Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling

Modeling and Computation in Planning: Better Heuristics from More Expressive Languages

Guillem Francès
Universitat Pompeu Fabra
Barcelona, Spain
guillem.frances@upf.edu

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain
hector.geffner@upf.edu

Abstract

Most of the key computational ideas in planning have been developed for simple planning languages where action preconditions and goals are conjunctions of *propositional* atoms. Preconditions and goals that do not fit into this form are normally converted into it either manually or automatically. In this work, we show that this modeling choice hides important structural information, resulting in poorer heuristics and weaker planning performance. From a theoretical point of view, we show that the direct generalization of *relaxed planning graph heuristics* to more expressive languages that implicitly allow *conjunctions of atoms with more than one state variable* leaves open a crisp gap, as it fails to properly account for the constraints over these variables. The simple propositional languages that are standard in planning do not remove this gap but "hide it under the rug" by forcing atoms to be of the form $X = c$, where c is a constant and X is a (usually boolean) state variable. Closing this gap in the computation of the relaxed planning graph for more expressive languages leads to a *more accurate but intractable* heuristic, yet a cost-effective tradeoff can be achieved using local forms of *constraint propagation* that result in better heuristics, better plans, and a more effective search. We show this empirically over a diverse set of illustrative examples using a fragment of the Functional STRIPS planning language.

Introduction

Consider a simple planning problem involving a set of integer variables $X_1, \ldots, X_n$ and actions that allow us to increase or decrease by one the value of any variable within the $[0, n]$ interval. Initially, all variables have value 0, and the goal is to achieve the inequalities $X_i < X_{i+1}$, for $i \in [0, n-1]$. A possible encoding of this problem in the standard classical planning languages would feature an atom $val(i, k)$ for each equality $X_i = k$, an atom $less(i, j)$ for each inequality $X_i < X_j$, a goal given by the conjunction of the atoms $less(i, i+1)$, $i \in [0, n-1]$, and an initial situation given by atoms $val(i, 0)$. Computationally, if P_i is the i-th propositional layer of the relaxed planning graph (RPG) corresponding to the initial state (Hoffmann and Nebel 2001), with P_0 being the initial layer, it is easy to see that layer P_1 will make true each of the goal atoms $less(i, i+1)$, since a single increment is needed to make each of these atoms true. The initial state of the problem thus has an $h_{\max}$ heuristic value of 1 (Bonet and Geffner 2001) and an h_{FF} value of $n-1$, whereas actually the shortest plan for the problem has $1 + 2 + \cdots + n - 1 = n(n-1)/2$ steps. This inability of the heuristics to provide a better approximation is not particularly remarkable; what is interesting is that this happens because the heuristic assumes that the goals $less(i, i+1)$ are independent when they are not. The additive heuristic (Bonet and Geffner 2001) actually makes this independence assumption explicit.

When delete-relaxation heuristics are analyzed over a more expressive encoding of this domain featuring numeric or multivalued state variables (Hernádvölgyi and Holte 1999; Rintanen and Jungholt 1999; Hoffmann 2003; Coles et al. 2008; Helmert 2009), a different picture emerges. It is well known that relaxed planning graph heuristics can be generalized to languages featuring multivalued variables plus arbitrary formulas in action preconditions and goals by following the so-called *value-accumulating semantics* (Gregory et al. 2012; Ivankovic et al. 2014). In this semantics, each state variable X has a domain $D(X)$ of possible values in each propositional layer P_k that grows as action effects supporting new values are triggered. The truth of a precondition, condition, or goal formula in layer P_k is defined inductively in the usual form from the truth of the atoms involving such variables. For example, an atom like $X < Y$ is deemed as true in layer P_k if there are values x and y for X and Y such that $x < y$. Similarly, a *conjunction* of atoms is deemed true in P_k if each atom in the conjunction is true in P_k. Applying this *value-accumulating semantics* to our problem, we find that each of the atoms $X_i < X_{i+1}$ is true in layer P_1, thus yielding the same heuristic assessments as in the propositional encoding. In the more expressive encoding, however, it is possible to see that there is a problem in the derivation of the heuristic that is not the result of the "delete-relaxation" but of the value-accumulating semantics. Namely, while it is correct to regard *each* of the atoms $X_i < X_{i+1}$ as true in layer P_1, where all variables have domain $\{0, 1\}$ and hence there are values for X_i and X_{i+1} that satisfy $X_i < X_{i+1}$, it is not correct to regard the *conjunction of all of them* as true, since there are no values for the state variables in the domains $D(X_i) = \{0, 1\}$ that can satisfy all the atoms $X_i < X_{i+1}$ *at the same time* (if $n > 2$). Indeed, only at layer P_{n-1} (where the domains of all variables are $D(X_i) = [0, n-1]$) can *all* the atoms $X_i < X_{i+1}$ be satisfied at the same time by means of the valuation that assigns each variable X_i the value $i - 1$.

From a logical perspective, it is possible to see that the value-accumulating semantics is too weak because it makes *two simplifications*, not just one. One simplification is *monotonicity* by which the variable domains $D(X)$ grow monotonically as new values for variable X become reachable, in line with the notion of "delete-relaxation". Yet there is a second simplification, namely, *decomposition*, by which a *conjunction* of atoms is regarded as *true* in a propositional layer whenever *each one* of the atoms in the conjunction is true. Like monotonicity, decomposition is *not* true in general. The reason is that a propositional layer encodes *not just one but a set of possible interpretations over the language*. In any *single* logical interpretation, the truth value of a conjunction is a function of the truth values of its conjuncts. But when there is a *set of interpretations*, it is possible that one interpretation makes one atom true, a second interpretation makes a second atom true, and yet no interpretation makes the two atoms true at the same time.

In our example, the domains $D(X_i) = \{0, 1\}$ associated with the state variables X_i in the propositional layer P_1 implicitly encode 2^n possible logical interpretations where each state variable X_i can have one of the two values. In this set of interpretations, for *any* atom $X_i < X_{i+1}$ at least one interpretation makes the atom true, yet no interpretation in the set makes the *conjunction* of all such atoms true.

Decomposition is the assumption that if there are interpretations that satisfy *each* of the atoms in a conjunction, there are also interpretations that satisfy *all* of the atoms in the conjunction. The set of possible interpretations corresponding to a layer P_k of the planning graph is determined by the set of values $D(X_i)$ that are possible for each of the state variables in that layer. It turns out that *decomposition is valid* when no two atoms in the conjunction involve the same state variable, *e.g.* in conjunctions such as $(X_1 > 3 \wedge X_2 < 2)$ or $(X_1 = true \wedge X_2 = true)$, where the conjuncts involve different state variables X_i. This class of conjunctions actually subsumes the language fragment that is standard for classical planners, in which preconditions, conditions, and goals are conjunctions of propositional atoms p, like $on(b_1, b_2)$, that can be taken as abbreviations for atoms $p = true$, p being a boolean state variable. In such conjunctions, no boolean state variable is mentioned more than once. This language fragment also subsumes the *restricted numeric planning tasks* in `Metric-FF` where atoms can contain at most one numeric variable, and hence can be of the form $X = c$ or $X > c$, where c is a constant, but not of the form $X > Y$, where both X and Y are numeric state variables.

The fact that decomposition is valid for simple languages where goals, preconditions, and conditions are conjunctions of atoms involving one state variable each, does *not* imply that such languages are more convenient for modeling and problem-solving than richer languages that do not enforce this restriction. On the contrary, *if the goal of the real problem contains different non-unary atoms involving the same state variables, a better alternative is to acknowledge the constraints between them in the language and to use such constraints to derive more informed heuristics*.

Our aims in this paper are thus (1) to advocate the use of more expressive planning languages for modeling, (2) to point to the gap left open by the value-accumulating semantics in its failure to account for the constraints imposed on state variables by conjunctive expressions, (3) to partially close this gap by using forms of constraint propagation in the definition and computation of the heuristics, and (4) to show that such heuristics can be cost-effective.

With these goals in mind, the rest of this paper is organized as follows. We first review Functional STRIPS, an expressive planning language where atoms can be arbitrary, variable-free first-order atoms (Geffner 2000). We then define the direct generalization of the h_{max} and h_{FF} heuristics for such a language following (Gregory et al. 2012; Ivankovic et al. 2014), based on the assumptions of *monotonicity* and *decomposition*, and introduce a stronger, constrained generalization that *retains monotonicity but avoids decomposition*. We finally define a polynomial approximation of this stronger but intractable heuristic and test it over a number of examples.

While the importance of more expressive planning languages for modeling is well-known (Gregory et al. 2012), our emphasis is mainly on the computational value of such extensions, and their use for understanding the limitations and possible elaborations of current heuristics. Some of the language extensions that we consider, however, such as the use of *global constraints* (van Hoeve and Katriel 2006), are novel in the context of planning and interesting on their own. The planning language also makes room for using predicate and function symbols whose denotation is fixed, and which can be characterized either *extensionally* by enumeration, or *intensionally* by means of procedures or *semantic attachments* (Dornhege et al. 2009). Yet, while the planning language accommodates constraints and semantic attachments, and the planning heuristics accommodate forms of constraint propagation, we hope to show that these are not add-ons but *pieces that fall into place from the logical analysis of the language and the heuristic computation.*

Functional STRIPS

Functional STRIPS (FSTRIPS) is a general modeling language for classical planning based on the quantifier-free fragment of first-order-logic involving *constant*, *function* and *relational or predicate symbols* but no variable symbols. We review it following (Geffner 2000).

Syntax

FSTRIPS assumes that *fluent* symbols, whose denotation may change as a result of the actions, are all *function* symbols. Fluent constant symbols can be seen as arity-0 function symbols, and fluent relational symbols as *boolean* function symbols of the same arity plus equality. For example, BLOCKSWORLD atoms like $on(a, b)$ can be encoded in FSTRIPS as $on(a, b) = true$, by making on a functional symbol, or in this case, more conveniently, as $loc(a) = b$ where loc is a function symbol denoting the block location.

Constant, functional and relational symbols whose denotation does not change are called *fixed* symbols. Among them, there is usually a finite set of object names, and constant, function, and relational symbols such as '3', '+' and '=', with the standard interpretation.

Terms, atoms, and formulas are defined from constant, function, and relational symbols in the standard way, except that in order for the representation of states to be finite and compact, the symbols, and hence the terms are typed. A *type* is given by a finite set of fixed constant symbols. The terms $f(t)$ where f is a fluent symbol and t is a tuple of fixed constant symbols are called *state variables*, as the state is actually determined by the value of such "variables".

An action a is described by the type of its arguments and two sets: the *precondition* and the *effects*. The precondition $Pre(a)$ is a formula, and the effects are *updates* of the form $f(t) := w$, where $f(t)$ and w are terms of the same type, f is a fluent symbol, and t is a tuple of terms. The updates express how fluent f changes when the action is taken. Conditional effects $C \rightarrow f(t) := w$, where C is a formula (possibly $C = true$), can be defined in a similar manner.

As an example, the action of moving a block b onto another block b' can be expressed by an action $move(b, b')$ with precondition $clear(b) = true \wedge clear(b') = true$, and effects $loc(b) := b'$ and $clear(loc(b)) := true$. In this case, the terms $clear(b)$ and $loc(b)$ for blocks b stand for state variables. The term $clear(loc(b))$ is not a state variable, as $loc(b)$ is not a fixed constant symbol. For any state variable X, we commonly abbreviate $X = true$ and $X = false$ as X and $\neg X$, and express conjunctions of atoms by the set of atoms in the conjunction. A FSTRIPS planning problem is a tuple $\langle F, I, O, G \rangle$, where I is a set of literals defining the initial situation, G is the goal formula, O is a set of actions, and F describes the symbols and their types.

Semantics

States represent logical interpretations over the language of FSTRIPS. The denotation of a symbol or term t in the state s is written as t^s. The denotation r^s of fixed symbols r does not depend on the state and it is written r^*. The denotation of standard fixed symbols like '3', '+', '=' is assumed to be given by the underlying programming language, while object names c are assumed to denote themselves so that $c^* = c$. The denotation of fixed (typed) function and relational symbols can be provided extensionally, by enumeration in the initial situation, or intensionally, by attaching actual functions to them (Dornhege et al. 2009).

Since the only fluent symbols are function symbols, and the types of their arguments are all finite, the (dynamic part of the) state can be represented as the value of a finite set of state variables $f(t)$, where f is a functional fluent and t is a tuple of fixed constant symbols. From the fixed denotation r^* of fixed symbols r, and the changing denotation of fluent symbols f captured by the values $[f(t)]^s$ of the state variables $f(t)$ associated with f, the denotation of arbitrary terms, atoms, and formulas follows in the standard way. The denotation t^s of any term not involving functional fluents, expressed also as t^*, is c^* if t is a constant symbol or, recursively, $g^*(t_1^*)$ if t is the compound term $g(t_1)$ where t_1 is a

```
types:  block, cell, direction

functions:
  loc(b: block):  cell
  next(x: cell, d: direction): cell

action move(b: block, d: direction)
  prec in-grid(next(loc(b), d))
  eff  loc(b) := next(loc(b), d)

goal:
  loc(b1) ≠ loc(b2) ∧ loc(b1) ≠ loc(b3) ∧
  loc(b2) = loc(b3)
```

Figure 1: Fragment of the GROUPING domain, where the goal is to group together blocks of the same color. The example shows three blocks b_1, b_2 and b_3 of two different colors; the static *color* predicate has been manually compiled away.

tuple of terms. Similarly, the denotation t^s of a term $f(t_1)$ where f is a fluent functional symbol is defined recursively as the value $[f(c)]^s$ of the *state variable* $f(c)$ in s where c is the tuple of constant symbols that name the tuple of objects t_1^s; i.e., $c^* = t_1^s$. In the same way, the denotation $[p(t)]^s$ of an atom $p(t)$ is $true/false$ iff the result of applying the boolean function p^* to the tuple of objects t^s yields $true/false$. The truth value B^s of the formulas B made up of such atoms in the state s follows then the usual rules.

An action a is applicable in a state s if $[Pre(a)]^s = true$. The state s_a that results from the action a in s satisfies the equation $f^{s_a}(t^s) = w^s$ for all the updates $f(t) := w$ that the action a triggers in s, and is otherwise equal to s. This means that the update changes the value of the *state variable* $f(c)$ to w^s iff the action triggers an update $f(t) := w$ in the state s for which $c^* = t^s$. For example, if $X = 2$ is true in s, then the update $X := X + 1$ increases the value of X to 3 without affecting other state variables. Similarly, if $loc(b) = b'$ is true in s, the update $clear(loc(b)) := true$ in s is equivalent to the update $clear(b') := true$.

A plan for a problem $\langle F, I, O, G \rangle$ is a sequence of applicable actions from O that maps the unique initial state where I is true into one of the states where G is true.

Modeling

The problem described in the introduction, which we call COUNTERS, can be encoded in FSTRIPS by modeling the integer variables X_i with a unary functional fluent val such that $val(i)$ denotes the value of X_i in the $[0, n]$ range. Thus, $val(1), \ldots, val(n)$ are the only state variables of the problem, and actions $increment(i)$ and $decrement(i)$ update their values. The goal is then expressed as a conjunction $\bigwedge_{i=0}^{n-1} val(i) < val(i+1)$.

A more interesting domain, which we call GROUPING, features several blocks of different colors lying on a grid, and the only available actions allow us to move one block to a neighboring cell, with no other preconditions (thus, a cell can accommodate an arbitrary number of blocks). The objective is to group the blocks by color, *i.e.* to position all blocks in a way such that two blocks are on the same cell iff they have the same color. Figure 1 shows part of the FSTRIPS problem specification, where *loc* is a fluent function denoting the position of any block, and $next$ and $in\text{-}grid$ are fixed functions representing the topology of the grid. A key feature of FSTRIPS is that fluent functional terms allow bypassing the restriction imposed by propositional languages that objects be referred to by their unique names (Geffner 2000). Thus we can use the goal expression $loc(b_1) \neq loc(b_2) \wedge loc(b_1) \neq loc(b_3) \wedge loc(b2) = loc(b_3)$, which cannot be modeled in standard PDDL without using an exponentially long encoding.

Relaxed Planning Graph

As noted by several authors, some of the heuristics that are useful in STRIPS like h_{max} and h_{FF} can be generalized to more expressive languages by means of the so-called *value-accumulating semantics* (Hoffmann 2003; Gregory et al. 2012; Ivankovic et al. 2014). In this interpretation, each propositional layer P_k of the relaxed planning graph keeps for each state variable X a set X^k of values that are possible in P_k. Such sets are used to define the *sets y^k of possible values or denotations of arbitrary terms, atoms, and formulas y*, and from them, the sets of possible values X^{k+1} for the next layer P_{k+1}. For layer P_0, $X^0 = \{X^s\}$, where s is the state for which the heuristic is sought. From the sets of possible values X^k for the state variables X in layer P_k, the set of possible denotations t^k of any term not involving functional fluents is $t^k = \{t^*\}$, while the set of possible denotations $[f(t)]^k$ for terms $f(t)$ where f is a fluent symbol is defined recursively as the *union* of the sets $[f(c)]^k$ where $f(c)$ is a state variable such that $c^* \in t^k$. In a similar way, the set of possible denotations $[p(t)]^k$ of an atom $p(t)$ in layer P_k includes the value *true* (*false*) iff $p^*(c^*) = true$ (*false*, respectively) for some tuple $c^* \in t^k$. The sets of possible denotations of disjunctions, conjunctions, and negations are defined recursively so that *true* is in $[A \vee B]^k$, $[A \wedge B]^k$ and $[\neg A]^k$ iff *true* is in A^k or in B^k, *true* is in both A^k and B^k, and *false* is in A^k respectively. Similarly, *false* is in $[A \vee B]^k$, $[A \wedge B]$ and $[\neg A]^k$ iff *false* is in both A^k and B^k, *false* is in A^k or in B^k, and *true* is in A^k respectively.

The set of possible values X^{k+1} for the state variable X in layer P_{k+1} is the union of X^k and the set of possible values x for X that are supported by conditional effects of actions a whose preconditions are possible in P_k, i.e., $true \in [Pre(a)]^k$. A conditional effect $C \rightarrow f(t) := w$ of a supports value x of X in P_k iff $X = f(c)$ for some tuple of constant symbols c such that $c^* \in t^k$ and $x \in w^k$.

This finishes the definition of the sequence of propositional layers $P_0, \ldots, P_k$ that make up the RPG for a given problem P and state s. When computing the heuristics h_{max} and h_{FF}, the computation stops in the first layer P_k where the goal formula G is true, *i.e.* $true \in G^k$, or where a fixed point has been reached without rendering the goal true, *i.e.* $X^k = X^{k+1}$ for all the state variables. In the second case, $h_{max}(s) = h_{\text{FF}}(s) = \infty$ as one can show that there is no plan for P from s. In the first case, $h_{max}(s) = k$, and a relaxed plan $\pi_{\text{FF}}(s)$ can be obtained backward from the goal by keeping track of the state variables X and values $x \in X^k$ that make the goal true, the actions a and effects $C \rightarrow f(t) := w$ supporting such values first, and iteratively, the variables and values that make $Pre(a)$ and C true. The heuristic $h_{\text{FF}}(s)$ is given by the number of different actions a in $\pi_{\text{FF}}(s)$ with each action a counted as many times as layers in $\pi_{\text{FF}}(s)$ where it is used, in accordance with the treatment of conditional effects in `FF`.

Under some restrictions, it is possible to show that $\pi_{\text{FF}}(s)$ is indeed a plan for a relaxation P' of P from the state s where "assignments" do not erase the "old" values of state variables, and hence where atoms like $X = x$ and $X = x'$ for $x \neq x'$ are not mutually exclusive and can both be true. In the COUNTERS example, for $n = 3$ the goal G is $(X_1 < X_2) \wedge (X_2 < X_3)$, the initial state s is such that $X_1^s = X_2^s = X_3^s = 0$, and actions increment or decrement each variable within the $[0, 3]$ range. We found that $h_{max}(s) = 1$ for this problem, as in layer P_k with $k = 1$, the possible set of values for the three variables is $X_1^k = X_2^k = X_3^k = \{0, 1\}$. This implies that $true \in [X_1 < X_2]^k$ as there are constants 0 and 1 in X_1^k and X_2^k such that $0 < 1$. Similarly, $true \in [X_2 < X_3]^k$, so we get $true \in G^k$ for $k = 1$. In this relaxation, variable X_1 can have *both values* 0 *and* 1 *at the same time*, using 0 to make the first goal true and 1 to make the second goal true. Indeed, in this relaxation, *self-contradictory goals like $X_1 = 0 \wedge X_1 = 1$ are achievable in one step as well.*

Constrained Relaxed Planning Graph

A weakness of RPG heuristics is the assumption that *state variables can take several values at the same time.* This simplification does not follow from the *monotonicity assumption* that underlies the value-accumulating semantics but from the way the sets of *possible* values X^k in layer P_k are used. The fact that these various values are all regarded as possible in layer P_k does not imply that they are *jointly* possible. The way to retain monotonicity in the construction of the planning graph while removing the assumption that a state variable can take several values at the same time is *to map the domains X^k of the state variables into a set V^k of possible interpretations over the language*. Indeed, given that an interpretation s over the language is determined by the values X^s of the state variables (Section 2), this set V^k is nothing but the set of interpretations v that result from selecting *one value* X^v for each state variable X among the set of values X^k that are possible for X in layer P_k.

As before, $X^0 = \{X^s\}$ when s is the seed state, and X^{k+1} contains all the values in X^k along with the set of possible values x for X supported by the effects of actions a whose preconditions are possible in P_k. However, a formula like $Pre(a)$ is now possible in P_k iff there is an interpretation $v \in V^k$ s.t. $[Pre(a)]^v = true$. Moreover, a conditional effect $C \rightarrow f(t) := w$ of a supports the value x of X in P_k iff there is an interpretation $v \in V^k$ where $[Pre(a)]^v$ and C^v are *true*, $x = w^v$, and $X = f(c)$ for $c^* = t^v$.

This alternative, logical interpretation of the propositional layers P_i affects the contents and computation of the RPG, keeping the assumption that the set of possible values of a state variable grows *monotonically*, but dropping the assumption of *decomposability*, that holds that such values are jointly possible. To compute the heuristics h^*_{max} and h^*_{FF}, the construction of the RPG stops at the first layer P_k where the goal formula G is satisfiable, *i.e.* where G^v is $true$ for some $v \in V^k$, or when a fixed point is reached without rendering the goal true. We distinguish these heuristics from the previous ones, as they behave in a different way, produce different results, and have different computational cost.

From a semantic standpoint, inconsistent goals like $(X < 3 \wedge X > 5)$ get infinite h^*_{max} and h^*_{FF} values, while the COUNTERS goal $\bigwedge_{i=0}^{n-1}(X_i < X_{i+1})$ results in optimal h_{max} and h_{FF} values, as the goal becomes satisfiable only at layer P_n. The bad news is that the new heuristics h^*_{max} and h^*_{FF} are *intractable*. Indeed, it is possible to reduce any SAT problem T into a planning problem P such that T is satisfiable iff $h^*_{max}(s_0) \leq 1$, where s_0 is the initial state of P. For the mapping, we just need boolean state variables X_i initially set to $false$ along with actions a_i that can make each variable X_i true. The goal G of P is the CNF formula T with the literals p_i and $\neg p_i$ replaced by the atoms $X_i = true$ and $X_i = false$ respectively. In general, the computation of the heuristics is exponential in the number of state variables of the problem, although this bound can be made tighter. In particular, if fluent symbols are not nested the bound is exponential in the number of state variables involved in any one action. The relaxed planning graph construction that follows the *value-accumulating semantics*, on the other hand, is polynomial, provided that the denotation of fixed function and relational symbols is represented extensionally.

Approximate Constrained Graph

We look now at methods for approximating the constrained relaxed planning graph (CRPG) in polynomial time, in order to derive heuristics that are more informed than those resulting from the unconstrained RPG but remain computationally tractable. For this, we impose some *restrictions* on the fragment of FSTRIPS that we will consider. We assume that *(a)* action preconditions, conditions, and goals are *conjunction of atoms*, rather than arbitrary formulas, and *(b) fluent symbols do not appear nested*. These restrictions still leave ample room for modeling, but allow mapping the bottleneck computation in the construction of the CRPG into a standard *constraint satisfaction problem* (CSP) (Dechter 2003; Rossi, Van Beek, and Walsh 2006). Although solving this CSP is NP-complete, we can take advantage of tractable but incomplete local consistency algorithms to prune the possible values of state variables. Without loss of generality, we will also assume that in all terms $f(t)$ where f is a fluent symbol, t is a tuple of *constant symbols*. Indeed, if that is not the case, then t must be a tuple of fixed compound terms t_i, which can be replaced at preprocessing by constant symbols c_i such that $t_i^* = c_i^*$. The result is that each occurrence of a term $f(t)$ stands for a particular state variable.

The intractability of the constrained RPG follows from checking whether there is an interpretation v in layer P_k that (1) makes the goal G true, or (2) supports the value x of a state variable X through a conditional effect $C \rightarrow f(t) := w$ of an action a. Under the assumptions above, however, for $X = f(t)$, task 2 reduces to checking the truth of the formula $Pre(a) \wedge C \wedge w = x$, so that both tasks 1 and 2 reduce to *checking whether there is an interpretation v that satisfies a conjunction of atoms.* Since the set of possible interpretations is determined by the sets of possible values X^k of each state variable X, the operation boils down to solving a CSP where the *variables* are the state variables X, the *domain* $D(X)$ of the variables X is X^k, and the *constraints* are given by the atoms in the conjunction.

The CSP that represents task 2 above, namely, the consistency test of the formula $Pre(a) \wedge C \wedge w = x$ for each action a and conditional effect $C \rightarrow f(t) := w$, usually involves a bounded and small set of state variables, and can thus be fully solved. On the other hand, the CSP that represents task 1 involves all the state variables that appear in the goal G, and can be solved approximately by using various forms of *local consistency*. In other words, the approximation in the construction of the CRPG applies only to checking whether the goal G is satisfiable in a propositional layer P_k, and relies on the notions of arc and node consistency.

Node consistency prunes the domain $D(X) = X^k$ of each state variable X by going through all the constraints (atoms) in the (goal) CSP that involve only the variable X and removing the values $x \in D(X)$ that do not satisfy all these unary constraints. *Arc consistency*, on the other hand, prunes the domain $D(X)$ of each state variable X by going through all the constraints in the CSP that involve X and another variable Y. If for any of these binary constraints and some $x \in D(X)$ there is no value $y \in D(Y)$ satisfying the constraint, then x is pruned from $D(X)$. The process iterates over all variables and constraints until a fixed point is reached where no further pruning is possible (Mackworth 1977). The goal G is *approximated as being satisfiable* if no state variable gets an empty domain. The h^c_{max} heuristic is defined by the index k of the first layer P_k where the goal is satisfiable, and it is a polynomial approximation of the intractable h^*_{max} heuristic. It is easy to see that $0 \leq h_{max} \leq h^c_{max} \leq h^*_{max} \leq h^*$, where h^* is the optimal heuristic. The pruning resulting from the goal CSP is used also in the *plan extraction procedure* that underlies the computation of the h^c_{FF} heuristic, where pruned values of goal variables are excluded. As an example, if the goal is the atom $X = Y$ where X and Y are state variables with domains $D(X) = \{e, d\}$ and $D(Y) = \{d, f\}$, arc consistency will prune the value e from $D(X)$ and f from $D(Y)$, so that plan extraction will backchain from atoms $X = d$ and $Y = d$.

Arc consistency applies only to binary constraints, *i.e.* to atoms involving two state variables. For atoms involving more variables, we use local consistency algorithms that depend on the constraint type, as is usually done for *global constraints* (Rossi, Van Beek, and Walsh 2006). For example, the quadratic number of binary constraints $X_i \neq X_j$ required to ensure that n variables X_i all have different values can be conveniently encoded with a single global $alldiff(X_1, \ldots, X_n)$ constraint, which additionally can prune the variable domains much more than the binary constraints. Indeed, arc consistency over the binary constraints finds no inconsistency when the domains are $D(X_i) = \{0, 1\}$ for all i, yet it is known that such constraints cannot be jointly satisfied if $|\cup_{i=1}^n D(X_i)| < n$. As an illustration, the goal of stacking all blocks in a single tower in BLOCKSWORLD, regardless of their rela-

Domain	#I	#C	Coverage		Plan length			Node expansions			Time (s.)		
			FF	FS0	FF	FS0	R	FF	FS0	R	FF	FS0	R
COUNT-0	13	11	**13**	11	770.09	**270.09**	2.51	770.09	**270.09**	2.51	**33.98**	318.42	0.13
COUNT-I	13	7	8	**9**	946.14	**204.43**	4.99	946.14	**204.43**	4.99	**34.16**	272.11	0.65
COUNT-R	39	17	17	**28**	499.00	**87.35**	4.10	499.00	**88.76**	4.04	154.50	**25.68**	3.39
GROUP.	72	42	48	**55**	424.24	**43.86**	9.61	681.24	**104.79**	12.28	354.45	**83.12**	41.57
GARD.	51	20	20	**33**	366.85	**86.55**	4.10	2635.95	**456.45**	14.28	205.89	**7.84**	39.68
PUSH.	17	5	5	**8**	65.40	**34.20**	1.43	404.60	**64.80**	3.18	**0.07**	3.98	0.01
PUSH-R	81	34	**53**	34	121.29	**59.38**	1.67	2964.88	**624.82**	7.23	**3.38**	214.33	0.03

Table 1: *Summary of results* for FF and FS0 using a greedy best-first search with heuristics h_{FF} and h^c_{FF} (FF's EHC disabled). #I denotes total number of instances and #C number of instances solved by both planners. Length, node expansion and time figures are averages over instances solved by both planners; R (for *ratio*) is the average of the per-instance FF / FS0 ratios. Best-of-class figures are shown in bold. LAMA and Metric-FF results are discussed in the text.

tive positions, can be compactly encoded with the global constraint $\textit{alldiff}(loc(b_1), \ldots, loc(b_n))$. To accommodate global constraints in our planner, it is necessary to implement a polynomial-time pruning algorithm for the specific constraint type. For the experiments below, we use node and arc consistency for unary and binary constraints, respectively, and bound consistency for the global *alldiff* and sum constraints (van Hoeve and Katriel 2006).

Experimental Results

In order to evaluate the empirical impact of the presented ideas, we have implemented a prototype planner and tested it on a number of problems encoded in FSTRIPS. We describe next the planner and the results. Both the planner and the benchmark problems are available on www.bitbucket.org/gfrances/pubs/wiki/icaps2015.

The FS0 Functional STRIPS Planner

The FS0 planner deals with a fragment of FSTRIPS featuring multivalued state variables, and additionally allows fixed symbols to be intensionally defined by external procedures, as in the semantic attachments paradigm (Dornhege et al. 2009). These include certain global constraints that can be used as fixed predicates. Currently only *alldiff* and sum are supported off-the-shelf, but any arbitrary constraint can be used as long as an implementation of a suitable local-consistency pruning algorithm is provided. FS0 currently employs a simple greedy best-first search (GBFS) strategy guided by either the h^c_{FF} or the $h^c_{\max}$ heuristics; the discussion that follows is restricted to results obtained with h^c_{FF}.

Experiments

We have run our FS0 planner on a number of domains that we describe next. We compare the results to those obtained by some standard planners on equivalent PDDL models, up to language limitations. In order to understand the differences in accuracy and computational cost of the h_{FF} and h^c_{FF} heuristics, we have run the FF (Hoffmann and Nebel 2001) and Metric-FF (Hoffmann 2003) planners (the latter on encodings with numeric fluents, if suitable), using the same greedy best-first search strategy (*i.e.* $f(n) = h(n)$ and enforced hill-climbing disabled). To complete the picture, we have also run the state-of-the-art Fast-Downward planner (LAMA-2011 configuration), that uses different search algorithms and exploits additional heuristic information from helpful actions and landmarks (Helmert 2006; Richter and Westphal 2010). Metric-FF and LAMA results are not shown in the tables but discussed on the next subsection — in the case of Metric-FF, because of the large gap in coverage; in the case of LAMA, because the different heuristics and search algorithms do not allow a direct comparison. All planners are run a maximum of 30 minutes on an AMD Opteron 6300@2.4Ghz, and are allowed a maximum of 8GB of memory. Table 1 shows summary statistics for all domains, whereas Table 2 shows detailed results for selected instances from each of the domains.

Counters Recall that in COUNTERS we have integer variables $X_1, \ldots, X_n$ and want to reach the inequalities $X_i < X_{i+1}$ by applying actions that increase or decrease by one the value of a single variable. We consider three variations of the problem that differ in the way in which variables are initialized: (1) all variables initialized to zero (COUNTERS-0), (2) all variables initialized to random values in the $[0, 2n]$ range (COUNTERS-RND), and (3) all variables initialized to decreasing values from the $[0, 2n]$ range (COUNTERS-INV).

For both FF and FS0, the number of expanded nodes agrees with plan length in all cases, meaning that plans are found greedily. However, FS0 plans are consistently shorter (incidentally, the relaxed plans computed with the approximate CRPG are valid plans for the non-relaxed problem, which is not the case for the RPG). FF performs quite well in COUNTERS-0, where the improved heuristic accuracy offered by FS0 does not compensate the increased running times. This is no longer true if the values of consecutive variables are decreasing. Both in COUNTERS-RND and COUNTERS-INV, FF shows a significantly poorer performance, solving only instances up to 20 and 12 variables, respectively. Again, this stems from the fact that each of the $X_i < X_j$ inequalities is conceived as *independent* (since they are actually encoded in apparently independent propositional atoms $less(i, i + 1)$), which frequently guides the search towards heuristic plateaus. This is not the case for the FS0 planner, which in spite of a much smaller number of expanded nodes per second, has a better coverage on these two variations of the problem, finding much shorter plans and expanding a consistently smaller amount of nodes.

Instance	Plan length		Node expansions		Time (s.)	
	FF	FS0	FF	FS0	FF	FS0
COUNTERS-0 ($n = 8$)	68	**28**	68	**28**	**0.03**	0.14
COUNTERS-0 ($n = 20$)	530	**190**	530	**190**	**2.46**	25.13
COUNTERS-0 ($n = 40$)	2260	**780**	2260	**780**	**197.65**	1796.9
COUNTERS-INV ($n = 8$)	54	**48**	54	**48**	**0.06**	0.36
COUNTERS-INV ($n = 20$)	416	**300**	416	**300**	**8.62**	99.99
COUNTERS-INV ($n = 44$)	**2148**	-	**2148**	-	**1073.46**	-
COUNTERS-RND ($n = 8$)	30	**26**	30	**26**	**0.05**	0.1
COUNTERS-RND ($n = 20$)	2250	**227**	2250	**227**	333.41	**37.1**
COUNTERS-RND ($n = 36$)	-	**680**	-	**680**	-	**1422.83**
GROUPING ($s = 5, b = 10, c = 2$)	287	**23**	391	**23**	3.34	**0.57**
GROUPING ($s = 7, b = 20, c = 5$)	215	**37**	259	**105**	**9.75**	26.39
GROUPING ($s = 9, b = 30, c = 7$)	638	**98**	752	**98**	1346.79	**169.78**
GARDENING ($s = 5, k = 4$)	121	**30**	144	**34**	3.47	**0.12**
GARDENING ($s = 10, k = 10$)	1156	**155**	8827	**244**	1058.55	**7.54**
GARDENING ($s = 15, k = 22$)	-	**360**	-	**1580**	-	**242.45**
PUSHING ($s = 7, k = 4$)	51	**49**	**85**	102	**0.02**	4.27
PUSHING ($s = 10, k = 7$)	-	**189**	-	**576**	-	**160.52**
PUSHING-RND ($s = 7, k = 4$)	**36**	38	**75**	105	**0.02**	3.16
PUSHING-RND ($s = 10, k = 8$)	423	**113**	31018	**4867**	**59.92**	1024.59

Table 2: *Details on selected instances* for the FF and FS0 planners. A dash indicates the solver timed out before finding a plan, and best-of-class numbers are shown in bold typeface. Particular instance parameters are described on the text.

Grouping In the GROUPING domain, some blocks of different colors are scattered on a grid, and we want to group them so that two blocks are in the same cell iff they have the same color. In standard PDDL, there is no compact manner of modeling a goal atom such as $loc(b_1) = loc(b_2)$. For this reason, we have devised an alternative formulation with two additional actions used, respectively, to (1) *tag* any cell as the *destination cell* for all blocks of a certain color, and (2) *secure* a block in its destination cell. We generate random instances with increasing grid size $s \times s$, $s \in \{5, 7, 9\}$, number of blocks $b \in \{5, 10, 15, 20, 30, 35, 40\}$ and number of colors c between 2 and 10, where blocks are assigned random colors and initial locations. The coverage of FS0 is slightly higher, and the h^c_{FF} heuristic used by FS0 proves to be much more informed than the unconstrained version used by FF, resulting on average on 12 times less node expansions and plans around 10 times shorter. This is likely because the delete-free relaxation of the problem allows all unpainted cells to be painted of all colors in the first layer of the RPG, thus producing poor heuristic guidance. FS0 does not incur on this type of distortion, as it understands that goal atoms such as $loc(b_1) = loc(b_2)$ and $loc(b_2) \neq loc(b_3)$ are constrained *to be satisfied at the same time*. Comparing both planners, the increased heuristic accuracy largely compensates in terms of search time the cost of the polynomial approximate solution of the CSP based on local consistency.

Gardening We now illustrate an additional way in which constraints can greatly improve the accuracy of the heuristic estimates. In the GARDENING domain, an agent in a grid needs to water several plants with a certain amount of water that is loaded from a tap and poured into the plants unit by unit. It is known that standard delete-free heuristics are misleading in this type of planning-with-resources environments (Coles et al. 2008), since they fail to account for the fact that the agent needs to load water *repeatedly*: in a delete-free world, one unit of water is enough to water all plants. As a consequence, the plans computed following delete-free heuristics tend to have the agent going back and forth to the water tap, loading each time a single unit of water. FS0, however, is actually able to accommodate and use a *flow constraint* equating the total amount of water obtained from the tap with the total amount of water poured into the plants. This only requires state variables $poured(p_1), \dots, poured(p_n)$, and $total$, plus a goal sum constraint $poured(p_1) + \cdots + poured(p_n) = total$.

We generate random instances with grid size $s \times s$, $s \in [4, 20]$, having one single water tap and $k = max(4, \lfloor s^2/10 \rfloor)$ plants to be watered, each with a random amount of water units ranging from 1 to 10. The results indeed show that FS0 significantly and systematically outperforms FF in all aspects, offering higher coverage and finding plans 4 times shorter, almost 40 times faster, on average.

Pushing Stones Finally, in the PUSHING domain, an agent moves around an obstacle-free grid, pushing k stones into k fixed different goal cells. A noticeable inconvenient of delete-free heuristics in this type of domain is that relaxed plans tend to push all stones to the closer goal cell, even if this means that several stones end up on the same cell. FSTRIPS allows us to express the (implicit) fact that stones must be placed in different cells through the use of a global constraint $alldiff(loc(s_1), \dots, loc(s_k))$, which FS0 exploits to infer more informative heuristic values. This stands in contrast to other heuristic planners, which might be able to indirectly model such a constraint but not to leverage it to improve the heuristic and the search.

To specifically test the potential impact of this constraint, we model a variation of the problem in which all stones concentrate near a single goal cell, with the rest of goal cells located on the other side of the grid (pushing). We also test another variation where agent, stones and goal cells are assigned random initial positions (pushing-rnd). For the first type of instances, FS0 is indeed able to heuristically exploit the *alldiff* constraint and scale up better, having an overall larger coverage, and finding significantly shorter plans. In random instances, on the other hand, FF offers slightly better coverage and a notably smaller runtime, but in terms of plan length and number of expanded nodes, FS0 still outperforms FF by a large margin.

Overview & Other Planners

We now briefly discuss the performance of the LAMA and Metric-FF planners on the same set of problems. In the COUNTERS domain family, LAMA offers a somewhat uneven performance, solving 27/33 instances of the random variation, but only 5/11 of the other two variations. Average plan length is in the three cases significantly better than FF, but at least a 30% worse than FS0, and while total runtimes decidedly dominate the two GBFS planners in the rnd and inv version, they are much worse in COUNTERS-0. For Metric-FF, the PDDL 2.1 encoding that we use is identical to the FSTRIPS encoding, save minor syntactic differences. Metric-FF solves 8/11 instances in COUNTERS-0, but for the other variations only solves a couple of instances. Given that the model is the same, this is strong evidence that at least in some cases it pays off to properly account for the constraints among state variables involved in several atoms.

In the GROUPING domain, LAMA has perfect coverage and is much faster than the two GBFS-based planners, but again the FS0 plans are significantly shorter. In the GARDENING domain, LAMA performs notably worse than FS0 in all aspects, solving 26/51 instances and finding plans that are on average about 4 times longer, in 13 times the amount of time. In the Metric-FF model, we have added the same flow constraint as an additional goal conjunct $total\_retrieved = total\_poured$ but this does not help the search: Metric-FF is not able to solve any of the instances, giving empirical support to the idea that even if we place additional constraints on the goal, these cannot be adequately exploited by the unconstrained h_{FF} heuristic, *precisely because* it does not take into account the constraints induced on state variables appearing in more than one goal atom. Finally, in the PUSHING domain LAMA outperforms in all aspects the two GBFS planners in the random variation, but shows a much poorer performance on the other variation, where the *alldiff* constraint proves its heuristic usefulness.

Overall, we have shown that in our test domains the use of the constrained h^c_{FF} heuristic consistently results in significantly shorter plans (between approximately 1.5 to 9.5 times shorter, depending on the domain) compared to its unconstrained counterpart. The heuristic assessments are also more accurate in all domains, resulting in a 2.5- to 14-fold decrease in the number of expanded nodes, depending on the domain. In some cases, the increased heuristic accuracy demands significantly larger computation time, although in terms of final coverage this overhead tends to be compensated; in other cases the increased accuracy itself already allows smaller average runtimes.

Discussion

The goal in the PUSHING domains can be described in FSTRIPS as placing each stone in a goal cell while ensuring that all stones are in different cells. In propositional planning, the goal is encoded differently, and the *alldiff* constraint is implicit, hence redundant. Indeed, the *flow constraint* of the FSTRIPS GARDENING domain is also redundant. The fact that the performance of FS0 is improved by adding redundant constraints reminds of CSP and SAT solvers, whose performance can also be improved by explicating implicit constraints. This distinguishes FS0 from the existing heuristic search planners that we are aware of, that *either make no room for any type of explicit constraints or cannot use them in the computation of the heuristic.* This capability is thus not a "bug" that makes the comparisons unfair but a "feature". Indeed, recent propositional planners illustrate the benefits of *recovering* multivalued (Helmert 2009) and flow constraints (Bonet and van den Briel 2014) that the modeler was forced to hide. In this paper, we advocate a different approach: to make room for these and other types of constraints at the language level, and to account for such constraints in the computation of the heuristics.

Summary

To sum up, we have considered the intertwined problems of modeling and computation in classical planning, and found that expressiveness and efficiency are not necessarily in conflict. Indeed, computationally it may pay to use richer languages when state variables appear more than once in action preconditions and goals, as long as the resulting constraints are accounted for in the computation of the heuristic. In our formulation, heuristics are obtained from a logical analysis where sets of values X^k that are possible for a state variable X in the P_k layer of the relaxed planning graph are thought of as encoding an exponential set of possible logical interpretations. Since these heuristics are more informed but intractable, we show how they can be effectively approximated with local consistency techniques. Our FS0 planner, implementing these ideas, supports a substantial fragment of the FSTRIPS language and further extends its expressiveness by allowing the denotation of fixed symbols to be defined by external procedures and by accommodating a limited library of global constraints — yet these features are not add-ons, but a result of our logical analysis. We have empirically shown the computational value of these ideas on a number of meaningful examples. In the future, we want to optimize the planner implementation and to make room for *state constraints*, *i.e.* invariants that hold not just in goal states but in all states.

Acknowledgments This work is partially funded by the Simulpast Project (CSD2010-00034), CONSOLIDER-INGENIO2010 program of the Spanish MICINN. H. Geffner thanks Maria Fox, Derek Long, and the members of the planning group at King's College for conversations on topics related to this paper. The FS0 planner is partially built upon the Lightweight Automated Planning Toolkit (www.lapkt.org) and the Fast Downward PDDL parser (www.fast-downward.org).

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Bonet, B., and van den Briel, M. 2014. Flow-based heuristics for optimal planning: Landmarks and merges. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. A hybrid relaxed planning graph-LP heuristic for numeric planning domains. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 52–59.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic attachments for domain-independent planning systems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 114–121.

Geffner, H. 2000. Functional STRIPS: A more flexible language for planning and problem solving. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer. 187–205.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning modulo theories: Extending the planning paradigm. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*.

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(5):503–535.

Hernádvölgyi, I. T., and Holte, R. C. 1999. PSVN: A vector representation for production systems. Technical Report TR-99-04, Department of Computer Science, University of Ottawa.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2003. The metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Ivankovic, F.; Haslum, P.; Thiébaux, S.; Shivashankar, V.; and Nau, D. S. 2014. Optimal planning with global numerical state constraints. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial intelligence* 8(1):99–118.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39(1):127–177.

Rintanen, J., and Jungholt, H. 1999. Numeric state variables in constraint-based planning. In *Proceedings of the European Conference on Planning (ECP-99)*, 109–121.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of constraint programming*. Elsevier.

van Hoeve, W.-J., and Katriel, I. 2006. Global constraints. *Handbook of constraint programming* 169–208.