OpCount4Sat: Operator Counting Heuristics for Satisficing Planning

Daniel Doebber[1], **André Grahl Pereira**[1], **Augusto B. Corrêa**[2]

[1] Federal University of Rio Grande do Sul, Brazil
[2] University of Basel, Switzerland
dmdoebber@gmail.com, agpereira@inf.ufrgs.br, augusto.blaascorrea@unibas.ch

Introduction

Operator counting heuristics (Pommerening et al. 2014) estimate the cost to the closest goal state by counting the minimum number of operators needed to achieve the goal condition. The idea behind operator counting is to use *constraints* to encode how often an operator needs to occur in a possible plan. These constraints express properties that every plan must satisfy. For example, an *operator counting constraint* could represent that every plan must contain operator o_1 or operator o_2 at least once. The set of constraints forms a *linear program* (LP), and planners use the optimal solution of this LP as the heuristic value for the state being evaluated. So far, these heuristics have been used in optimal planning (e.g., Pommerening, Röger, and Helmert 2013; Pommerening, Helmert, and Bonet 2017; Pommerening 2017).

In our work, we extend the application of operator counting heuristics to satisficing planning. In particular, we focus on the *post-hoc optimization constraints* (Pommerening, Röger, and Helmert 2013). The key idea of our method is to not use an LP-solver to compute a heuristic, but to solve the constraints greedily, one-by-one. In this way, we give up the requirement of an optimal solution to satisfy the constraints, but our heuristic computation becomes much faster, and we solve the integer version of problem.

We detail our algorithm next. We introduce the OPCOUNT4SAT planner that uses these ideas and participated on the International Planning Competition (IPC) 2023 on the satisficing and agile tracks. OPCOUNT4SAT is built directly on top of Fast Downward (Helmert 2006, 2009).

We assume that the reader is familiar with abstractions in classical planning and, in particular, with the concept of *pattern databases* (PDB) (e.g., Culberson and Schaeffer 1998; Edelkamp 2002). For more details on post-hoc optimization, we refer to the work by Pommerening, Röger, and Helmert (2013). For a thorough discussion on operator counting heuristics (and other LP-based techniques in planning), we recommend the work by Pommerening (2017).

Post-Hoc Optimization

Post-hoc optimization (Pommerening, Röger, and Helmert 2013) is a technique to make PDB heuristics more informed. It uses the notion of operator counting to get better estimates than simply getting the maximum or summing multiple heuristics values. Let $\mathsf{Count}_o \in \mathbb{N}_0$ be a variable estimate how many times operator o is needed to reach a goal. We know that $h^P(s)$ – i.e., the heuristic value from a pattern P in state s – *cannot be higher than the cost of all operators used in the abstract plan*. More formally, we define a post-hoc optimization constraint:

$$h^P(s) \leq \sum_{o \in \mathcal{O}_P} \mathit{cost}(o)\, \mathsf{Count}_o,$$

where $\mathcal{O}_P$ is the subset of the operators (in the original task) that *affect* the pattern P. We say that an operator o affects the pattern P if it (i) has a variable v from P in its effect, and (ii) whenever $v \in P$ occurs both in the effect and the precondition of o, then they have different values.

The main idea is that we have one post-hoc optimization constraint for each pattern $P_1, \ldots, P_k$, and satisfying all of k constraints simultaneously will give us a more informative heuristic than simply aggregating the $h^{P_1}(s), \ldots, h^{P_m}(s)$ values by their maximum value. Moreover, if we minimize the total cost of the operators "counted" to satisfy these constraints, we are still guaranteed to have an admissible heuristic value (Pommerening, Röger, and Helmert 2013).

Let $\mathcal{O}$ be the set of operators of the original task. We can frame this idea using the following integer program (IP):

$$\text{Minimize} \sum_{o \in \mathcal{O}} \mathit{cost}(o)\, \mathsf{Count}_o \tag{1}$$

$$h^P(s) \leq \sum_{o \in \mathcal{O}_P} \mathit{cost}(o)\, \mathsf{Count}_o \tag{2}$$

$$\mathsf{Count}_o \geq 0 \tag{3}$$

where (2) is quantified for all $P \in \{P_1, \ldots, P_k\}$, and (3) is quantified for all $o \in \mathcal{O}$.

The optimal value of this IP can be used as a heuristic for planning. Of course, computing one IP for every evaluated state is too expensive, so planners use the LP-relaxation of this program to come up with an admissible heuristic quickly.

Our Approach

We introduce a method to use post-hoc optimization as a heuristic function for satisficing planning, called OPCOUNT4SAT. It solves the IP problem of the post-hoc op-

timization heuristic aiming to minimize the objective function with a simple and fast greedy algorithm, but without optimality guarantees.

OPCOUNT4SAT iteratively satisfies each constraint in some arbitrary order. In our submission, we consider first constraints that are generated by smaller patterns. For each operator o, the algorithm maintains the current count of operators Count_o globally. All operators start with the count set to zero. Then, for each constraint, it iteratively increments the value of each Count_o, until the constraint is satisfied. Note that the count of operators in a constraint can be incremented many times in sequence before the constraint is satisfied. Once the constraint is satisfied, we move on to the next constraint. No operator is incremented if the current global count of operators Count_o already satisfies the constraint. The value of the heuristic function returned by the algorithm is the sum of the counts of the operators.

We use constraints from *Sys1*, *Sys2*, *Sys3*, and *Sys4* – the systematically generated patterns with 1, 2, 3, and 4 variables (Pommerening, Röger, and Helmert 2013). However, it might be impractical to comptue *Sys3* and *Sys4*. Therefore, we limit the number of patterns used from *Sys3* and *Sys4* to the sum of the number of patterns from *Sys1* and *Sys2*. The patterns used from *Sys3* and *Sys4* are randomly selected from the respective sets. Finally, we use a memory limit of 3.2 GiB for the PDB construction. If this limit is reached, we simply compute the heuristic with the PDBs already built.

Result Analysis

Our submission of OpCount4Sat used a greedy best-first search algorithm guided by the heuristic function described above. In the satisficing track, the planner solved 53 out of 140 tasks if we consider the domain version in which the planner solved the most tasks (normalized or not). Both memory and time were limiting factors. For example, the planner failed to solve 16 tasks of Labyrinth due to memory limit, and it failed to solve 11 tasks of Folding and 13 tasks of Slitherlink due to time limit. Also, OpCount4Sat supports a limited fragment of PDDL – it does not support conditional effects and derived variables, for example. Thus, it did not run for neither version (original and normalized) of the Rubik's Cube domain.

In the agile track, OpCount4Sat only solved 30 out of the 140 tasks. The issues mentioned for the satisficing track still apply here. The shorter time limit (5 minutes) impacted the planner's performance in some particular domains. For example, the coverage in the Ricochet Robots domain decreased from 14 to 0. This happened because the generation of the PDBs already takes more than 2 minutes even in the smallest instance, leaving only a few seconds left for the search.

References

Culberson, J. C.; and Schaeffer, J. 1998. Pattern Databases. *Computational Intelligence*, 14(3): 318–334.

Edelkamp, S. 2002. Symbolic Pattern Databases in Heuristic Search Planning. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 274–283. AAAI Press.

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

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, 173: 503–535.

Pommerening, F. 2017. *New Perspectives on Cost Partitioning for Optimal Classical Planning*. Ph.D. thesis, University of Basel.

Pommerening, F.; Helmert, M.; and Bonet, B. 2017. Abstraction Heuristics, Cost Partitioning and Network Flows. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 228–232. AAAI Press.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364. AAAI Press.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based Heuristics for Cost-optimal Planning. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 226–234. AAAI Press.