On Using Lazy Greedy Best-First Search with Subgoaling Relaxation in Numeric Planning Problems

Enrico Scala[1], **Luigi Bonassi**[2]
[1]University of Brescia
[2]Oxford Robotics Institute, University of Oxford
enrico.scala@unibs.it, luigibonassi@robots.ox.ac.uk

Abstract

This paper studies the use of lazy greedy best-first search for numeric planning problems in combination with relaxation-based heuristics, helpful actions, and up-to-jumping actions. In particular, the new search schema that we study, whilst postponing evaluation of the heuristic at expansion time, focuses the search over those states that are reached by helpful and up-to-jumping actions. In addition, we revisit linear abstractions by improving the balance between computation time and information, providing guidance in non-simple numeric planning problems, too. The new search schema compares favorably over the IPC-23 benchmarks with alternative complete heuristic search planners from the literature.

Introduction

Planning with numeric information is crucial in a variety of domains, from robotics (e.g, (Kiam et al. 2020)) to traffic control (e.g., (Vallati et al. 2016)) and personalised medication (Alon et al. 2024). Despite its general undecidability (Helmert 2002), numeric information allows for very compact encodings that let numeric planners tackle problems out of reach for classical planners.

In this paper, we study how to make forward heuristic search more effective in numeric planning. We do so through the lens of lazy greedy best-first search (Helmert 2006) extended to account more natively helfpul actions and up-to-jumping actions (Scala et al. 2020b). Our contribution is a new search schema, namely lazy greedy best-first search with focus and jumps, which addresses challenges along two different dimensions: mitigating the computational burden of heuristic computation through deferred evaluation; making the search more focused and aware of the numeric structure with a schema that puts priorities to helpful actions and jumps derived by the numeric subgoaling relaxation. Notably, differently from previous works that also use helpful actions (Scala et al. 2020b) by making the search incomplete, our search schema preserves completeness; that is, we are able to find a solution if one exists. Secondly, to provide guidance in non-simple numeric problems, we propose a new abstraction with a simpler yet more effective transformation (Li et al. 2018). Lastly, we perform an extensive experimental analysis over the numeric benchmarks of the 2023 International Planning Competition (IPC-23) (Taitler et al. 2024). Our experiments focus on single-queue and show that these new expedients do boost heuristic search. Moreover, when combined together they provide a planner obtaining higher coverage than alternative heuristic search approaches (Chen and Thiébaux 2024; Scala et al. 2020b).

Numeric Planning Problems

A numeric planning problem P is a tuple $\langle X, F, I, G, A, c\rangle$, where X and F are numeric and Boolean variables. The state space of P is the set of all valuations for X and F. Let $v \in X \cup F$, and s be a state, $s(v)$ indicates the value of v in s. I is the *initial* state. Let L be the language of propositional formulas having as terminals either numeric conditions, i.e., $\xi\{\geq, >, =\}0$ with ξ a numeric expression over X and $\mathbb{Q}$, or Boolean equality, i.e., $p = \{\top, \bot\}$ with $p \in F$. G, the goal, is an element of L. A is a set of actions such that each $a \in A$ is a pair $\langle \mathrm{pre}, \mathrm{eff}\rangle$ where $\mathrm{pre} \in L$ and eff is a set of numeric and Boolean assignments. Numeric assignments are of the form $x := \xi$ with $x \in X$ and ξ being a numeric expression; Boolean ones are of the form $p := \{\top, \bot\}$. c is a function assigning non-negative values to actions in A. Applying an action $a \in A$ in s generates a new state $s' = s[a]$, such that for all $v \in X \cup F$ we have $s'(v) = s(v)$ if v is not affected by a. Otherwise, if $v \in X$, then $s'(v) = \xi$ with $(v := \xi) \in \mathrm{eff}(a)$ else ($v \in F$) $s'(v) = \top$ if $(v := \top) \in \mathrm{eff}(a)$ or $s'(v) = \bot$ if $(v := \bot) \in \mathrm{eff}(a)$. We assume no conflicting effects.

A plan for $P = \langle X, F, I, G, A, c\rangle$ is an actions sequence $\pi = \langle a_0, \ldots, a_{n-1}\rangle$ from A. Let $\langle s_0, s_1 = a_0[s_0], \ldots, s_n = a_{n-1}[s_{n-1}]\rangle$ be the induced state trajectory from I, π solves P if $\forall i = 0, \ldots, n-1 \cdot s_i \models pre(a_i)$, $I = s_0$, $s_n \models G$. A plan π solving P is optimal if there is no other plan $\pi' \neq \pi$ such that π' solves P and $\sum_{a\in\pi} c(a) > \sum_{a\in\pi'} c(a)$.

Relaxations and Guidance Mechanisms

A powerful approach to solving numeric planning problems is forward state-space search guided by a heuristic function, usually devised from a problem relaxation. We first present the heuristic framework we use, and then later the search schema. We focus here on the numeric subgoaling relaxation (Scala et al. 2020a), as it has proved effective in many ways (e.g., heuristic estimates (Scala et al. 2020a; Kuroiwa

et al. 2022; Piacentini et al. 2018), landmarks (Kuroiwa et al. 2021; Scala et al. 2017), other search guidance mechanisms (Scala et al. 2020b), and connection with classical planning (Haslum and Geffner 2000; Haslum 2009; Helmert and Domshlak 2009)).

Subgoaling Heuristics and Search Guidance Mechanisms

As heuristics we consider the h^{hbd}_{add}, h_{add} for succinctness, and h_{mrp}. The idea behind these heuristics is decomposition: reaching a set of (sub)goals is approximated looking at the cost of reaching each (sub)goal in isolation, regressing over achieving actions, and recursively accumulating the cost of getting each action precondition. Both heuristics differentiate Boolean (BC) from numeric conditions, which are in turn differentiated into simple and hard conditions. A numeric condition is i) simple (SC) if it is only affected by actions with effects of the form $x = x + k$ with $k \in \mathbb{Q}$, ii) hard (HC) otherwise. Let g be a propositional formula. Then $h_{add}(s, g) \doteq 0$ if $s \models g$. Otherwise, $h_{add}(s, g)$ is defined as:

$$\begin{cases} \min\limits_{a \in ach(g)} c(a) + h_{add}(s, \mathrm{pre}(a)) & \text{if } g \text{ is BC} \\ \min\limits_{a \in ach(g)} m^a_g(s) \cdot c(a) + h_{add}(s, \mathrm{pre}(a)) & \text{if } g \text{ is SC} \\ \min\limits_{a \in rel(g)} c(a) + h_{add}(s, \mathrm{pre}(a)) & \text{if } g \text{ is HC} \\ \sum\limits_{g' \in g} h_{add}(s, g') & \text{if } g \text{ is } \bigwedge_i g_i \\ \min\limits_{g' \in g} h_{add}(s, g') & \text{if } g \text{ is } \bigvee_i g_i \end{cases}$$

Above, $ach(g)$ is the set of the actions that make g true when g is a Boolean terminal, while it is the set of actions that can make a SC g true with some number of repetitions of a's effects $m^a_g(s)$. $m^a_g(s)$ can be computed in closed form (Scala et al. 2020a; Kuroiwa et al. 2022). $rel(g)$ are all actions affecting g. The next section proposes how to revisit the work by Li et al. (2018) and numeric error estimates (Chen and Thiébaux 2024) to have an informed yet efficient prediction for the cost of HCs.

An alternative estimate can be obtained through the best achievers from h_{add} and accumulating all necessary pairs of action-repetition. This gives a multi-relaxed plan (MRP) of the form $\pi^+(s) = \{(a_0, m_0), \ldots, (a_k, m_k)\}$, which can be used to compute h_{mrp} (more details in Scala et al. (2020b)):

$$\mathrm{h}_{mrp}(s) \doteq \sum_{a \in \{a^\star \mid (a^\star, m^\star) \in \pi^+(s)\}} c(a) \cdot \max_{(a, m') \in \pi^+(s)} \{m'\}$$

From a MRP we can also have helpful actions and up-to-jumping actions (Scala et al. 2020b).

Helpful actions. Let s be a state, an action is helpful if applicable in s and if it is an achiever of a subgoal $g \not\models s$ within G or some precondition of actions in $\pi^+(s)$. We use $pref(s)$ to refer to the helpful actions in state s. Helpful actions are widely used and at the core of state-of-the-art planners such as LAMA (Taitler et al. 2024; Richter and Westphal 2010).

Up-to-jumping actions. Up-to-jumping actions are pairs (a, m) following a different semantics of actions. Let s be a state, applying (a, m) generates a state $s' = s[a]_1..[a]_{m'}$ where $m' \leq m$; as action a can falsify its own precondition, an up-to-jumping action can stop earlier than m, i.e., as soon as we get to a state s' s.t. $s' \not\models pre(a)$. Up-to-jumping actions provide shortcuts in the search, connecting states not directly connected, hopefully closer to a goal state. Up-to-jumping actions for a state s, $J(S)$, can be generated as follows (Scala et al. 2020b): $J(s) \doteq \{(a, m) | (a, m) \in \pi, m > 1, m = L(\pi^+, a)\}$ with $L(\pi^+, a) = \arg\min_m \{m | (a, m) \in \pi^+\}$. Note that we can use one heuristic as an estimate while drawing search guidance from another. We exploit this observation in our experiments and found that our best configuration uses h_{add} as an estimate, and gets helpful actions and up-to-jumping actions from a multi repetition plan.

Linear Abstraction

Existing heuristics based on subgoaling relaxation provide informed guidance but almost ignore everything which is not a SC or a Boolean condition. To overcome this limitation, Li et al. (2018) propose a transformation called *linear abstraction*. The idea is to transform a linear planning problem into a simple one over-approximating reachability. Linear planning problems are those that feature linear effects, whereas simple problems feature only simple effects. Without loss of generality, we assume all effects represented as $e : x := x + \xi$ (Scala et al. 2016). An effect e is simple if ξ is a constant, while e is linear if ξ is not simple but a linear expression. For an action a, we use $\mathrm{eff}^s(a)$ and $\mathrm{eff}^l(a)$ to denote the simple and linear effects of a, respectively. The abstraction compiles each linear effect e in $\mathrm{eff}^l(a)$ into multiple actions with a single constant effect. It does so by decomposing the values of ξ into a set of disjoint non-empty intervals $\{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$ representing values for ξ except 0. Then, for each of these intervals $\mathcal{I}$, the abstraction creates a new action $a_{\mathcal{I}}$, whose execution is possible only if the value of ξ in the current state belongs to $\mathcal{I}$, and that changes the variable by a constant value k that belongs to $\mathcal{I}$. This schema is applied for every linear effect, transforming a non-simple problem P into a simple problem P_{abs}. Following Li et al. (2018), we consider effect decomposition that leads to a *safe* abstraction. That is, P_{abs} over-approximates the solutions of the original problem P, and therefore P_{abs} is only used for heuristic computation. This ensures that the relaxation applied to P_{abs} is unsolvable only when P is unsolvable. This means that if for some subgoaling heuristic h and state s of P_{abs} we have $h(s) = \infty$, then we can safely prune s.

Revisiting Abstractions for Non-Simple Conditions

The linear abstraction requires a practical decomposition of the right-hand side of linear effects. The approach by Li et al. (2018) considers an effect decomposition extracted by running AIBR (Scala et al. 2016) on the original problem P. Practically, this approach decomposes linear expressions of effects into many intervals. This fine-grained decomposition introduces many actions in the abstracted problem, prioritizing heuristic precision over computation time. To overcome this issue, and with the idea of trading off precision and computation time, we made the abstraction parametric on the number of intervals to consider. For example, we can use the first two intervals (one positive and one negative) extracted from the AIBR analysis. Moreover, we consider a

	AIBR			Fixed
# intervals	All	4	2	2
	56	68	72	**89**

Table 1: Number of solved IPC-23 linear problems with different abstractions. Results obtained by running a Greedy Best-First Search with the h_{mrp} and 300 seconds timeout.

decomposition into fixed intervals for all linear expressions. Table 1 indicates that the best value for the linear numeric problems of the IPC benchmarks is obtained when the number of intervals is equal to two. Moreover, the best approach consists of fixing the decomposition. For the sake of simplicity, we then focus on explaining the linear abstraction that we actually use, i.e., that with two fixed intervals. For each linear effect $e : x := x + \xi$ of an action a, we replace this effect with two actions, a_e^+ and a_e^- such that a_e^+ can be executed when ξ is positive and increases x by 1; a_e^- can be executed when ξ is negative and decreases x by -1. Intuitively, this decomposition splits every expression ξ into two intervals: $(-\infty, 0)$ and $(0, +\infty)$.

Formally, the linear abstraction considered transforms a linear numeric problem $P = \langle X, F, I, G, A, c\rangle$ into a simple numeric problem P_{abs} by reformulating A into a new set of actions $A_{abs} = \bigcup_{a\in A} \langle \mathrm{pre}(a), \mathrm{eff}^s(a)\rangle \cup \bigcup_{(x:=x+\xi)\in \mathrm{eff}^l(a)} \{a_e^+, a_e^-\}$ such that $a_e^+ = \langle \mathrm{pre}(a) \wedge \xi > 0, \{x := x + 1\}\rangle$ and $a_e^- = \langle \mathrm{pre}(a) \wedge \xi < 0, \{x := x - 1\}\rangle$. Also, the newly introduced actions retain the cost of the original action. This modification allows to treat linear conditions as SCs, and therefore exploit potentially more informed estimates.

Together with the abstraction over linear numeric effect, we introduce a further abstraction for the remaining HC cases using numeric error (Chen and Thiébaux 2024; Fainekos and Pappas 2009). Informally, we can deal with the remaining HC cases by taking the cheapest and relaxed reachable action affecting them, multiplying this action with the numeric error (pretending the action contributes of one unit) and then summing to the result the heuristic over its precondition. Let $NE(s, g)$ be the numeric error, if g is HC $h_{add}(s, g) = \min_{a\in rel(g)} c(a) \cdot NE(s, g) + h_{add}(s, \mathrm{pre}(a))$.

Lazy GBFS with Focus and Jumps

Solving planning via forward-state space search (Bonet and Geffner 2001) using best-first search algorithms revolves around the idea of incrementally exploring a frontier of valid plan-prefixes starting from the initial state, using a function f to order the expansion. Let n be any node, setting $f(n) = g(n) + h(n)$ results in A* (Dechter and Pearl 1985); $f(n) = h(n)$ in greedy best-first search (GBFS).

Lazy Greedy Best-First Search is a variant of GBFS that defers the evaluation of the heuristic of a node to when it is popped out from the frontier. This technique has been proposed in the Fast-Downward Planning system (Helmert 2006) based on the observation that the computation of the heuristic can be a bottleneck, so evaluating only at expansion time provides a better trade-off. Here, we study a variant of this schema. We combine deferred evaluation with helpful actions and exploit the nature of the numeric planning problem through up-to-jumping actions. Every time we extract a node from the frontier, we evaluate the heuristic value of the associated state s, and then add its successors to the frontier with different priorities. If the successor s' is reached through a helpful action a, then $h(s')$ is the heuristic of s less the cost of a. This way, priority is given to those states reached by helpful actions, and therefore the idea to make the Best-First more *focused*. Then, we add in the frontier states reached by up-to-jumping actions. Each such state s' is assigned a heuristic discounting the cost of reaching s' with the actions in the jump. Note that, by definition of up-to-jumping actions, we need a simulation step, i.e., applying the action for m' times where $m' \leq m$ in that the action preconditions can be violated before the m-th execution.

We name the resulting algorithm Lazy Greedy Best-First Search with Focus and Jumps, in short LG-FJ. Algorithm 1 summarizes the pseudo-code, highlighting with the comments *Focus* and *Jumps* the actual modifications to the original schema. Differently from the use of helpful actions presented in Scala et al. (2020b), LG-FJ does not do any unsafe pruning, so it eventually finds a solution if one exists.

Algorithm 1: Lazy GBFS with Focus and Jumps

```
Input: Start state I, Goal Formula G, Heuristic Function h,
       Jumping Function J, Helpful Function Pref
Output: Solution from I to s_g ⊨ G (if any), otherwise Failure
F ← {(I, h(I))}                                  ▷ Initialize Frontier
C ← ∅                                            ▷ Initialize Closed
while F is not empty do
    s ← pop(F)                                   ▷ Pop state with lowest h-value
    if s ⊨ G then return path(I, s)              ▷ Path from I to s
    C = C ∪ {s}
    for a, s' ∈ succ(s) such that s' ∉ C do
        if a ∈ Pref(s) then
            push(F, (s', h(s) − c(a)))           ▷ Focus
        else
            push(F, (s', h(s)))
    for a, m ∈ J(s) do
        (s', m') = apply(s, a, m)                ▷ Jumps
        if s' ∉ C then push(F, (s', h(s) − m' × c(a)))
return Failure
```

Experimental Evaluation

We evaluate the impact of Lazy Greedy Best-First Search (LGBFS) with the enhancement presented in the previous section against other complete heuristic search schemata. We focused on problems from the IPC-23 (Taitler et al. 2024), which are divided in Simple and Linear Numeric Problems (SNP, LNP resp.). To understand the contribution of each enhancement we run several experiments and present the most representative steps below. We start with GBFS guided by the h_{mrp} heuristic. We call this system, $\mathrm{G}(h_{mrp})$ (Scala et al. 2020b). Then we extend $\mathrm{G}(h_{mrp})$ with the proposed linear abstraction to better face linear numeric planning problems, thus obtaining $\mathrm{G}(h^{abs}_{mrp})$. Next, we consider using LGBFS guided by the h_{add} heuristic

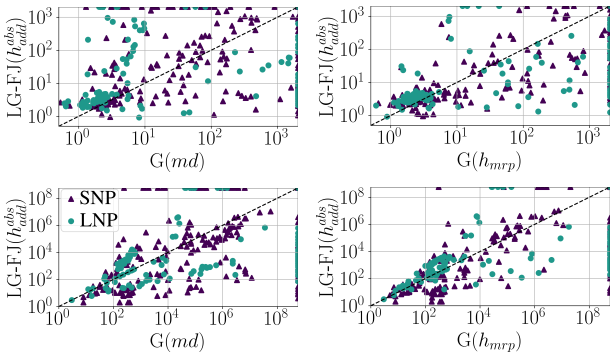

Figure 1: Pairwise comparison between LG-FJ(h^{abs}_{add}) and G(md) and between LG-FJ(h^{abs}_{add}) and G(h_{mrp}) in terms of runtime (above) and expanded nodes (below).

Domain	G(md)	G(h_{mrp})	G(h^{abs}_{mrp})	LG(h^{abs}_{add})	LG-FJ(h^{abs}_{add})
BGrouping	**20**	17	16	**20**	19
Counters	15	15	15	10	**18**
Delivery	16	11	11	17	**18**
Expedition	**7**	3	3	2	2
Farmland	**20**	**20**	**20**	**20**	**20**
HPower	4	**5**	4	**5**	4
MPrime	9	13	13	17	**19**
MTrader	**19**	2	2	6	14
Pathways	1	2	2	2	**3**
Rover	**9**	4	4	4	4
Sailing	5	**20**	**20**	16	**20**
Sugar	**20**	5	5	6	16
Watering	**20**	19	19	19	18
Partial SNP	165	136	134	144	**175**
Drone	**19**	16	16	11	11
FO-Counters	9	7	9	7	**19**
FO-Farmland	**20**	**20**	**20**	**20**	**20**
FO-Sailing	4	11	12	18	**20**
Settlers	0	**2**	**2**	1	**2**
TPP	**20**	5	12	**20**	**20**
Zeno	18	**20**	**20**	19	**20**
Partial LNP	90	81	91	96	**112**
Total	255	217	225	240	**287**

Table 2: Coverage of all systems across all domains.

(h_{add} worked better than h_{mrp} on LGBFS) in conjunction with the linear abstraction, namely LG(h^{abs}_{add}). Lastly, we extend LG(h^{abs}_{add}) by adopting the full-fledged LG-FJ, thus obtaining LG-FJ(h^{abs}_{add}). Together with this system, we also run GBFS guided by the manhattan distance heuristic (Chen and Thiébaux 2024), namely G(md). In all configurations, ties are broken to favor smaller g-values. We also observed G(md) outperforming both Patty (Cardellini, Giunchiglia, and Maratea 2024), and the winner of the IPC-23 competition, NLM-CutPlan (Kuroiwa, Shleyfman, and Beck 2022, 2023) run with the Sat configuration. We analyzed the number of solved instances (coverage), time, solution cost, and number of expanded nodes. All implementations are on ENHSP (https://github.com/hstairs/enhsp/releases/tag/enhsp20-0.21.0), and additional results at https://github.com/hstairs/icaps25-LG-FJ-SuppMaterial/. Experiments run on an Intel Xeon Gold 6140M 2.3 GHz, with 1800s timeout and 8GB of memory.

Table 2 shows the coverage achieved by all systems. Globally, each technique improves over the baseline. Specifically, compared to G(h_{mrp}), the linear abstraction improves coverage by 3.7%, while switching to LG(h^{abs}_{add}) increases coverage by 10.6%. Yet, the greatest gain in coverage is achieved when using the most sophisticated LG-FJ; Compared to G(h_{mrp}), LG-FJ(h^{abs}_{add}) increases coverage by 32.3%. Also, Table 2 shows that only LG-FJ(h^{abs}_{add}) surpasses the second-best performer G(md). Interestingly, the strength of G(md) is its simplicity: the employed heuristic trades off (some) informativeness for negligible computation time, which, compared to the simpler search approaches, pays off over the IPC-23 domains. Although there is no clear dominance between G(md) and LG-FJ(h^{abs}_{add}), we observe that overall LG-FJ(h^{abs}_{add}) performs better over both simple and linear planning problems.

To better understand the impact of LG-FJ we compare LG-FJ(h^{abs}_{add}) with the second-best performer G(md) and the baseline G(h_{mrp}). Figure 1 reports a pairwise comparison between LG-FJ(h^{abs}_{add}), G(md), and G(h_{mrp}) in terms of runtime (above) and expanded nodes (below). We observe that G(md) is slightly faster than LG-FJ(h^{abs}_{add}): G(md) solves 181 instances before LG-FJ(h^{abs}_{add}) does, while LG-FJ(h^{abs}_{add}) is faster in 140 problems. This is expected; the faster manhattan distance heuristic allows for lower running times. However, we also observe that LG-FJ(h^{abs}_{add}) solves 66 problems that G(md) does not solve, while G(md) only solves 34 instances that LG-FJ(h^{abs}_{add}) does not solve. Compared to G(h_{mrp}), LG-FJ(h^{abs}_{add}) is faster in 174 problems, while G(h_{mrp}) achieves lower runtimes in 120 instances. Regarding expanded nodes, LG-FJ(h^{abs}_{add}) is much better than G(md), which confirms that G(md) is less informed but expands nodes at a higher rate. Also, we observe that LG-FJ(h^{abs}_{add}) expands fewer nodes than G(h_{mrp}). Moreover, Figure 1 also reveals that LG-FJ(h^{abs}_{add}) is very complementary to G(md) in terms of run-time.

We also compared LG-FJ(h^{abs}_{add}) vs G(md) and G(h_{mrp}) in terms of solution cost. In this case, G(md) and G(h_{mrp}) outperform LG-FJ(h^{abs}_{add}). G(md) finds cheaper solutions in 135 instances, compared to only 35 for LG-FJ(h^{abs}_{add}). Similarly, G(h_{mrp}) finds cheaper solutions in 136 instances, while LG-FJ(h^{abs}_{add}) performs better in just 15. Indeed, LG-FJ(h^{abs}_{add}) prioritizes speed over cost minimization. In terms of solution cost, G(md) and G(h_{mrp}) perform similarly; G(md) finds cheaper solutions in a slightly higher number of problems (60 vs 55), but on average, G(h_{mrp}) does better in 10 domains, while G(md) outperforms it in 6.

Finally, we run G(h_{mrp}) plus helpful action pruning and up-to-jumping actions. This solved 245 instances; 268 with the revisited linear abstraction, proving that LG-FJ(h^{abs}_{add}) has more coverage to alternative incomplete methods, too.

Future Works

A key direction for our future work is to experiment using multi-queue heuristic search strategies and portfolio-based planner, together with novelty (Chen and Thiébaux 2024). We are eager to study the impact on coverage and quality of solutions. This study could reveal new synergies and enhance robustness and scalability of numeric planners.

Acknowledgments

Enrico Scala was supported by the project NEACD: Neurosymbolic Enhanced Active Cyber Defence (CUP J33C22002810001). Luigi Bonassi was supported by the joint UKRI and AISI-DSIT Systemic Safety Grant [grant number UKRI854].

References

Alon, L.; Weitman, H.; Shleyfman, A.; and Kaminka, G. A. 2024. Planning to be Healthy: Towards Personalized Medication Planning. In Endriss, U.; Melo, F. S.; Bach, K.; Diz, A. J. B.; Alonso-Moral, J. M.; Barro, S.; and Heintz, F., eds., *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, 4232–4239. IOS Press.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129: 5–33.

Cardellini, M.; Giunchiglia, E.; and Maratea, M. 2024. Symbolic Numeric Planning with Patterns. In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, 20070–20077. AAAI Press.

Chen, D. Z.; and Thiébaux, S. 2024. Novelty Heuristics, Multi-Queue Search, and Portfolios for Numeric Planning. In Felner, A.; and Li, J., eds., *Seventeenth International Symposium on Combinatorial Search, SOCS 2024, Kananaskis, Alberta, Canada, June 6-8, 2024*, 203–207. AAAI Press.

Dechter, R.; and Pearl, J. 1985. Generalized Best-First Search Strategies and the Optimality of A*. *J. ACM*, 32(3): 505–536.

Fainekos, G. E.; and Pappas, G. J. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42): 4262–4291.

Haslum, P. 2009. $h^m(P) = h^1(P^m)$: Alternative Characterisations of the Generalisation From $h^{\max}$ To h^m. In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI.

Haslum, P.; and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *AIPS*, 140–149. AAAI.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *AIPS*, 44–53. AAAI.

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

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI.

Kiam, J. J.; Scala, E.; Javega, M. R.; and Schulte, A. 2020. An AI-Based Planning Framework for HAPS in a Time-Varying Environment. In *ICAPS*, 412–420. AAAI Press.

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2022. LM-Cut Heuristics for Optimal Linear Numeric Planning. In *ICAPS*, 203–212. AAAI Press.

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2023. Extracting and Exploiting Bounds of Numeric Variables for Optimal Linear Numeric Planning. In *ECAI*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, 1332–1339. IOS Press.

Kuroiwa, R.; Shleyfman, A.; Piacentini, C.; Castro, M. P.; and Beck, J. C. 2021. LM-Cut and Operator Counting Heuristics for Optimal Numeric Planning with Simple Conditions. In *ICAPS*, volume 31, 210–218.

Kuroiwa, R.; Shleyfman, A.; Piacentini, C.; Castro, M. P.; and Beck, J. C. 2022. The LM-Cut Heuristic Family for Optimal Numeric Planning with Simple Conditions. *J. Artif. Intell. Res.*, 75: 1477–1548.

Li, D.; Scala, E.; Haslum, P.; and Bogomolov, S. 2018. Effect-Abstraction Based Relaxation for Linear Numeric Planning. In *IJCAI*, 4787–4793. ijcai.org.

Piacentini, C.; Castro, M. P.; Ciré, A. A.; and Beck, J. C. 2018. Linear and Integer Programming-Based Heuristics for Cost-Optimal Numeric Planning. In *AAAI*, 6254–6261. AAAI Press.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39: 127–177.

Scala, E.; Haslum, P.; Magazzeni, D.; and Thiébaux, S. 2017. Landmarks for Numeric Planning Problems. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 4384–4390. ijcai.org.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2016. Interval-Based Relaxation for General Numeric Planning. In *ECAI*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, 655–663. IOS Press.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2020a. Subgoaling Techniques for Satisficing and Optimal Numeric Planning. *J. Artif. Intell. Res.*, 68: 691–752.

Scala, E.; Saetti, A.; Serina, I.; and Gerevini, A. E. 2020b. Search-Guidance Mechanisms for Numeric Planning Through Subgoaling Relaxation. In *ICAPS*.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fiser, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Mag.*, 45(2): 280–296.

Vallati, M.; Magazzeni, D.; Schutter, B. D.; Chrpa, L.; and McCluskey, T. L. 2016. Efficient Macroscopic Urban Traffic Models for Reducing Congestion: A PDDL+ Planning Approach. In *Proceedings of AAAI*, 3188–3194.