Optimal Planning for Delete-free Tasks with Incremental LM-cut

Florian Pommerening
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
pommeren@informatik.uni-freiburg.de

Malte Helmert
Universität Basel
Fachbereich Informatik
Bernoullistrasse 16
4056 Basel, Switzerland
malte.helmert@unibas.ch

Abstract

Optimal plans of delete-free planning tasks are interesting both in domains that have no delete effects and as the relaxation heuristic h^+ in general planning. Many heuristics for optimal and satisficing planning approximate the h^+ heuristic, which is well-informed and admissible but intractable to compute. In this work, branch-and-bound and IDA* search are used in a search space tailored to delete-free planning together with an incrementally computed version of the LM-cut heuristic. The resulting algorithm for optimal delete-free planning exceeds the performance of A* with the LM-cut heuristic in the state-of-the-art planner Fast Downward.

Introduction

The delete-relaxation heuristic h^+ is a well-informed heuristic for classical planning. It is defined as the optimal solution of a task where all negative effects are ignored. Since it is intractable, it is seldom used in practice and approximations are used instead. For example, the additive heuristic (Bonet and Geffner 2001) is an upper bound approximation for h^+. The FF heuristic (Hoffmann and Nebel 2001), additive FF heuristic (Keyder and Geffner 2008), set-additive heuristic (Keyder and Geffner 2008) and local Steiner tree heuristic (Keyder and Geffner 2009) are all defined as costs of (potentially suboptimal) solutions to a delete-relaxed task and are hence also upper bounds for h^+. Admissible heuristics based on delete relaxations provide lower bounds for h^+, including Bonet and Geffner's (2001) $h^{\max}$, Karpas and Domshlak's (2009) admissible landmark heuristics and the LM-cut heuristic $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009).

For most of these heuristics, it is unknown how closely they approximate h^+. An algorithm that determines optimal solutions for delete-free planning tasks could help answer this question. Moreover, it could be applied directly to domains that contain no delete effects, such as the minimal seed set problem (Gefen and Brafman 2011).

Recent work by Betz and Helmert (2009) demonstrated the informativeness of h^+ as a heuristic for general planning tasks. Bonet and Helmert (2010) relate h^+ computation to finding optimal hitting sets for sets of disjunctive action landmarks. Bonet and Castillo (2011) turn this idea into an algorithm that computes h^+ through an implicit hitting set problem formulation. Haslum, Slaney, and Thiébaux (2012) follow the same basic approach, but use a different algorithm for generating the disjunctive action landmarks to hit. Based on this algorithm for computing h^+, Haslum (2012) defines the heuristic h^{++} that uses an optimal relaxed plan as input and discovers plans with increasing costs that converge on the optimal solution of the original planning task.

One way to calculate h^+ is to solve the delete relaxation of a task with an optimal planner. For example, A* can be used with the LM-cut heuristic, which is based on disjunctive action landmarks. In regular A* search, these landmarks are computed from scratch for every search state.

We present a new approach for solving delete-free planning tasks using the LM-cut heuristic. The key ingredients are a search space exploiting the properties of delete-free tasks and the use of depth-first search algorithms that allow incremental computation of $h^{\text{LM-cut}}$.

Notation

The planning tasks considered here are delete-free propositional STRIPS tasks with action costs:

Definition 1 (Delete-free planning task)**.** A (delete-free) *planning task* is a tuple $\Pi = \langle V, I, G, O \rangle$ with a finite set of propositional *variables* V, an *initial state* $I \subseteq V$, a set of *goals* $G \subseteq V$ and a finite set of *operators* O, where each $o \in O$ consists of a set of *preconditions* $\text{pre}(o) \subseteq V$, a set of (add) *effects* $\text{add}(o) \subseteq V$, and a *cost* $\text{cost}(o) \in \mathbb{R}_0^+$.

An operator o is *applicable* in state $s \subseteq V$ if $\text{pre}(o) \subseteq s$. Applying o in s results in the state $s[o] = s \cup \text{add}(o)$. An operator sequence $\pi = o_1 \dots o_n$ is applicable in s if there are states $s_0, \dots, s_n$ such that $s_0 = s$ and for all $1 \leq i \leq n$, o_i is applicable in s_{i-1} and $s_{i-1}[o] = s_i$. Applying π in s results in $s[\pi] = s_n$. If $G \subseteq s[\pi]$, the sequence π is called a *plan* for s. The cost of an operator sequence is $\text{cost}(o_1 \dots o_n) = \sum_{i=1}^{n} \text{cost}(o_i)$.

Heuristic functions h map states s to *heuristic values* $h(s) \in \mathbb{R}_0^+ \cup \{\infty\}$. The *perfect (delete-relaxation) heuristic* h^+ maps each state s to the cost of a cheapest plan for s. Heuristics h that never overestimate this cost, i.e. where $h(s) \leq h^+(s)$ for all s, are called *admissible*.

Optimal planning is the problem of finding minimal-cost plans for I. In the delete-free setting, this problem is NP-

equivalent (Bylander 1994) and not approximable within a constant factor unless P = NP (Betz and Helmert 2009).

Theory

Branch-and-Bound. *Depth-first branch-and-bound search* (Lawler and Wood 1966) is an informed search procedure which uses a heuristic to prune parts of its search tree. It finds the cheapest solution in a given interval or proves that there is no such solution. If the full interval $[0, \infty)$ is used, a globally cheapest solution is discovered. Whenever a plan π is discovered, the interval is intersected with $[0, \text{cost}(\pi))$ and the search for a cheaper solution continues. For each search node σ, the sum $f(\sigma)$ of a heuristic value $h(\sigma)$ and the cost needed to reach this node $g(\sigma)$ is an estimate for all solutions in the subtree rooted in σ. A node is pruned if the search interval does not intersect the interval $[f(\sigma), \infty)$, as this means that no solution represented in this subtree has a cost that is contained in the search interval. Branch-and-bound search is complete and optimal if the search space is finite and the heuristic is admissible.

Search Space. The search space used here differs from the one used by general planning algorithms. Instead of introducing one successor for each applicable operator in the search node's state, we pick *one* applicable operator and branch over the decision whether this operator is applied now or never. This is possible for delete-free STRIPS tasks, where no operator needs to be applied more than once and it is never beneficial to defer application of an operator to a later stage in the plan. If s is the state associated with the current search node and o is the chosen operator, then one successor node has the associated state $s[o]$ and the other has associated state s, and in both successor nodes o is removed from the set of operators.[1] As an additional optimization, if any operator with cost 0 is applicable in s, it is applied and subsequently removed from the task immediately without branching, as applying it can never be harmful. This optimization is also mentioned by Gefen and Brafman (2011).

A major advantage of this search space over a traditional one is that due to the removal of operators, heuristic values can increase quickly or even become infinite if a critical operator is removed, leading to more pruning.

LM-cut Heuristic. We use the LM-cut heuristic within branch-and-bound search, which experiments have shown to provide very accurate lower bounds to h^+ for many planning tasks (Helmert and Domshlak 2009). LM-cut is based on *disjunctive action landmarks*, i.e., operator sets L such that at least one operator in L must be contained in every solution. The cost of a landmark is the minimum over the costs of all operators contained in it. We refer to the literature for a complete description of $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009; Bonet and Helmert 2010). For our purposes, we only need to know that the LM-cut computation is performed in *rounds* where each round proceeds as follows:

1. Compute the $h^{\max}$ values of all variables. If $h^{\max}(g) = 0$ for all goal variables g, stop and return the computed heuristic value. If $h^{\max}(g) = \infty$ for some goal variable g, stop and return ∞.
2. Use the $h^{\max}$ values to compute a landmark L and add the cost c of L to the heuristic value (which starts as 0).
3. Reduce the operator costs of all operators in L by c.

In the usual (nonincremental) LM-cut heuristic, each computed landmark is discarded after step 3. and all operator costs are restored to their original values before returning.

Incremental LM-cut. The search space described above allows incremental computation of $h^{\text{LM-cut}}(s)$. To do so, we store the landmark set $\mathcal{L}$ computed by the LM-cut procedure as well as the remaining operator costs at the time the procedure ends. Assume we have just computed the heuristic for search node σ and are now considering a successor of σ.

For the successor σ_o where operator o is applied, we discharge all landmarks in $\mathcal{L}$ containing o by removing them from $\mathcal{L}$ and adding the costs of these landmarks back to their operators to undo step 3. of the LM-cut computation. All other landmarks in $\mathcal{L}$ must also be landmarks of σ_o, so we can start the LM-cut computation with these landmarks already computed. For the successor $\sigma_{\neg o}$ where operator o is not applied but forbidden, all landmarks in $\mathcal{L}$ remain valid and o can be removed from all landmarks containing it. If this makes some landmark empty, the node is a dead end. Again, we can start the LM-cut computation with many landmarks already computed.

In the common case of *binary-cost* planning tasks, where all action costs are equal to 0 or 1, the landmarks produced by LM-cut are always disjoint, so each operator o can be part of at most one landmark, which we call its *containing landmark* $\mathcal{L}(o)$ (undefined if o is not contained in any landmark). In this case, the incremental LM-cut computation can be made particularly efficient by applying the following rules:

- o was just applied and $\mathcal{L}(o)$ is undefined: no recomputation at all is necessary
- o was just applied and $|\mathcal{L}(o)| = 1$ (i.e., o is the only operator in its landmark): discharge $\mathcal{L}(o)$; no further recomputation is necessary
- o was just applied and $|\mathcal{L}(o)| > 1$: discharge $\mathcal{L}(o)$ and run the LM-cut procedure to check for new landmarks
- o was just forbidden and $\mathcal{L}(o)$ is undefined: no recomputation at all is necessary[2]
- o was just forbidden and $|\mathcal{L}(o)| = 1$: the new node is a dead end; return a heuristic value of ∞ immediately
- o was just forbidden and $|\mathcal{L}(o)| > 1$: remove o from $|\mathcal{L}(o)|$ and run the LM-cut procedure to check for new landmarks

Formal correctness proofs for these rules are provided by Pommerening (2011), along with examples showing that in the two cases where LM-cut computations are performed, they are indeed necessary to avoid missing landmarks.

[1] For a formal proof that the search space is correct in the sense that it always contains some optimal plan if the task is solvable, we refer to the first author's Master's thesis (Pommerening 2011).

[2] This is only true if we apply the previously mentioned optimization of automatically applying zero-cost operators.

Improvements

Variable Ordering. The search space of our branch-and-bound search is similar to the one used in backtracking algorithms for constraint satisfaction problems (CSPs). One important consideration for CSP search is which variable to branch over next (in our case: which applicable operator to consider). A frequently used heuristic for choosing the next variable to assign in CSPs is the *minimum remaining values* heuristic (Bacchus and van Run 1995) that chooses a variable with a minimal number of possible values, i.e. one that is maximally constrained. In our setting, this corresponds to branching over an applicable operator in a smallest landmark. We follow this strategy by determining the minimum size of a landmark that contains an applicable operator, collecting all applicable operators from such landmarks, and then selecting one such operator uniformly at random. (We do not consider operator cost in our branching decision because our experiments are limited to unit-cost task.)

Unit Propagation. The search space can be seen as defining an implicit model finding (SAT) task. For each operator, the algorithm has to decide whether it is part of the desired solution or not, which can be seen as a propositional variable. Dependencies between operators implicitly define constraints between such variables; in particular, landmarks correspond to propositional clauses. This view allows us to adopt the idea of *unit propagation* (Davis, Logemann, and Loveland 1962): if there is a single-operator landmark $L = \{o\}$ in σ and o is applicable, we can apply o without branching, i.e., only generate successor σ_o but not $\sigma_{\neg o}$. However, unit propagation can only lead to trivial run time improvements in the presence of the above-mentioned variable selection strategy (which prefers branching over short landmarks) and incremental LM-cut heuristic (which immediately identifies $\sigma_{\neg o}$ as a dead end).

Pure Symbol Heuristic. We already mentioned the automatic application of zero-cost operators. This can be compared to the *pure symbol heuristic* (Russell and Norvig 2010, Ch. 7) for satisfiability, which automatically assigns either a false or true value to a propositional variable if this choice clearly dominates assigning the complementary truth value.

Preprocessing

To reduce the search space further, we apply two preprocessing techniques that preserve optimal solution costs. First, we remove variables and operators that are irrelevant to the goal according to a standard back-chaining relevance analysis (e.g., Nebel, Dimopoulos, and Koehler 1997).

Second, following an observation by Haslum et al. (2012), we remove all operators that are not *first achievers* of any variable (cf. Richter, Helmert, and Westphal 2008).[3] We also remove variables which have no achievers.

Both filters are executed at least once and then repeated alternately until one of them reaches a fixpoint. Either filter can trigger additional pruning in the other one arbitrarily often. To see this, consider the task family where the nth task has variables $\{i, g, a, b_1, \ldots, b_n\}$, initial state $\{i\}$, goals $\{g\}$ and operators $\langle i \to a\rangle, \langle a \to g\rangle, \langle a \to b_1\rangle, \langle b_1 \to a, b_2\rangle, \ldots, \langle b_{n-1} \to a, b_n\rangle, \langle b_n \to a\rangle$, where $\langle P \to E\rangle$ is an operator with preconditions P, effects E, and cost 1.

[3] This transformation can be performed in polynomial time and preserves optimality for delete-free tasks. Neither of these properties holds for general planning tasks.

Experiments

We evaluated the algorithm on delete relaxations of all IPC 1998–2006 domains for which the grounded task representation generated by Fast Downward (Helmert 2006) does not have conditional effects or axioms. All experiments were run on a single core of an AMD Opteron 2356 CPU with a 2 GB memory limit and a 5 minute time limit.

We report the probability (as a percentage) to solve a randomly selected task from a randomly selected domain (*coverage*) and expected values for run time and number of expansions. The latter are reported as logarithmically scaled scores between 0 ($\geq$300 s and $\geq$1 000 000 expansions resp.) and 100 ($\leq$1 s and $\leq$100 expansions resp.), following the scoring system suggested by Richter and Helmert (2009). Because of the randomized variable selection strategy of the branch-and-bound search, we repeated the experiments 5 times and report averages (μ) and standard deviations (σ).

Comparison to A*+ LM-cut. In an initial experiment, we compared our branch-and-bound algorithm to the state-of-the-art planner Fast Downward. Our planner uses the "translator" part of Fast Downward for grounding, and hence the two planners start from the same grounded task representation. Other than this, we share no code with Fast Downward.

Our algorithm improves over the coverage of Fast Downward by nearly ten percentage points, clearly demonstrating the utility of the dedicated search space and incremental heuristic computation (Table 1, rows 1 and 2).

Initial Upper Bound. To further improve our algorithm, we compute an initial upper bound before starting the search. Tighter bounds in branch-and-bound search can reduce the search effort due to additional pruning. Specifically, we use the local Steiner tree improvement heuristic h^{lst} by Keyder and Geffner (2009). This heuristic is a post-processing technique for the additive FF heuristic $h^{\mathrm{FF/add}}$, which in turn is based on the additive and FF heuristics. Given a plan generated by $h^{\mathrm{FF/add}}$, h^{lst} selects a state variable v and partitions the plan into three parts: one needed only to reach v, one that depends on v being true, and the rest of the plan. An alternative for the first part is then generated and the part is replaced if the alternative is cheaper. This process is repeated for every state variable v achieved by the plan. As the heuristic value of h^{lst} is the cost of a plan for the relaxed task, it is an upper bound for h^+. Using this upper bound led to a modest but statistically significant improvement in coverage and a larger improvement in the expansion score due to the additional pruning (Table 1, row 3).

Improving All Plans. Building on these results we tried finding tighter upper bounds not only at the start of the search but also during search. Whenever branch-and-bound search discovered a new cheapest solution, we use the h^{lst} idea to try to find an even cheaper solution in order to reduce the upper bound more quickly. For this purpose, we

Name	Time score μ	Time score σ	Expansion score μ	Expansion score σ	Coverage score μ	Coverage score σ
Fast Downward with A* and $h^{\text{LM-cut}}$	44.249	-	46.096	-	49.249	-
Branch-and-bound search	54.294	0.099	51.974	0.085	59.032	0.301
BnB + initial upper bound	55.326	0.119	54.033	0.094	59.981	0.267
BnB + initial upper bound + local Steiner tree improvement	55.618	0.116	54.502	0.144	60.519	0.466
IDA*	55.188	0.117	53.667	0.082	60.120	0.112

Table 1: Algorithm scores for a 300 s time limit.

generalized the local Steiner tree improvement method in such a way that it can be seeded with an arbitrary initial plan (the original method exploits the particular way in which $h^{\text{FF/add}}$ generates plans; see Pommerening, 2011, for details on our generalization). The additional pruning introduced by improved solutions increased the coverage and expansion scores by another 0.5 points, again a modest but statistically significant improvement (Table 1, row 4).

IDA*. We also experimented with a different approach to calculate h^+ with an incrementally calculated LM-cut heuristic. Our branch-and-bound algorithm can be used to perform an iterative deepening A*(IDA*) search (Korf 1985; Korf, Reid, and Edelkamp 2001) by performing a sequence of zero-window branch-and-bound searches (i.e., searches with equal lower and upper bound). The first solution discovered by IDA* is optimal by construction, so the previous solution improvement techniques cannot be used here. The results are comparable to those for branch-and-bound search with solution improvement (Table 1, row 5).

Details and Comparison to Hitting-Set-Based Methods. Table 2 shows the number of solved tasks in each domain for our two best performing search strategies, for Fast Downward using A*+ $h^{\text{LM-cut}}$ (FD), and for the hitting-set-based approaches by Bonet and Castillo (BC) and by Haslum et al. (HST). Our branch-and-bound algorithm almost dominates Fast Downward, solving more than 50 additional tasks and only offering worse coverage in two domains. Compared to the approach by Bonet and Castillo, we solve more than 100 additional tasks and achieve the same or better coverage in all domains. The approach by Haslum et al. solves around 30 more tasks than we do, but the results are clearly complementary, with our IDA* algorithm solving more tasks than Haslum et al. in nine domains and fewer in seven domains.

Comparing our two algorithms to each other, neither branch-and-bound nor IDA* dominates the other. Note that the domains where IDA* search performs better contain only few high-quality upper bounds. As discussed, a tighter upper bound increases pruning and therefore decreases run time in branch-and-bound search. However, IDA* search does not use upper bounds and is hence independent of their quality.

In 370 cases, both bounds are perfect and the search for h^+ trivializes to calculating the two bounds. We remark that this is the first comparison of h^{lst} to h^+, and it shows that h^{lst} often provides excellent approximations to the optimal relaxed plan. For the domains blocks, gripper and miconic-strips, polynomial algorithms computing h^+ exist (Betz and Helmert 2009). In these domains, h^{lst} was always perfect and $h^{\text{LM-cut}}$ was only off by 1 in one blocks task.

Domain (#tasks)	FD	BC	HST	BnB	IDA*	Bounds perfect ($h^{\text{LM-cut}}$/h^{lst}/both)
airport (50)	34	**50**	49	**50**	**50**	(50/41/41)
blocks (35)	**35**	**35**	**35**	**35**	**35**	(34/35/34)
depot (22)	7	5	**18**	14	14	(2/6/2)
driverlog (20)	14	2	11	**15**	**15**	(3/6/2)
freecell (80)	6	1	**21**	2	3	(0/0/0)
grid (5)	1	1	**4**	2	2	(1/2/1)
gripper (20)	**20**	**20**	**20**	**20**	**20**	(20/20/20)
logistics00 (28)	23	26	**28**	**28**	**28**	(26/24/22)
logistics98 (35)	9	7	12	**16**	15	(13/12/9)
miconic-strips (150)	**150**	**150**	**150**	**150**	**150**	(150/150/150)
mprime (35)	**27**	14	25	**27**	26	(7/22/4)
mystery (30)	26	16	23	**28**	**28**	(9/21/5)
openstacks-strips (30)	5	0	**6**	5	4	(0/5/0)
pathways (30)	5	4	**27**	5	5	(5/11/5)
pipesworld-notankage (50)	17	3	15	18	**19**	(1/2/1)
pipesworld-tankage (50)	10	2	**13**	9	10	(0/3/0)
psr-small (50)	**50**	**50**	**50**	**50**	**50**	(50/50/50)
rovers (40)	13	12	17	**19**	**19**	(10/14/9)
satellite (36)	6	6	8	8	**9**	(6/1/1)
tpp (30)	13	12	13	23	**24**	(20/6/6)
trucks-strips (30)	7	3	**19**	9	9	(0/30/0)
zenotravel (20)	**13**	8	**13**	**13**	**13**	(9/11/8)
Total (876)	491	427	**577**	546	548	(416/472/370)

Table 2: Number of solved tasks for Fast Downward (FD), Bonet & Castillo (BC), Haslum et al. (HST), branch-and-bound with all improvements (BnB) and IDA*. Best results in bold. The last column shows how often the initial lower bound $h^{\text{LM-cut}}$/initial upper bound h^{lst}/both are perfect.

Conclusion

Using an incremental computation of LM-cut in a branch-and-bound search, we were able to calculate optimal plans for delete-free planning tasks. We used a search space where an operator is never branched over more than once. Two modifications that use a local Steiner tree improvement procedure to tighten the upper bound were discussed and shown to be useful. An IDA* search showed similar results with advantages in different domains. One aspect we did not discuss is that branch-and-bound is also very useful when optimality is not mandatory, as it is an anytime algorithm.

Extending the incremental LM-cut computation to general planning would also be possible, but in this case the search space would have to be changed as well. This is one direction we are planning to explore in future work.

Acknowledgments

This work was supported by DFG grant HE 5919/2-1.

References

Bacchus, F., and van Run, P. 1995. Dynamic variable ordering in CSPs. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP 1995)*, volume 976 of *Lecture Notes in Computer Science*. 258–275.

Betz, C., and Helmert, M. 2009. Planning with h^+ in theory and practice. In *KI 2009: Advances in Artificial Intelligence*, volume 5803 of *Lecture Notes in Computer Science*. 9–16.

Bonet, B., and Castillo, J. 2011. A complete algorithm for generating landmarks. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 315–318.

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

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*. 329–334.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69:165–204.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem-proving. *Communications of the ACM* 5:394–397.

Gefen, A., and Brafman, R. I. 2011. The minimal seed set problem. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 319–322.

Haslum, P.; Slaney, J.; and Thiébaux, S. 2012. Minimal landmarks for optimal delete-free planning. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169.

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.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, 588–592.

Keyder, E., and Geffner, H. 2009. Trees of shortest paths vs. Steiner trees: Understanding and improving delete relaxation heuristics. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1734–1739.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-A*. *Artificial Intelligence* 129:199–218.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Lawler, E. L., and Wood, D. E. 1966. Branch-and-bound methods: A survey. *Operations Research* 14(4):699–719.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proceedings of the 4th European Conference on Planning (ECP 1997)*, 338–350.

Pommerening, F. 2011. Optimal planning for delete-free tasks with incremental LM-cut. Master's thesis, Albert-Ludwigs-Universität Freiburg.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 273–280.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, 975–982.

Russell, S. J., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Pearson Education, 3rd edition.