Sequencing Operator Counts

Toby Davies, Adrian R. Pearce, Peter Stuckey
National ICT Australia and
The University of Melbourne
Melbourne, Australia
firstname.lastname@nicta.com.au

Nir Lipovetzky
Computing & Information Systems
The University of Melbourne
Melbourne, Australia
nir.lipovetzky@unimelb.edu.au

Abstract

Operator-counting is a recently developed framework for analysing and integrating many state-of-the-art heuristics for planning using Linear Programming. In cost-optimal planning only the objective value of these heuristics is traditionally used to guide the search, however the primal solution, i.e. the operator counts, contain useful information. We exploit this information using a SAT-based approach which given an operator-count, either finds a valid plan; or generates a generalized landmark constraint violated by that count. We show that these generalized landmarks can be used to encode the perfect heuristic, h^*, as a Mixed Integer Program. Our most interesting experimental result is that finding or refuting a sequence for an operator-count is most often empirically efficient, enabling a novel and promising approach to planning based on Logic-Based Benders Decomposition (LBBD).

Introduction

We investigate the problem of sequencing operator counts obtained from an operator counting heuristic. The algorithm will find a feasible sequence, if it exists, or obtain an explanation why there is no plan that uses only the operators counted. We refer to these explanations as generalised disjunctive action landmarks.

Disjunctive action landmarks are a core feature of many admissible heuristics (Helmert and Domshlak 2009; Bonet and Helmert 2010; Haslum, Slaney, and Thiébaux 2012; Imai and Fukunaga 2014). Admissible heuristics based on these landmarks count the occurrence of any operator at most once. Most are dominated by the optimal delete relaxation h^+ (Helmert and Domshlak 2009).

We generalize this notion of disjunctive action landmarks to count operators multiple times, and show that admissible heuristics using generalized landmarks are capable of defining the perfect heuristic h^*. As disjunctive action landmarks are the only kind of landmark we consider in this paper, we will refer to them simply as "landmarks".

We present a complete, incremental algorithm for generating generalized landmarks, prove that generalized landmarks can encode h^*, and experimentally verify that this algorithm computes h^*. We show that even if we compute h^*,

the corresponding operator count does not necessarily represent a valid plan. Our approach can be used both as an incremental lower bound function and as an optimal planner, much like h^{++} (Haslum 2012), as our approach does not terminate until it finds a proof that it has computed h^*, i.e. finds a plan with optimal cost.

We explain this approach to planning in terms of Logic-Based Benders Decomposition (LBBD). LBBD partitions an optimization problem in terms of a Mixed Integer Programming master problem, and one or more combinatorial subproblems used to explain flaws in the master problem.

This approach to planning is particularly promising for two reasons. Firstly, it introduces a principled interaction between operator-counting heuristics, and SAT. This interaction can be applied to any explanation-based combinatorial search approach including SAT Modulo Theories (SMT) (Nieuwenhuis, Oliveras, and Tinelli 2006) and constraint programming using Lazy Clause Generation (LCG) (Ohrimenko, Stuckey, and Codish 2009). Constraints or theories capable of generating clausal explanations can be added to the SAT model we present, potentially allowing direct integration of cost-optimal planning with SMT and state-of-the-art scheduling approaches based on LCG. Planning Modulo Theories problems (Gregory et al. 2012) could therefore potentially be tackled using the extensive range of *existing* theories and constraints *already implemented* by the SMT and constraint programming communities.

Secondly, this approach decomposes the planning problem into problems for which there exist well-suited optimisation technologies: Mixed Integer Programming handling the linear counting constraints; and Conflict-Directed Clause Learning for the problem of operator sequencing given operator counts. This allows planning to take advantage of the ever improving performance of both of these widespread and industrially applied technologies.

Background

SAS$^+$ planning A SAS^+ planning task is a tuple $\langle V, O, s_0, s_*, c \rangle$ where V is a set of finite domain state variables, O is a set of operators, s_0 is a full assignment of each variable to one of its values representing the initial state, and s_* is a partial assignment of some subset of V representing the goal states. Finally c is a function $O \to \mathbb{N}$ that assigns a non-negative cost to each operator.

Each variable $X \in V$ has a domain $D(X)$, we sometimes abuse notation and write $X = x \in V$ which should be read $X \in V \wedge x \in D(X)$. Each operator o has a set of preconditions $pre(o)$ which is a partial assignment representing the preconditions of that operator, and a set of postconditions $post(o)$ which is a partial assignment representing the effects of the operator. Note that for simplicity, we do not distinguish between preconditions and prevail conditions in this paper.

A state s in the search space is a full assignment of every variable to a value, s is said to satisfy a partial assignment F if all assignments in F are also in s, i.e. $X = x \in F \Rightarrow X = x \in s$. A state is said to be a goal state if it satisfies the partial assignment s_*.

An operator $o \in O$ is applicable in s if s satisfies the partial assignment $pre(o)$. If o is applicable in state s, applying o yields a new state s' which is the same as s except that all assignments $X = x \in post(o)$ replace any assignment to X.

A plan π is a sequence of operators $o_1, \cdots o_n$ such that o_1 is applicable in s_0, each subsequent operator is applicable in the state resulting from applying the previous operators in sequence, and the final state satisfies s_*. An optimal plan has the minimum sum of operator costs of all plans, a SAS^+ planning task may have many optimal plans.

Mixed Integer Programming   A Mixed Integer Program (MIP) is a representation of a combinatorial optimisation problem in terms of linear constraints over some finite set of integer and continuous variables. Finding a solution to a MIP is an NP-complete problem, however its linear relaxation, (which replaces all integer variables with continuous ones) can be optimised in polynomial time.

In recent years many admissible planning heuristics have been proposed that use linear programs (van den Briel et al. 2007; Coles et al. 2008; Bonet 2013; Pommerening et al. 2014; Bonet and van den Briel 2014).

Of particular interest is the family of operator-counting heuristics. Operator-counting uses a linear programming framework with a common set of variables Y_o representing the count of occurrences of each operator o in some relaxed representation of a plan. One or more component heuristics can be encoded as linear constraints on these variables such that the combined operator-counting heuristic dominates each of the component heuristics, often strictly (Pommerening et al. 2014). Pommerening et al. describe how to encode a variety of heuristics within this framework, including optimal cost partitioning (Katz and Domshlak 2008) of LM-Cut landmarks (Helmert and Domshlak 2009).

As presented in the original paper, operator-counting heuristics may be linear programs, by requiring operator count variables to be integer, we obtain a MIP that is at least as tight as the LP. As we wish to obtain perfect operator counts, branching on operators will be essential in general, and in the remainder of this paper we will assume that any operator-counting heuristics are in fact MIPs, and we will be explicit when we solve only the linear relaxation.

Operator Counts   The solution to an operator-counting heuristic assigns a count to each operator o whenever the MIP is optimized. To distinguish the count assigned to each operator by a solution to an operator-counting heuristic from the variable Y_o, we refer to a solution to the heuristic as an operator-count $\mathcal{C}$.

We primarily treat an operator-count as a function from operators to their count assigned in the last solution of the MIP. The values of each Y_o will change throughout any solving process, conversely an operator count $\mathcal{C}$ should be considered an immutable copy of the assignments. We also refer to operator counts as multisets: a count $\mathcal{C}$ is said to be a superset of another count $\mathcal{C}'$ if $\forall o \in O : \mathcal{C}(o) \geq \mathcal{C}'(o)$.

An operator-count $\mathcal{C}$ is said to be a projection of a plan π if for each distinct operator o, there exist exactly $\mathcal{C}(o)$ copies of that operator in π. We say an operator count is *perfect* if it is the projection of an optimal plan.

Delete Relaxation   The delete relaxation changes the SAS^+ transition function such that applying an operator o does not replace previous assignments to variables, but accumulates them.

Imai and Fukunaga (2014) introduce a new MIP encoding of the optimal delete relaxation heuristic h^+. Their model uses 0-1 variables $\mathcal{U}(o)$ for each operator o to represent the fact that at least one o appears in the delete relaxed plan. They also present an extension to h^+ which they call "counting constraints" which are roughly equivalent to the lower-bound constraints in single-variable flow models (van den Briel et al. 2007). These constraints utilise additional integer variables $\mathcal{N}(o)$ which count occurrences of an operator o. For consistency with Pommerening et al. (2014), we will denote $\mathcal{N}(o)$ as Y_o.

Incremental lower bounding   Incremental lower bounding is a general technique for obtaining high-quality lower bounds, which can be useful in proving the quality of an existing plan. Incremental lower bounding was most prominently used in planning by Haslum (2012), however the technique is used throughout the various optimisation communities, referred to as "dual" techniques, reflecting the dual (lower) bound obtained from a linear program. Haslum (2012) describe a distinctive property of incremental lower bounding techniques as "informed by flaws in the current [optimum]".

Generalized Landmarks

The constraints in the h^+ model of Imai and Fukunaga (2014) rely on operator variables being binary. In general this does not hold with operator counting heuristics, in particular, flow-based heuristics (van den Briel et al. 2007; Bonet 2013; Bonet and van den Briel 2014) can count arbitrarily many executions of an operator. Imai and Fukunaga use $\mathcal{N}(o)$ variables to handle this in their "counting constraint" extension. Changing these $\mathcal{N}(o)$ variables to Y_o leads us to a simple but interesting alternative notation for
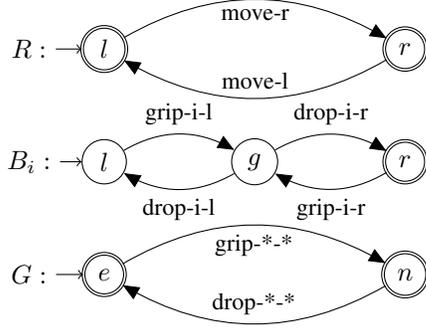
Figure 1: Domain Transition Graphs in gripper. The variable R represents the location of the robot (in the left or right room) ; B_i represents the location of ball i, (in the left or right room, or in the gripper); G represents the state of the gripper (empty or non-empty). For each automaton, the initial state is marked by an incoming arrow, and the states consistent with the goal s_* are double circled.

their core $\mathcal{U}(o)$ variables, which we denote $[Y_o \geq 1]$.[1] We generalize this notion by separating variables representing the number of times an operator o occurs, Y_o, from variables representing if an operator occurs at least k times, $[Y_o \geq k]$, which we refer to as bounds literals.

Bounds literals can be used to form linear constraints of the form $[Y_{o_1} \geq k_1] + \cdots + [Y_{o_n} \geq k_n] \geq 1$ which we call *generalized landmark constraints*. Note that only one bounds literal need occur per operator within the same landmark. If the same operator o had two literals $[Y_o \geq k_1]$ and $[Y_o \geq k_2]$ in the same landmark with $k_2 > k_1$, then $[Y_o \geq k_2]$ can be omitted without changing the solutions, as $[Y_o \geq k_1] \geq [Y_o \geq k_2]$.

Definition 1. *A generalized landmark constraint is a linear inequality of the form:*

$$\sum_{i \in L} [Y_{o_i} \geq k_i] \geq 1$$

for some $L \subseteq O$.

We call these generalized landmarks because any traditional disjunctive action landmark can be encoded as a generalized landmark by setting all $k_i = 1$.

Consider an instance of the simplified gripper domain shown in Figure 1 with 2 balls. The goal is to move both balls from the left room to the right, using a robot with a single "gripper" which can hold only one ball at a time. The robot starts in the right room, and can only pick-up and drop balls in the room it is currently occupying.

The operator count obtained by h^+ on this domain counts the following.

$\mathcal{C}(o) = 1 \quad \textbf{if } o \in \{move{-}l, move{-}r, grip{-}1{-}l,$
$\qquad\qquad grip{-}2{-}l, drop{-}1{-}l, drop{-}2{-}l\}$

$\mathcal{C}(o) = 0 \quad \textbf{otherwise}$

---

[1]Iverson brackets denote binary variables of the form $[P]$ that take the value 1 iff the condition P holds.

Note that there is only one occurrence of of the $move{-}r$ operator, however all feasible plans must contain two of this operator. We can add the constraint $[Y_{move{-}r} \geq 2] \geq 1$ to explain this requirement. With the addition of this constraint the MIP returns the optimal operator count for this instance.

If there existed an alternative path for the robot R to move between rooms the constraints are more complex. Consider an additional operator $move{-}r'$ identical to $move{-}r$. The constraint we described above would no longer be valid: it is possible to solve the planning problem with one of each of the two identical operators.

However adding both of the following constraints:

$$[Y_{move{-}r} \geq 2] + [Y_{move{-}r'} \geq 1] \geq 1$$

and

$$[Y_{move{-}r} \geq 1] + [Y_{move{-}r'} \geq 2] \geq 1$$

captures the requirement that two of the move operators must occur in any feasible plan. These can be read as "either $move{-}r$ occurs at least twice, or $move{-}r'$ occurs at least once"; and "either $move{-}r$ occurs at least once, or $move{-}r'$ occurs at least twice." The conjunction of these implies that a total of at least two of these operators must occur.

To enforce the correct behaviour of these bounds literals we need to add the following *domain constraints*[2] to our model:

$$[Y_o \geq k] \leq [Y_o \geq k - 1] \qquad \text{if } k > 0 \quad (1)$$

$$Y_o \geq \sum_{i=1}^{k} [Y_o \geq i] \qquad (2)$$

$$Y_o \leq M[Y_o \geq k] + k - 1 \qquad (3)$$

Where M is a sufficiently large number such that no feasible plan could contain more than M of any individual operator. In practice this number need only be as large as the longest plan the solver could feasibly solve. Constraint 1 ensures that a bound can't hold unless the next smallest bound also holds; 2 ensures that if k bounds literals are set, then at least k operators must occur; and finally 3 ensures that if k or more operators occur, the bounds literal $[Y_o \geq k]$ must be set.

Note that the constraint

$$[Y_{move{-}r} \geq 1] + [Y_{move{-}r'} \geq 1] \geq 1$$

is semantically equivalent to the traditional landmark constraint

$$Y_{move{-}r} + Y_{move{-}r'} \geq 1$$

however the former has a tighter linear relaxation since $Y_o \geq [Y_o \geq 1]$ always holds for any o. For example, the linear relaxation could assign $[Y_o \geq 1] = 0.5$, $[Y_o \geq 2] = 0.5$, $Y_o = 1$. In this case the constraint $[Y_o \geq 1] \geq 1$ is violated, but $Y_o \geq 1$ is not. Consequently if any operator-counting heuristic uses bounds literals, it is always preferable to encode landmark constraints using the bounds literals.

---

[2]*domain constraints* reflect the fact that Y_o variables are finite domain variables, and the bounds literals we use are closely related to bounds literals used in lazy clause generation (Ohrimenko, Stuckey, and Codish 2009), where the same term is used

Lemma 1. *A SAS$^+$ problem's cost function c can be replaced by $c'(o) = max(c(o), \epsilon)$ where $\epsilon > 0$ leaving at least one identical optimal plan.*

Proof. There exists an upper bound on optimal plan length l. Either all actions are uniform cost which can be assumed to be 1, or there exists a minimum cost difference between operators $\delta = \min(c(o) - c(o')|o, o' \in O \wedge c(o) > c(o'))$.

If $0 < \epsilon < \frac{\delta}{l}$, the sum of ϵ-cost actions in an optimal plan must be less than δ, and thus can only change the cost-order of plans which are either both suboptimal, or of equal cost according to c. $\qquad\square$

Theorem 1. *For any solvable SAS$^+$ planning problem there exists a set of generalized landmark constraints (with the domain constraints for all the bounds literals involved) such that solving a MIP with these constraints will compute $h^*(s_0)$.*

Proof. By Lemma 1, we assume strictly positive action costs without loss of generality.

An optimal operator count $\mathcal{C}$ (which may initially be empty) can be obtained by solving the MIP. If $\mathcal{C}$ does not represent the projection of a plan, then the generalized landmark constraint:

$$\sum_{o \in O} [Y_o \geq \mathcal{C}(o) + 1] \geq 1$$

can be added.

This constraint can be read as "at least one operator must be applied at least one more time". This is clearly violated by $\mathcal{C}$, and can only possibly invalidate subsets of $\mathcal{C}$. If any strict subset of $\mathcal{C}$ were feasible, it would be optimal, consequently this new constraint changes the optimum solution without affecting the admissibility of the MIP.

There are only finitely many distinct operator counts with the same cost, and each iteration of this process invalidates exactly one operator count.

Consequently this process will eventually terminate with: an operator count that is the projection of an optimal plan, if any plan exists; an infeasible MIP; or an operator count containing more operators than states in the state space, implying that no solution exists. $\qquad\square$

This process will generate sufficiently many generalized landmarks to compute h^*, but each landmark invalidates only one new operator count. Consequently, using these landmarks in a heuristic would likely be inefficient. If we were to omit bounds literals for some operators from the landmark, it would invalidate many more operator counts. This is similar to traditional landmarks which are stronger when they contain a small subset of operators. To obtain smaller, more focused landmarks we turn to the conflict analysis built in to modern "Conflict-Directed Clause Learning" SAT solvers.

SAT Encoding for Operator Sequencing

Assumptions are a feature of most SAT solvers' incremental interfaces, these allow the user to temporarily assert unit clauses. Importantly, if the resulting formula including these unit clauses is not satisfiable, the final conflict in the unsatisfiability proof can always be re-written in terms of a subset of the assumptions. This conflict clause represents a necessary (though not in general sufficient) property required of any valid model. In our SAT encoding assumptions will be used to ensure that only the operators selected by the operator counting heuristic are actually used.

The most important high-level constraint in achieving this is the at-most-k constraint (denoted $\leq_k$). $\leq_k (S)$ enforces that k or fewer literals from a set S are simultaneously true. This is a well studied constraint in satisfiability, and we use the sequential counter encoding by Sinz (2005), which is identical to the $O(n)$ encoding of Rintanen (2006) in the $\leq_1$ case.

We denote by $X =_i x$ that $X = x$ holds after operator layer i and by o_i that operator o occurs in layer i.

Given an operator count $\mathcal{C}$ and a number of layers $L = \sum_{o \in O} \mathcal{C}(o)$, the following constraints for each layer l form the core SAT model:

$$\leq_1 (\{o_l | o \in O\})$$
$$\forall X \in V : \quad \leq_1 (\{X =_l x_i | x_i \in D(X)\})$$
$$\forall X = x \in s_0 : \quad X =_0 x$$
$$\forall o \in O : \quad \bigwedge_{X = x \in pre(o)} (\neg o_l \vee X =_{l-1} x)$$
$$\forall o \in O : \quad \bigwedge_{X = x \in post(o)} (\neg o_l \vee X =_l x)$$
$$\forall X = x \in V : \quad X =_{l+1} x \Rightarrow$$
$$X =_l x \vee \bigvee_{o \in prod(x)} o_l$$
$$\forall X = x \in s_* : \quad X =_L x \vee [\Sigma\mathcal{C}(o) \geq L + 1]$$
$$\forall o \in O : \quad \leq_{\mathcal{C}(o)} (\{o_l | l \in [1, L]\}) \vee [Y_o \geq \mathcal{C}(o) + 1]$$

Additionally we add the following assumptions: $\neg[\Sigma\mathcal{C}(o) \geq L + 1]$ (i.e. that the goal is achieved by layer L); and $\neg[Y_o \geq \mathcal{C}(o) + 1]$ for each operator o. Any resulting conflict clause will be written in terms of the negation of a subset of these assumptions.

The final conflict clause will thus contain $[\Sigma\mathcal{C}(o) \geq L + 1]$ and some subset of the bounds literals implied by the operator count. Specifically it will be of the form:

$$[\Sigma\mathcal{C}(o) \geq L+1] \vee [Y_{o_1} \geq \mathcal{C}(o_1)+1] \vee \cdots \vee [Y_{o_n} \geq \mathcal{C}(o_n)+1]$$

. This translates nearly directly to a generalized landmark cut by replacing $\vee$ with $+$ and appending ≥ 1. The only complication is the $\neg[\Sigma\mathcal{C}(o) \geq L + 1]$ literal, which we tackle by adding an artificial operator T with zero cost (representing the total operator count) to the MIP, constrained such that:

$$Y_T = \sum_{o \in O} Y_o$$

Using this new operator, we can replace the total operator count literal $[\Sigma \mathcal{C}(o) \geq L + 1]$ with the bounds literal for the artificial operator T:

$$[\Sigma \mathcal{C}(o) \geq L + 1] \equiv [Y_T \geq L + 1]$$

It should be noted that the SAT formula we describe only ensures that no more operators occur than were chosen in the operator count. Thus it can sequence any subset of an operator count, allowing it to be used with approximate solutions while guaranteeing that the same proof of admissibility as in Theorem 1 applies.

We described earlier the two "hand-made" generalized landmarks one would add in order to improve the delete relaxation in the gripper domain, however the first cut generated from the SAT solver was:

$$[Y_{grab-1-r} \geq 1] + [Y_{drop-1-l} \geq 1] + [Y_{move-r} \geq 2]$$
$$+ [Y_{drop-2-l} \geq 1] + [Y_{grab-2-r} \geq 1] + [Y_{move-l} \geq 2]$$
$$+ [Y_T \geq 7] \geq 1$$

In spite of the conflict analysis in MiniSAT (Eén and Sörensson 2004), this cut clearly contains irrelevant bounds literals, and "cut strengthening": removing irrelevant parts of generated cuts, will definitely be an important improvement to techniques using generalized landmarks. In some scheduling domains, cut strengthening has been shown to be responsible for orders of magnitude decreases in runtime (Ciré, Coban, and Hooker 2013).

Each operator omitted from a conflict roughly doubles the number of operator counts that conflict applies to, drastically decreasing the number of iterations needed to find a perfect operator count. In our experiments, most of the conflicts learnt included no more than 10% of the total operators. While this sounds good, in practice this still means many conflicts were over 200 operators long, so there is clear room for improvement.

Planning using Logic-Based Benders

Logic-Based Benders Decomposition (LBBD) (Hooker and Ottosson 2003) is an approach to decomposing combinatorial search problems into a master MIP and one or more combinatorial subproblems. The master and subproblems share some variables such that the sub problem becomes easier to solve or prove infeasible once those variables it shares with the master are fixed. Importantly, LBBD allows for mixing of different optimisation technologies which may be better suited to the master problem and subproblems.

The master problem represents a relaxation of the original problem, and the subproblem checks for and *explains* flaws in that relaxation. Explanations in this context are constraints on the variables in the master problem. By incrementally adding these explanations, the master problem incrementally approaches the true solution.

First the master MIP is solved, and the optimal values of the shared variables are taken from the master, and this optimal assignment is assumed within the subproblem, which is then solved. If the subproblem is satifiable then a solution to the original problem has been found. If, as in our case, the
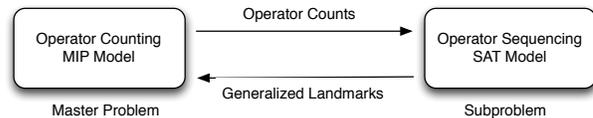

Figure 2: A Logic-Based Benders Decomposition Approach to Optimal Planning

objective function is fully modelled in the master problem, then this solution is optimal.[3]

If the subproblem is not satisfiable, some violated necessary condition on the shared variables is detected, and a constraint (the Benders cut) representing this condition is added to the master problem. The process is then iterated until the master problem's relaxation becomes satisfiable.

In our case, the master MIP is any operator-counting heuristic, the operator counts (strictly speaking the bounds literals) are shared variables, and the termination condition is that the optimal operator count is perfect.

Canonical planning (where each operator can be applied at most once) is NP-complete (Vidal and Geffner 2006), meaningfully easier than the full planning problem. Many domains in planning are canonically plannable, that is there exists a plan containing only one instance of each operator. Our subproblem, of sequencing the operators, is pseudo-polynomially reducible to a canonical planning problem, by replacing each operator o with $\mathcal{C}(o)$ copies of itself, since $\mathcal{C}(o)$ is usually small in optimal plans the subproblem is often much easier than the full problem in practice.

In our problem solving the master MIP problem takes considerably longer than the SAT subproblem. Hence we wish to consider how to speedup the LBBD solving process by using approximate answers to the master problem.

Solving the linear relaxation of the master MIP problem is considerably faster than solving it to integrality. Given an LP optimal solution to the linear relaxation, and rounding up the variables will simply increase the operator counts available to the sequencing subproblem (compared to the MIP optimal). Since the SAT subproblem only makes use of the upper bounds on operator counts, if it finds this relaxed subproblem is unsatisfiable, it will generate a cut which removes the current LP optimal. If on the other hand it finds a solution to this relaxed sequencing subproblem, then we have a feasible plan to the original problem. If this plan has the same cost as the lower-bound found by the LP, to the nearest integer, then we have optimally solved the planning problem. If the plan cost is more than the lower bound, this solution can be used to bound the search process: the MIP no longer needs to explore branches where the lower bound exceeds the cost of the best suboptimal plan found.

If a plan is found from a rounded-up operator count from an LP optimal, the MIP needs to branch before we can continue adding cuts. Importantly all modern MIP solvers

---

[3]In general, where the subproblem requires optimisation this is not true, but we omit this case for simplicity as it does not apply to our decomposition. See Hooker and Ottosson (2003).

provide user-specified cut generation and heuristic solution facilities via callbacks. We call our SAT sequencing procedure inside the python callback interface of Gurobi 5.6 (Gurobi Optimization 2013) if both the cardinality and objective of the rounded up operator count is within 20% of the linear count. We test for this to avoid generating SAT formulas for counts that are too far from the linear optimum, as the memory cost of adding layers to the formula is quite high. If we call the sequencing procedure, we either add a violated cut or a heuristic solution.

We do not modify Gurobi or MiniSAT's default branching behaviours, though it should be noted that state-of-the-art planning-as-satisfiability solvers use heuristics to simulate backward-chaining (Rintanen 2012). We expect similar improvements could be possible by tailoring branching strategies in a MIP solver for the operator counting problem.

There is one caveat to using callbacks to add cuts to the MIP: it is impossible to lazily add bounds literals during the MIP's search. Consequently we do two things: pre-allocate bounds literals up to $[Y_o \geq 2]$; and add a relaxed version of the cut where each absent literal $[Y_o \geq k]$ is replaced by Y_o/k. This happens in only a small percentage of cuts as even rounded-up operator counts rarely contain more than one of each operator. Cuts containing such Y_o/k terms are weaker than the normal landmarks, as for any value of Y_o:

$$[Y_o \geq k] \leq Y_o/k$$

However unless there is more than one missing bounds literal, this constraint still invalidates the current linear optimum. If a weakened constraint is generated that does not invalidate the current linear optimum, the MIP search is restarted after any weakened terms are replaced with the correct bounds literals.

This general approach of computing constraints violated by a close-to-optimal solution has many similarities with the improved landmark generation procedure of Haslum, Slaney, and Thiébaux (2012). In particular the incremental lower-bounding procedure h^{++} is very similar to our approach: both maintain a relaxation of the planning task (A delete-free problem with conjunctive conditions or an operator-counting MIP respectively); both incrementally refine their relaxation by finding flaws in the current relaxation's optimum (required conjunctive conditions or generalized landmarks respectively); and both exclusively find necessary properties of all plans, rather than focusing on extending promising plan prefixes, as in A^*-based planners.

Experiments

The main purpose of the experiments is to: experimentally validate Theorem 1; and to investigate how sequencing performs on a wide array of near optimal operator counts. We use IPC-2011 benchmarks, since our current prototype does not handle conditional effects required by IPC-2014 benchmarks, and since the implementation is preliminary it scales poorly to the significantly larger IPC-2014 benchmarks.

From the IPC-2011 benchmarks, we omit the `floorplan` domain as *hpp*'s parser rejects the domain file, and the `tidybot` and `parking` domains as

OpSeq's Python base heuristic encoder exceeds the 1-hour time limit in more than 90% of instances.

The initial MIP master problem contains constraints from the dynamically-merged flow heuristic (Bonet and van den Briel 2014) (including LM-Cut landmarks (Helmert and Domshlak 2009)), and the h+ base encoding of Imai and Fukunaga (2014).

Since sequencing considers all subsets of an operator count, rounding the linear optimum up makes for harder-to-sequence solutions than the true MIP optima most of the time. Nonetheless, Figure 3 shows that 99% of all the sequence calls take less than 1 second, although there is a significant long-tailed distribution: 0.01% of sequence calls took over 5 minutes. Some of this variance should be reduced by modifying the SAT solver's variable choice heuristics: the sequential counter implementation of $\leq_1$ which we use can be made significantly more robust using such techniques (Marques-Silva and Lynce 2007).

We show breakdowns for the sequence times in each domain in the left-hand part of Table 1. The "Seqs" column shows the number of sequence calls made in all instances of a domain. The "Dom SeqTime" columns show the average sequence times for all sequence calls made in that domain. The "Inst SeqTime" columns show the arithmetic mean of instance averages, this biases the results towards the behaviour seen in larger instances where fewer sequence calls occur. For example, in `barman`, the overall average sequence time was 0.38 seconds, but, as should be expected, most of the nearly 80,000 sequence calls occurred in easier instances so the average sequence times for each instance, treating each instance's average as a single data point show the average was 13 seconds. The first fifteen instances of `barman` have sub-second geometric mean sequence times, the largest five however have geometric means between 4 and 47 seconds, but under 400 sequence calls were made.

We observe similar sub-second geometric means in most domains in both cases, though the arithmetic means are noticeably larger in larger instances. We believe there are two reasons for this: firstly the sequence times include generating the SAT formula, which often takes longer than solving in the first sequence call; and secondly, earlier calls have fewer learned clauses to aid solving.

To evaluate our technique as a dual (incremental lower-bounding) algorithm, we use a dual of the standard IPC quality score, where instead of dividing the best known plan cost by the plan cost found by the planner, the lower bound found by the planner is divided by the best lower bound proved by any planner. We compare against *hpp*, a comparable incremental lower bounding solver; and *SymBA*-2* the winner of the most recent IPC optimal track (Torralba et al. 2014). We use the versions of both planners submitted to the IPC 2014 optimal track.

As observed by Haslum (2012), optimal planners using admissible heuristics in state based (or symbolic) search can also be used as lower-bounding procedures by observing the smallest f value of nodes in the open list. Since *SymBA**, like most other planners, regularly prints its best known lower bound, it is trivial to obtain a lower bound from its output, even if it has not successfully solved the planning

Benchmark	Seqs	Dom SeqTime		Inst SeqTime		OpSeq			Hpp			SymBA*-2		
		Arith	Geom	Arith	Geom	C	=	Q	C	=	Q	C	=	Q
barman	79392	0.38	0.06	13.06	4.84	0	0	9.37	0	0	9.14	**11**	**20**	**20.00**
elevators	7437	2.55	0.09	2.08	0.26	11	11	19.38	0	0	16.47	**19**	**20**	**20.00**
nomystery	8761	0.04	0.01	0.25	0.03	5	10	18.33	5	8	8.00	**15**	**18**	**19.82**
openstacks	1655	11.21	0.12	63.19	1.60	0	0	5.52	0	0	5.52	**20**	**20**	**20.00**
parcprinter	102	1.65	0.09	2.38	0.19	**20**	**20**	**20.00**	20	20	20.00	17	17	18.63
pegsol	91466	0.45	0.02	13.53	0.12	2	5	15.97	0	0	12.43	**19**	**20**	**20.00**
scanalyzer	57023	0.04	0.02	6.36	0.15	1	3	7.99	3	**14**	**18.93**	9	10	14.32
sokoban	121117	0.38	0.11	1.77	0.34	0	2	10.70	1	2	11.27	**20**	**20**	**20.00**
transport	5223	4.42	0.17	33.76	8.38	5	13	**19.47**	0	0	12.41	**11**	14	17.81
visitall	97	15.84	0.08	14.49	0.27	**14**	**20**	**20.00**	5	13	19.21	12	12	15.70
woodworking	160	11.83	0.61	11.57	0.97	**20**	**20**	**20.00**	18	18	19.95	19	19	19.74
Total	—	—	—	—	—	78	104	166.74	52	75	153.33	**172**	**189**	**206.02**

Table 1: Average sequence times, Coverage (C), Number of best bounds (=), and Dual quality scores (Q) for IPC-2011 sequential optimal track benchmarks.

problem. Table 1 shows for each solver the number of instances solved optimally (column "C"); the number of instances where the solver found the best bound of any solver (column "="); and the dual quality score described above (column "Q").

We see from these results that *SymBA*-2* is extremely effective, beating both dual techniques in all three metrics in all but 5 domains, although in 4 of these 5 domains, *OpSeq* earns the best dual quality score, and in 3 domains even beats *SymbA** in coverage. *OpSeq* also beats the previous state-of-the-art in incremental lower bounding in 9 of the 11 domains investigated.

We see from the quality scores in the "Q" columns, that even when *OpSeq* fails to solve, it usually finds good lower bounds, this can be seen particularly in the `nomystery` and `transport` domains. Just as in primal techniques, it is far easier to find a plan than to prove that a plan is optimal, so in a dual technique it is much easier to find h^* than it is to prove that we have found it by finding a plan.

Combining primal with dual techniques are among the most effective optimisation techniques: commercial mixed integer programming solvers use both lower bounding from the linear relaxation; and sophisticated primal heuristics to find solutions close to the linear optimum. It could be argued that this is to some extent what *SymBA** does too: it interleaves spending time on improving abstractions (dual bounds) with searching in the original search space (Edelkamp, Kissmann, and Torralba 2012).

Related Work

We have discussed in several places the relationship between our work and h^{++}, being the only other dedicated incremental lower-bounding procedure the authors are aware of.

The only other approach using SAT-based planning techniques in cost-optimal planning that the authors are aware of uses MaxSAT combined with a SAT encoding of the delete relaxation (Robinson et al. 2010). SAT planning has also been used to generate upper bounds to improve performance of state-based search (Robinson, Gretton, and Pham 2008).

There have also been a number of "Optimal" SAT encodings, which find cost-optimal plans only when those plans
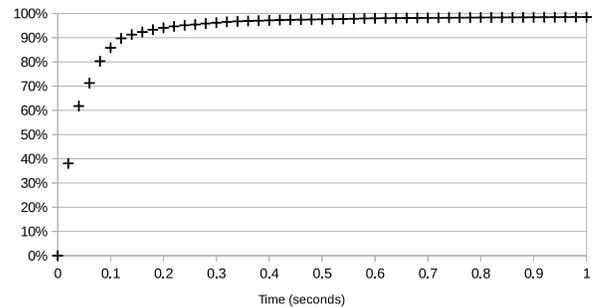

Figure 3: Cumulative Frequency of Sequence Times

happen to have the same makespan as the makespan-optimal plan (Chen, Lu, and Huang 2008; Giunchiglia and Maratea 2010). However this is clearly not the same as true cost-optimal planning.

Other generalizations of landmarks have been proposed including "multivalued landmarks" (Zhang et al. 2013), which identify operator sets which must collectively be executed more than once: $\sum Y_o \geq k$. These can be encoded in our generalized landmarks using an artificial operator to group the operators in the landmark (just as for the total operator count) leading to a constraint like $[\sum Y_o \geq k] \geq 1$; or directly as set of $O(\binom{N}{k-1})$ "standard" generalized landmarks.

There exist other heuristics bounded by h^*, including many heuristics enhanced by the P^C and P_{ce}^C compilations (Haslum 2012; Keyder, Hoffmann, and Haslum 2012) and abstraction heuristics such as symbolic-merge-and-shrink (Torralba, López, and Borrajo 2013). Symbolic-merge-and-shrink like most heuristics in optimal planning provides only a lower bound to guide search; whereas if an optimal operator count is sequenced successfully, the planning problem is solved. Similar behaviour is also seen in the P^C and P_{ce}^C compilations when no flaws can be extracted, the planning problem is solved optimally.

There are more sophisticated planning-as-SAT encodings

that we could have added our counting constraint to, in particular the $\exists$-step and $\forall$-step encodings (Rintanen, Heljanko, and Niemelä 2006; Wehrle and Rintanen 2007), and the SASE encoding (Huang, Chen, and Zhang 2012). However the absence of a tighter upper bound than simply the total number of operators required to refute any possible sequence of an operator count made the core advantage of these encodings less obvious. It would be interesting to compare these base encodings with ours, and a theorem giving such an upper bound would likely be an important breakthrough for the LBBD approach to planning.

Conclusions and Further Work

We have defined a simple generalization of landmarks which allow the encoding of admissible heuristics upper-bounded only by h^*, rather than h^+. We also introduce a SAT-based, complete algorithm for generating a generalized landmark violated by a given operator count which is usually very fast. We experimentally confirmed that h^* can be computed using only this algorithm, and demonstrated that it outperforms the previous state-of-the-art in incremental lower bounding: h^{++}.

Our approach can usually generate violated constraints from solutions to the linear relaxation of an operator count heuristic, rather than the NP-hard MIP. Importantly, if such a cut can be generated, it is guaranteed to invalidate the current linear optimum, and the current rounded solution, ensuring such heuristically generated cuts are relevant. This is in contrast to a similar improvement to a complete algorithm for generating traditional landmarks (Haslum, Slaney, and Thiébaux 2012), which relies on approximate hitting sets, with no guarantee that the generated landmark invalidates the current optimum hitting set, nor indeed that the landmark will change the next approximate hitting set generated. This suggests a more traditional use for our generalized landmark generation algorithm: applying our algorithm to the delete relaxation can generate traditional landmarks, and the relative performance of this approach would be interesting to investigate.

There are other more conventional applications for generalized landmarks as well, such as pre-processing to generate an initial set of generalized landmarks which can then be used in an analogue of Incremental LM-Cut (Pommerening and Helmert 2013). We expect this to provide improved heuristic guidance near the root of the search where it is most valuable. While we use a complete algorithm to generate landmarks, there is an obvious fast but incomplete algorithm obtained by simply terminating early when long-tailed behaviour is observed.

Such an approach could also potentially be used as a kind of look-ahead in an optimal planner: if an operator counting heuristic returns a sufficiently small operator count, our approach could test if the operator count is sequenceable, and if so, terminate the search early with an optimal plan.

We plan to investigate an extension to our approach which explains the states in which generalized landmarks apply, such that landmarks generated in the initial state can be easily re-used in a state-based search when they become applicable again.

However we chose to explore the more novel LBBD approach to planning using generalized landmarks in the hope that this decomposition between counting and sequencing will lead to cost-optimal planning algorithms capable of handling richer constraints such as numeric state variables, resources and scheduling constraints as in (Hoffmann et al. 2007; Gregory et al. 2012; Nareyek et al. 2005).

We believe this approach is interesting and promising because it allows a principled interaction between state-of-the-art heuristics, and explanation-based combinatorial search approaches including SAT, SMT and LCG. Any constraint capable of explaining its inferences can be added to the SAT subproblem, potentially allowing direct integration of cost-optimal planning with SMT and state-of-the-art scheduling approaches based on constraint programming with lazy clause generation. This means that, by extending the approach we present, we should be able to solve similar problems to Planning Modulo Theories (Gregory et al. 2012) by taking advantage of the extensive range of *existing* theories and constraints *already implemented* by the SMT and constraint programming communities.

Acknowledgements

References

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *European Conference on Artificial Intelligence*, 329–334.

Bonet, B., and van den Briel, M. 2014. Flow-based heuristics for optimal planning: Landmarks and merges. In *International Conference on Automated Planning and Scheduling*, 47–55.

Bonet, B. 2013. An admissible heuristic for SAS+ planning obtained from the state equation. In Rossi, F., ed., *International Joint Conference on Artificial Intelligence*, 2268–2274. IJCAI/AAAI.

Chen, Y.; Lu, Q.; and Huang, R. 2008. Plan-A: A cost-optimal planner based on SAT-constrained optimization. In *International Planning Competition*.

Ciré, A.; Coban, E.; and Hooker, J. N. 2013. Mixed integer programming vs. logic-based Benders decomposition for planning and scheduling. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer. 325–331.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. A hybrid relaxed planning graph-LP heuristic for numeric planning domains. In *International Conference on Automated Planning and Scheduling*, 52–59.

Edelkamp, S.; Kissmann, P.; and Torralba, A. 2012. Symbolic A* Search with Pattern Databases and the Merge-and-Shrink Abstraction. In *European Conference on Artificial Intelligence*, 306–311.

Eén, N., and Sörensson, N. 2004. An extensible SAT-solver. In *Theory and applications of satisfiability testing*, 502–518. Springer.

Giunchiglia, E., and Maratea, M. 2010. Introducing preferences in planning as satisfiability. *Journal of Logic and Computation* 205–229.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning modulo theories: Extending the planning paradigm. In *International Conference on Automated Planning and Scheduling*, 65–73.

Gurobi Optimization, I. 2013. Gurobi optimizer reference manual.

Haslum, P.; Slaney, J.; and Thiébaux, S. 2012. Minimal landmarks for optimal delete-free planning. In *International Conference on Automated Planning and Scheduling*, 353–357.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In *International Conference on Automated Planning and Scheduling*, 74–82.

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

Hoffmann, J.; Gomes, C. P.; Selman, B.; and Kautz, H. A. 2007. SAT encodings of state-space reachability problems in numeric domains. In *International Joint Conference on Artificial Intelligence*, 1918–1923.

Hooker, J. N., and Ottosson, G. 2003. Logic-based Benders decomposition. *Mathematical Programming* 96(1):33–60.

Huang, R.; Chen, Y.; and Zhang, W. 2012. SAS+ planning as satisfiability. *Journal of Artificial Intelligence Research* 43(1):293–328.

Imai, T., and Fukunaga, A. 2014. A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *European Conference on Artificial Intelligence*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 459–464. IOS Press.

Katz, M., and Domshlak, C. 2008. Optimal additive composition of abstraction-based admissible heuristics. In *International Conference on Automated Planning and Scheduling*, 174–181.

Keyder, E. R.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In *International Conference on Automated Planning and Scheduling*, 128–136.

Marques-Silva, J., and Lynce, I. 2007. Towards robust cnf encodings of cardinality constraints. In *Principles and Practice of Constraint Programming–CP 2007*. Springer. 483–497.

Nareyek, A.; Freuder, E. C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H.; Rintanen, J.; and Tate, A. 2005. Constraints and ai planning. *Intelligent Systems, IEEE* 20(2):62–72.

Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(*T*). *J. ACM* 53(6):937–977.

Ohrimenko, O.; Stuckey, P.; and Codish, M. 2009. Propagation via lazy clause generation. *Constraints* 14(3):357–391.

Pommerening, F., and Helmert, M. 2013. Incremental lm-cut. In *International Conference on Automated Planning and Scheduling*, 162–170.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *International Conference on Automated Planning and Scheduling*, 226–234.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12):1031–1080.

Rintanen, J. 2006. Compact representation of sets of binary constraints. *Frontiers in Artificial Intelligence and Applications* 141:143.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Robinson, N.; Gretton, C.; Pham, D.; and Sattar, A. 2010. Partial weighted MaxSAT for optimal planning. In Zhang, B.-T., and Orgun, M., eds., *PRICAI 2010: Trends in Artificial Intelligence*, volume 6230 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 231–243.

Robinson, N.; Gretton, C.; and Pham, D.-N. 2008. Co-plan: Combining SAT-based planning with forward-search. *Proceedings of the Sixth International Planning Competition (IPC-6)*.

Sinz, C. 2005. Towards an optimal cnf encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming-CP 2005*. Springer. 827–831.

Torralba, A.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA^*: A symbolic bidirectional A^* planner. In *Proceedings of the Eighth International Planning Competition (IPC-2014)*, 105–108. AAAI Press.

Torralba, A.; López, C. L.; and Borrajo, D. 2013. Symbolic merge-and-shrink for cost-optimal planning. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 2394–2400. AAAI Press.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In Bessière, C., ed., *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *Lecture Notes in Computer Science*. Springer. 651–665.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence* 170(3):298–335.

Wehrle, M., and Rintanen, J. 2007. Planning as satisfiability with relaxed ∃-step plans. In Orgun, M., and Thornton, J., eds., *AI 2007: Advances in Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 244–253.

Zhang, L.; Wang, C.-J.; Wu, J.; Liu, M.; and Xie, J.-y. 2013. Planning with multi-valued landmarks. In *Proceedings of the Twenty-Seventh Conference on Artificial Intelligence (AAAI-13)*, 1653–1654.