851

HAL Id: hal-00641851

https://inria.hal.science/hal-00641851v1

Submitted on 16 Nov 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Everything You Always Wanted to Know About *Planning* (But Were Afraid to Ask)

Jörg Hoffmann

INRIA,
Nancy, France
joerg.hoffmann@inria.fr

Abstract. Domain-independent planning is one of the long-standing sub-areas of Artificial Intelligence (AI), aiming at approaching human problem-solving flexibility. The area has long had an affinity towards playful illustrative examples, imprinting it on the mind of many a student as an area concerned with the rearrangement of blocks, and with the order in which to put on socks and shoes (not to mention the disposal of bombs in toilets). Working on the assumption that this "student" is you – the readers in earlier stages of their careers – I herein aim to answer three questions that you surely desired to ask back then already: *What is it good for? Does it work? Is it interesting to do research in?* Answering the latter two questions in the affirmative (of course!), I outline some of the major developments of the last decade, revolutionizing the ability of planning to scale up, and the understanding of the enabling technology. Answering the first question, I point out that modern planning proves to be quite useful for solving practical problems - including, perhaps, yours.

Disclaimer. *This exposition is but a little teaser to stimulate your appetite. It's far from a comprehensive summary of the field. The choice of topics and literature is a willful sample according to my personal interests, and of course it over-represents my own contribution. The language and style are sloppy. On the positive side, the paper is entertaining and easy to read (or so I hope).*

1 Planning? What's that?

Planning is the problem of selecting a goal-leading course of actions based on a high-level description of the world. One could fill books (and that's what people have done [12]) with the different variants of what exactly this means. Herein, we will make do with the most canonical definition:

Definition 1. *A **planning task** is a 4-tuple $\Pi = (\mathcal{V}, \mathcal{A}, s_0, s_\star)$ where:*
(i) $\mathcal{V} = \{v_1, \ldots, v_n\}$ *is a set of finite-domain **state variables**.*
(ii) $\mathcal{A}$ *is a set of **actions** a, where each a is a pair (pre_a, eff_a) of partial variable assignments called **preconditions** and **effects**.*
(iii) s_0 *is a complete variable assignment called the **initial state**, and $s_\star$ is a partial variable assignment called the **goal**.*

I omit the straightforward formal semantics associated with this syntax. Suffice it to say that the task is associated with its **state space**, the directed graph of all **states** (complete variable assignments), with an arc from s to s' iff there exists $a \in \mathcal{A}$ so that pre_a complies with s, and changing s according to eff_a yields s' (note that eff_a over-writes previous variable values; we don't distinguish "adds" and "deletes"). A **plan** is a path leading from s_0 to a state complying with $s_\star$. The plan is **optimal** if it is a shortest such path.

From a computer science perspective, planning is just one formalism succinctly describing large transition systems, similar to automata networks or Turing machines. Trivially, planning is hard (**PSPACE**-complete in our case here). What makes planning special is its purpose in AI. We'll get to that in a moment. A particularly special aspect of planning are the illustrative examples that have long dominated the field. Fig. 1 gives two of the most emblematic scenarios students are confronted with when first hearing of this beautiful area.

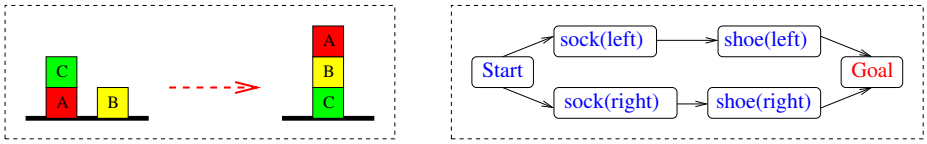

Fig. 1: This is planning (?)

Fig. 1 (left) actually is the only scientific object in the planning area that has ever been adorned with its inventor's name ("The Sussman Anomaly"). Isn't that depressing? I wouldn't insist on having a "Hoffmann's Theorem", but surely *someone* did more interesting planning stuff than that, sometime. Anyway, Fig. 1 (right) is perhaps even more counterproductive. Few people will appreciate the importance of AI technology in putting on their socks and shoes. As for the widely used benchmark called "Bomb in the toilet" (inner workings omitted for the sake of sanity), I guess the best that can be said about it is that it is not associated with an emblematic illustration. So, without any further ado:

2 What is it good for?

Back in the day of GOFAI[1], planning got started having in mind to approach human problem solving flexibility. One may argue whether or not, to accomplish this, it makes sense to assume mathematically precise world models as the planning input. But, in the spirit of the 21st century, let's just forget philosophy and be rock-bottom pragmatic: How can we earn money with planning?

What characterizes planning research is the attempt to create *one* planning solver that will perform sufficiently well on *all* possible domains (inputs). That will never work out (the problem is hard), but there's been tremendous algorithmic progress in the last decade. This will pay off for you if either:

[1] "Good Old-Fashioned AI", cf. `http://en.wikipedia.org/wiki/GOFAI`

(A) **Your problem is subject to frequent change.** If you implement your own solver, you'll have to keep adapting it. Using planning, it suffices to change the declarative planning model.
(B) **It would be costly to implement your own solver.** Unless your problem is quite easy, making a solver will cost time+money. Writing the planning model is typically much less effort.

In other words, *planning is a cost-effective method for software engineering.* It's a model-based approach. The planning model serves as a high-level programming language decoupling the problem from its solution.

Scenario (A) is a classical argument made in favor of planning, and has long been the reason for its investigation in the context of space travel, where "adapting the solver" can be problematic due to circumstance (watched Apollo 13, anyone?). In the meantime, industrial applications exist also down on this earth. For example, at Xerox a planning architecture is successfully being employed to flexibly control highly configurable printing systems [36]. At SAP, planning fits seamlessly into a major model-driven software engineering effort aimed at effective change management [24].

Scenario (B) is a tad unconventional, but is quite real and may indeed be the "killer app" for planning technology: *planning is a quick hack to get things up and running.* Rapid prototyping, in other words. Planning people don't usually think of themselves in these terms, and this point was driven home to me only quite recently, in a conversation with Alexander Koller who has been using planning for natural language sentence generation [27,28]. When I asked him why he doesn't develop a specific solver that could be more effective – a typical if not emblematic planning question – his answer was: "well, that's not the research problem I'm interested in; the planner works reasonably well, and I don't want to spend the time working out an alternative".

Shortly afterward, I actually found myself being better friends with the development department than with the research department of a company I was working with. "If pigs could fly", you might think right now; but it's true. The respective punchlines were "oh, but we could come up with something that works much better than the planner" (research department) vs. "yeah, maybe, but you don't know when and my product deadline is next month" (development department). The company – Core Security Technologies, `http://www.coresecurity.com/` – now employs a variant of my Metric-FF planner [19] in their product, serving to intelligently select possible attacks in regular security checks against a client network [29]. Note that, both for Alexander and Core Security Technologies, the "quick hack" actually turned into a long-term solution!

Generality is of course not a unique virtue of planning. SAT and CP, for example, are competing model-based approaches. Planning has potential advantages in modeling, since planning models are very high-level and can thus be more human-readable and easier to modify. In terms of solving, the approaches are complementary. Generally speaking, one can expect constraint-based methods to have the edge in combinatorial optimization. But for obtaining reasonable solutions quickly, in particular in applications where finding a feasible solution is already hard, a planner might be the better choice. Let me outline why.

3 Does it work?

The curse of planning is dimensionality, aka the state explosion problem. The major news from the last decade is that we now have techniques able to tackle this problem fairly well, judging at least from performance in an ever-growing set of established benchmarks from the International Planning Competition. We now have some 40 domains and well over 1000 instances, all encoded in the same common language PDDL [31,10].

Like SAT, planning experienced a major scalability breakthrough in the last decade. Where in SAT this is largely thanks to clause learning techniques, in planning it's largely thanks to heuristic search. The reader who has followed the planning literature just a little bit will be familiar with this development. Let me say a few words for the benefit of the reader that didn't; everybody else may skip to below Fig. 2. A major player here is this simple definition:[2]

Definition 2. *Let $\Pi = (\mathcal{V}, \mathcal{A}, s_0, s_\star)$ be a planning task. An action sequence $\langle a_1, \ldots, a_n \rangle$ is a* ***relaxed plan*** *for Π iff, with $s_0^+ = \{(v, c) \mid s_0(v) = c\}$ and $s_i^+ = s_{i-1}^+ \cup eff_{a_i}$, we have that $s_\star \subseteq s_n^+$ and that $pre_{a_i} \subseteq s_{i-1}^+$ for $1 \leq i \leq n$. The relaxed plan is* ***optimal*** *if n is minimal; the* ***relaxed plan heuristic*** *is then $h^+(\Pi) = n$.*

Many planners use this kind of heuristic in a forward state space search, where for every state s visited during search on planning task $\Pi = (\mathcal{V}, \mathcal{A}, s_0, s_\star)$, the planner evaluates the relaxed plan heuristic for the "local" task $\Pi_s = (\mathcal{V}, \mathcal{A}, s, s_\star)$. Hence a relaxed planning task is solved in every individual search state.

The relaxed plan heuristic assumes that, whenever a variable changes its value, it obtains the new value *and* retains the old one. In other words, we never over-write previous values; what we achieved once will remain true forever. That is, of course, not the case in the real world, or in any planning application known to man. Just to illustrate, the relaxed plan heuristic for the n-discs Towers-of-Hanoi problem is n, under-estimating the actual plan length by an exponential number of steps because it effectively ignores all the interactions between different disc moves. Despite this, it turns out that h^+ delivers excellent search guidance for many planning domains. We'll get into a little more detail on this below (Section 4.2); to give an intuition why h^+ could be informative, perhaps a few examples are helpful.

If the planning domain is graph distance – finding a shortest path in a graph – then h^+ is exact (because shortest paths "never walk back"). The same goes for (discrete) path finding in the presence of obstacles. In the sliding-tiles puzzle, h^+ strictly dominates Manhattan distance. In TSP, relaxed plans are equivalent to the minimum spanning tree approximation. If a planning task contains some of these kinds of structure, then these reasonable approximations will be captured.

Obviously, h^+ is lower-bounding (admissible) and consistent. It first appeared in the literature in 1994, as a footnote of an investigation of planning tractability [6] (describing it as an uninteresting sub-class). Its use for search guidance was

[2] In a logics-based representation where action effects are positive or negative, this definition is equivalent to ignoring the negative effects.

first proposed 2 years later [30,5], and was proliferated during the last decade to become *the* most successful technique for finding plans effectively.

Computing h^+ itself is actually hard, but upper bounds can be computed easily [2,23], essentially by generating *some* (not necessarily optimal) relaxed plan. These bounds tend to be close to h^+ in many benchmarks [20], but do not provide any theoretical guarantees so these heuristic functions are in general not admissible. What we can use them for is satisficing (greedy) search for plans. They also provide us with natural action pruning techniques, simply by giving a preference to those actions that are used by the relaxed plan [23,32].

Virtually every winner of the satisficing tracks at the planning competitions since the year 2000 makes use of the described techniques in one way or another, within one or another instance of heuristic search (e.g., [23,11,33]). To give an impression of the scale of the performance boost, Fig. 2 compares the state of the art prior to the year 2000 – which was based essentially on Graphplan [1] – to my FF planner [23] that dominated the 2000 competition.

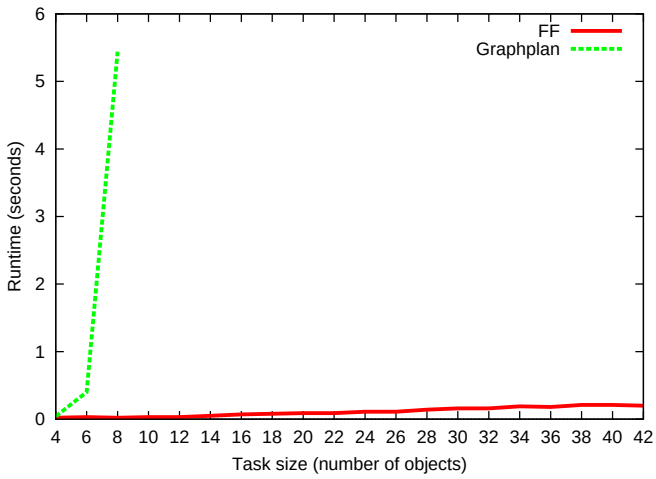

Fig. 2: Good-bye, Graphplan!

The benchmark domain underlying Fig. 2 is extremely simple (robot transports n objects from A to B), but does not exaggerate the point. Whereas previously our best planners exploded with tiny size parameters already, we could suddenly scale up to instances many many times larger, even obtain near-linear runtime behavior as in Fig. 2. Current heuristic search planners comfortably solve instances with astronomically large state spaces (10^{100} is not unheard of), so long as their search guidance mechanisms point them "the right direction".[3]

What does this mean for the comparison to the competing model-based approaches – SAT and CP – I mentioned above? *Should you personally consider to use planning to solve your problem? Or are you better off with something else?*

[3] In difference to FF, Graphplan gives an optimality guarantee. But it's a useless one (an algorithm artifact rather than a realistic optimization criterion). Anyhow, after having to wait for a couple of millennia, presumably the plan will be kinda useless.

I certainly wouldn't claim to have a definite answer, but let me try to give a guideline. As outlined, the effective planners mentioned above do not give any hard guarantees on solution quality. guarantee is essential to you, don't go there. Otherwise, if your problem is naturally formulated in terms of planning, then it is definitely worth a try. Which approach will be more effective computationally depends on "the kind of structure" your problem has. Constraint reasoning is probably better at playing through tightly interconnected options, like solving a puzzle, finding a clever schedule, or finding a complicated erroneous behavior in Model Checking. But such reasoning will be hopeless in a domain more like the one from Fig. 2, where guided greediness is key to finding large feasible solutions. As an intuitive if imprecise picture, a SAT solver may spend ages determining that n pigeons need at least n holes; heuristic planners will just go ahead and assign them.

Not wanting this paper to drag on forever, I'll only briefly touch upon two additional recent and important developments. First, optimal planning (with additive action costs functions) has undergone a major boost in the last years, mainly thanks to a new admissible heuristic called LM-cut [14] that under-estimates h^+. Combining admissible with non-admissible estimators yields powerful bounded-suboptimal search algorithms (see e.g. [38]) so, contradicting what I said above, this may be your way to go if quality guarantees are of essence.

A very recent development is that SAT-based planning – which has been more in the Graphplan ballpark up to now – seems to be about to catch up to heuristic search. Dramatic speed-ups are obtained through a combination of clever encodings, SAT call scheduling, and planning-specific branching heuristics [35,34]. In Jussi Rintanen's experiments, these planners manage to solve as many, and more, benchmark instances as the best heuristic search planners. They weren't as competitive in the 2011 planning competition, though. But certainly there is the potential to form another powerful tool for applying planning.[4]

Summing up, planning has made dramatic scalability progress, and might well be useful for solving some of your problems (whatever they are). So perhaps you'll reconsider the impression you got as a student when looking at (your equivalent of) Fig. 1, and you'll ask yourself:

4 Is it interesting to do research in?

Why, of course it is!

The malicious reader might be tempted at this point to put forward arguments of a psychological nature, connecting this statement and the fact that I've been working in planning for about 15 years now. I'd be happy to meet in a bar sometime to discuss this, but perhaps I can convince you here already.

[4] If you're curious about the self-contradiction with respect to the discussion of SAT/CP above: (a) the world is too complicated for a 12-page LNCS-style paper; (b) planning-specific greediness in the branching heuristic is a key element of these new SAT planners. Thus perhaps, to some extent, they are able to combine the virtues of heuristic search planning with those of constraint reasoning.

One thing that makes planning interesting is the wide open space of promising algorithmic possibilities. Whereas in SAT, the DPLL algorithm family (in particular CDCL: Conflict Driven Clause Learning) has been ruling the house since ages and people are busy finding new ways to push around the bits in unit propagation,[5] planning has always been characterized by a profusion of very different algorithms. One may argue that (a) heuristic search has been ruling the house in the planning competition since 10 years, which (b) is just as boring. I concede (a), except for recalling the aforementioned modern SAT-based planners [35,34]. But I beg to differ on (b) – it's not as boring as DPLL.

There is a wide open algorithmic space as to *how to automatically generate heuristic functions*. Arguably, this question boils down to "how to abstract the problem" since heuristic estimates are universally generated by considering some simplification ("abstraction"/"relaxation") of the planning task at hand. But the land of abstractions is indeed one of unlimited possibilities. That is particularly true of planning in difference to, e.g., Verification where abstractions also are paramount. In Verification, an abstraction must, as far as the to-be-verified property is concerned, be identical to the original system. In heuristic search planning, we can abstract in whichever way we want, and as much as we want, and we will still obtain potentially useful estimates.

Is, then, the life of a researcher in heuristic search planning characterized by the following pseudo-code?

while (not retired) **do**
  think up some new heuristic $h^{foo\text{-}bar}$
  run it on the benchmarks
endwhile

Fig. 3: The life of a planning researcher?

The answer to that one is "NO!". Far beyond just improving performance on benchmarks, the *understanding* of heuristics is where heuristic search planning really turns into a natural science. Dramatic progress has been made, in that science, during the last years. For example, Bonet and Geffner [3] proved (and exploited) connections between h^+, logic programming, and knowledge compilation; Katz and Domshlak [26] proved that optimal cost-partitionings[6] can be computed in polynomial time; Bonet and Helmert [4] proved that h^+ is equivalent to maximal hitting sets over action landmarks (sets of actions at least one of which takes part in every plan). Let me give just a little bit more detail on two other recent results, addressing what are probably the two most fundamental questions when trying to understand heuristics.

[5] My sincere apologies to colleagues from the SAT area for this polemical formulation. The comparison to SAT in the present context is owed to the fact that everybody (including myself) thinks that SAT is really cool.

[6] Given an ensemble of abstractions, a cost partitioning is a function that distributes the cost of each action across these abstractions, so that the sum of all abstraction heuristics is still a lower bound. The cost partitioning is optimal in a given state s if that sum, in s, is maximal.

4.1 How do heuristics relate to each other?

An arsenal of heuristic functions is, naturally, not merely a list $h_1, \ldots, h_N$ of names and associated methods. There are algorithmic relationships. In planning, specifically, the heuristics proposed so far can be classified as: *critical-paths heuristics*, which relate to Graphplan [13]; *relaxation heuristics* relating to h^+ [2,23]; *abstraction heuristics*, like pattern databases, homomorphically mapping the state space to a smaller abstract state space [9,15]; and *landmark heuristics*, based on counting yet un-achieved landmarks [33,25]. The relationships implied by this classification are obvious, and most of them are simply the historical effect of people building on each other's work. However, Helmert and Domshlak [14] recently showed how we can make connections *across* these classes, uncovering for example that certain apparently unrelated heuristics are, in fact, equivalent.

Helmert and Domshlak [14] introduce a *compilation framework for classes of heuristics*. Given two classes A and B of admissible heuristics (e.g., A=critical-paths heuristics vs. B=landmark heuristics), A can be compiled into B if, for any given state s and heuristic $h^A \in A$, one can in time polynomial in the size of the planning task construct a heuristic $h^B \in B$ so that $h^A(s) \leq h^B(s)$. That is, we can always at least simulate A through B, with only polynomial overhead. Fig. 4 provides their results overview for illustration.

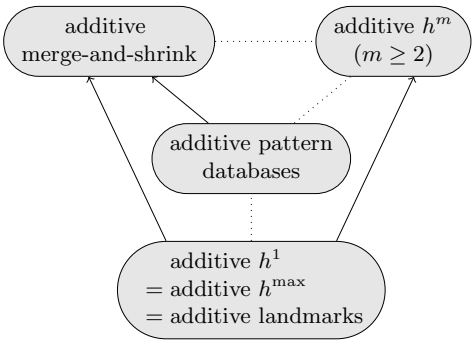

Fig. 4: Helmert and Domshlak's [14] compilation results. Arrows indicate compilability, dotted lines indicate that compilation is not possible (non-compilability of h^m into merge-and-shrink actually is a recent and yet unpublished result).

Additivity in Fig. 4 refers to the use of heuristic ensembles and cost partitionings. The h^+ heuristic is not included because it cannot be compilable into any heuristic (else we could compute h^+ in polynomial time). Major discoveries here are that landmarks are incomparable with pattern databases but can be compiled into merge-and-shrink (a generalization of pattern databases), and that landmarks are equivalent to h^{max} which is a relaxation heuristic. The latter result is easy for compilation into h^{max}, but is quite non-trivial for the other direction. The proof construction – constructing landmarks in a way so that the eventual heuristic value will at least dominate h^{max} – has been implemented. In fact, it's the aforementioned LM-cut heuristic that has revolutionized the scalability of optimal planning!

4.2 Under what circumstances does a given heuristic work well?

Any heuristic gives good search guidance only on some sub-class of planning tasks. What we would like to understand is what that class is. Ideally, we would like a machine to implement this "understanding", so that the planning technology could test automatically whether or not any one heuristic is likely to work well. In other words, we want a fortune-teller for search performance. Anybody experienced in search knows that this is so difficult one can just as well look into a crystal ball. Surprisingly, it turns out that for the h^+ heuristic – the most influential heuristic by far in planning – a suitable crystal ball actually exists.

The "crystal ball" is called TorchLight [16,22]. Its history started in the year 2000 when I tried to understand, manually and on a per-benchmark-domain basis, where and in what ways h^+ is informative. The final outcome of that was a "planning benchmark domain taxonomy", dividing them into classes differing with respect to the topology of the search space surface under h^+ [17,18,21]. Most strikingly, in many of the domains, *no local minima exist at all*.

Thus a very basic crystal ball should be able to divine whether or not there are local minima under h^+. In 2001, having attempted this in vain for several months, I gave up. In 2009 – while explaining to someone why this is never gonna work – I finally realized that *causal graphs* can do the trick. The vertices in this graph are the state variables, and there is an arc (x, y) iff moving y sometimes involves a condition on x. As it turns out, if the causal graph is acyclic and all variable transitions are invertible, then there are no local minima under h^+. This sufficient condition is easily testable, and can be significantly generalized. Voilà our crystal ball! Fig. 5 overviews its performance.

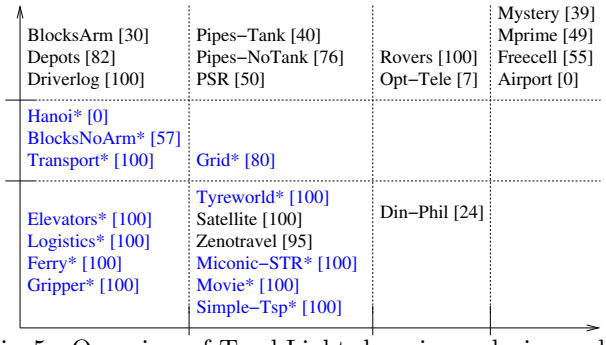

Fig. 5: Overview of TorchLight domain analysis results.

The table structure of Fig. 5 corresponds to my planning domain taxonomy. Leaving out the details, a domain's topology is the "easier" the nearer it is to the bottom left; domains without local minima are (highlighted in blue and) marked with a "*". The numbers in brackets give TorchLight's estimation of the domain's difficulty, namely the fraction of sampled states proved to not be on a local minimum. Thus 0="very hard", 100="very easy". This estimate is, of course, not perfect. But it correlates well with planner performance [22], and is indeed more refined than my own hand-made analysis. All *I* could give you is a "local minima exist? yes/no". So, don't ask me, ask TorchLight!

5 And now, what?

I was kinda hoping that at this point you might feel tempted to play around a little bit with an actual planner. For your convenience, I put a little starter package at `http://www.loria.fr/~hoffmanj/PlanningForDummies.zip`. It contains the FF source code as well as 3 simple benchmark domains with instance generators. Linux executables and some example planning tasks are included. I was so nice to also add a README file. If you get seriously interested, you should have a look through the planning competition web pages `http://ipc.icaps-conference.org/`, and through the page of Fast Downward `http://www.fast-downward.org/` which is quickly becoming the main implementation basis for new planning techniques. Do send me email if you have questions.

Scientifically, I'd like to close this paper by striking a blow for research on supporting *modeling* for planning. This is an active research area. The planning competition has a separate track of events for knowledge engineering (latest edition: `http://kti.mff.cuni.cz/~bartak/ICKEPS2009/`). Several research groups are developing modeling environments, e.g. itSIMPLE [39] `http://code.google.com/p/itsimple/` and GIPO [37] `http://scom.hud.ac.uk/planform/gipo/`. Learning techniques support the automatic extraction of domain models from various kinds of data [8,7]. Still, in my humble opinion, this issue is not nearly given sufficient attention. The vast majority of planning researchers are concerned with what's going on inside their planners, and worry little (at all?) about where these planners will actually get their PDDL input from – in practice, not in the planning competition!

I was no different until quite recently. Then I worked on 3 different applications, with Alexander Koller [27], at SAP [24], with Core Security Technologies [29]. In each and every one of these, it was easy to obtain good planning performance by simple modifications of FF. The real issue was designing the PDDL. Contrary to common lore, planning is not "automatic". Yes, it suffices to describe the domain. But that is not a push-button operation.

Remember what I pointed out previously: *planning is a quick hack to get things up and running.* I believe that our technology has great potential as a method for rapid prototyping. We are very far indeed from fully exploiting that potential. Search algorithms like branch-and-bound are widely known, and any practitioner (with a CS background) having a search problem to solve is likely to go ahead and build on those. People simply don't know that, perhaps, they could just describe their problem to a planner and be done. We need to get this knowledge out there, and we need to provide users with the tools needed to conveniently access our technology. Advancing the outreach of planning in this way is our major challenge for the coming decade – apart, of course, from keeping ourselves happy by proving interesting theorems about heuristic functions ☺

Acknowledgments. I would like to thank Stefan Edelkamp for inviting me to write this paper. I am grateful to all my colleagues from the planning community, and I thank them specifically for their work as described and cited herein. I hope we'll still be friends after you read this.

References

1. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence 90(1-2), 279–298 (1997)
2. Bonet, B., Geffner, H.: Planning as heuristic search. Artificial Intelligence 129(1–2), 5–33 (2001)
3. Bonet, B., Geffner, H.: Heuristics for planning with penalties and rewards formulated in logic and computed through circuits. Artificial Intelligence 172(12-13), 1579–1604 (2008)
4. Bonet, B., Helmert, M.: Strengthening landmark heuristics via hitting sets. In: Proceedings of the 19th European Conference on Artificial Intelligence (2010)
5. Bonet, B., Loerincs, G., Geffner, H.: A robust and fast action selection mechanism for planning. In: Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (1997)
6. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence 69(1–2), 165–204 (1994)
7. Castillo, L.A., Morales, L., González-Ferrer, A., Fernández-Olivares, J., Borrajo, D., Onaindia, E.: Automatic generation of temporal planning domains for e-learning problems. Journal of Scheduling 13(4), 347–362 (2010)
8. Cresswell, S., McCluskey, T.L., West, M.M.: Acquisition of object-centred domain models from planning examples. In: Proceedings of the 19th International Conference on Automated Planning and Scheduling (2009)
9. Edelkamp, S.: Planning with pattern databases. In: Recent Advances in AI Planning. 6th European Conference on Planning (2001)
10. Fox, M., Long, D.: PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research 20, 61–124 (2003)
11. Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs. Journal of Artificial Intelligence Research 20, 239–290 (2003)
12. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (2004)
13. Haslum, P., Geffner, H.: Admissible heuristics for optimal planning. In: Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (2000)
14. Helmert, M., Domshlak, C.: Landmarks, critical paths and abstractions: What's the difference anyway? In: Proceedings of the 19th International Conference on Automated Planning and Scheduling (2009)
15. Helmert, M., Haslum, P., Hoffmann, J.: Flexible abstraction heuristics for optimal sequential planning. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling (2007)
16. Hoffmann, J.: Where ignoring delete lists works, part II: Causal graphs. In: Proceedings of the 21st International Conference on Automated Planning and Scheduling (2011)
17. Hoffmann, J.: Local search topology in planning benchmarks: An empirical analysis. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (2001)
18. Hoffmann, J.: Local search topology in planning benchmarks: A theoretical analysis. In: Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (2002)

19. Hoffmann, J.: The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. Journal of Artificial Intelligence Research 20, 291–341 (2003)
20. Hoffmann, J.: Utilizing Problem Structure in Planning: A Local Search Approach, Lecture Notes in Artificial Intelligence, vol. 2854. Springer-Verlag (2003)
21. Hoffmann, J.: Where 'ignoring delete lists' works: Local search topology in planning benchmarks. Journal of Artificial Intelligence Research 24, 685–758 (2005)
22. Hoffmann, J.: Analyzing search topology without running any search: On the connection between causal graphs and h^+. Journal of Artificial Intelligence Research 41, 155–229 (2011)
23. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302 (2001)
24. Hoffmann, J., Weber, I., Kraft, F.M.: SAP speaks PDDL. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (2010)
25. Karpas, E., Domshlak, C.: Cost-optimal planning with landmarks. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (2009)
26. Katz, M., Domshlak, C.: Optimal additive composition of abstraction-based admissible heuristics. In: Proceedings of the 18th International Conference on Automated Planning and Scheduling (2008)
27. Koller, A., Hoffmann, J.: Waking up a sleeping rabbit: On natural-language sentence generation with FF. In: Proceedings of the 20th International Conference on Automated Planning and Scheduling (2010)
28. Koller, A., Petrick, R.: Experiences with planning for natural language generation. Computational Intelligence 27(1), 23–40 (2011)
29. Lucangeli, J., Sarraute, C., Richarte, G.: Attack planning in the real world. In: Proceedings of the 2nd Workshop on Intelligent Security (2010)
30. McDermott, D.: A heuristic estimator for means-ends analysis in planning. In: Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (1996)
31. McDermott, D., et al.: The PDDL Planning Domain Definition Language. The AIPS-98 Planning Competition Committee (1998)
32. Richter, S., Helmert, M.: Preferred operators and deferred evaluation in satisficing planning. In: Proceedings of the 19th International Conference on Automated Planning and Scheduling (2009)
33. Richter, S., Westphal, M.: The LAMA planner: Guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research 39, 127–177 (2010)
34. Rintanen, J.: Heuristics for planning with SAT. In: Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming (2010)
35. Rintanen, J., Heljanko, K., Niemelä, I.: Planning as satisfiability: parallel plans and algorithms for plan search. Artificial Intelligence 170(12-13), 1031–1080 (2006)
36. Ruml, W., Do, M.B., Zhou, R., Fromherz, M.P.J.: On-line planning and scheduling: An application to controlling modular printers. Journal of Artificial Intelligence Research 40, 415–468 (2011)
37. Simpson, R.M., Kitchin, D.E., McCluskey, T.L.: Planning domain definition using GIPO. Knowledge Engineering Review 22(2), 117–134 (2007)
38. Thayer, J., Ruml, W.: Bounded suboptimal search: A direct approach using inadmissible estimates. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (2011)
39. Vaquero, T.S., Romero, V., Tonidandel, F., Silva, J.R.: itSIMPLE 2.0: an integrated tool for designing planning domains. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling (2007)