∃-STRIPS: Existential Quantification in Planning and Constraint Satisfaction

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

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

Abstract

Existentially quantified variables in goals and action preconditions are part of the standard PDDL planning language, yet few planners support them, while those that do compile them away at an exponential cost. In this work, we argue that existential variables are an essential feature for representing and reasoning with constraints in planning, and that it is harmful to compile them away or avoid them altogether, since this hides part of the problem structure that can be exploited computationally. We show how to do this by formulating an extension of the standard delete-relaxation heuristics that handles existential variables. While this extension is simple, the consequences for both modeling and computation are important. Furthermore, by allowing existential variables in STRIPS and treating them properly, CSPs can be represented and solved in a direct manner as action-less, fluent-less STRIPS planning problems, something important for problems involving restrictions. In addition, functional fluents in Functional STRIPS can be compiled away with no effect on the structure and informativeness of the resulting heuristic. Experiments are reported comparing our native ∃-STRIPS planner with state-of-the-art STRIPS planners over compiled and propositional encodings, and with a Functional STRIPS planner.

1 Introduction

Modeling and computation are essential components for using planners and many other automated tools. On the one hand, we want planners to scale up gracefully to large problems, even if the task is computationally intractable in the general case [Bylander, 1994]; on the other, we want users to be able to encode problems naturally and directly without having to understand how the different planners work. Indeed, the need to "hack" encodings to make planners work is one of the factors that hinder the use of planners by non-experts. Since modeling and computation are heavily coupled, the problem cannot be entirely solved, yet this doesn't mean that progress on this front is not possible and necessary. In this work, we focus on the use of existentially quantified variables in goals and action preconditions in STRIPS and the way in which they can have a positive impact on both modeling and computation. Existential variables provide indeed a very simple and convenient modeling feature in problems where the goal is not a fully-specified single state but leaves room for choices. The goal in a Sokoban-like problem, for instance, may require placing a set of packages in different target locations without explicitly saying in which target location to place each package; or in the blocks-world, the goal may be to reach a state where a blue block is on a red block, without specifying the exact block identities. While there are other mechanisms to model such open choices, existential variables make them explicit, which is useful for reasoning about them. Existentially quantified variables in preconditions and goals are part of the standard PDDL language [McDermott *et al.*, 1998] but few planners support them, and those that do compile them away at a cost that is exponential in their number, which is not a good option from a computational standpoint. The usual alternative is to avoid existential variables altogether, which as we will show is not good either, since it leads to clumsy encodings that hide structure and result in less informed heuristics than the compilation-based approach, not to mention the fact that coming up with such alternative encodings is not always trivial. We illustrate these various points below.

Consider the problem of placing a blue block on top of a red block, and this block on top of a blue block. The problem can be easily encoded in PDDL with a goal formula

$$\exists z_1, z_2, z_3.\, on(z_1, z_2) \wedge on(z_2, z_3) \wedge bl(z_1) \wedge red(z_2) \wedge bl(z_3).$$

Modern classical planners like FF, FD, and LAMA [Hoffmann and Nebel, 2001; Helmert, 2006; Richter and Westphal, 2010] accept such existentially quantified formulas, but they precompile the quantification away, producing in this case a goal in disjunctive normal form (DNF) with terms

$$on(a, b) \wedge on(b, c) \wedge bl(a) \wedge red(b) \wedge bl(c)$$

for all possible combinations of blocks a, b and c. This DNF goal is mapped in turn into an atomic dummy goal that can be achieved with extra actions whose preconditions are those terms [Gazen and Knoblock, 1997]. The immediate problem with this compilation is that it blows up. The number of terms in the compilation is exponential in the number of variables, and thus it cannot scale up to more than a few quantified variables. This is a problem because it is not always easy to find alternative, equivalent, and compact propositional encodings. Do and Tran [2013], for instance, present one such encoding for a similar problem, but the encoding assumes that the goal is complete and describes a single state. In such a case, one can actually forget about the blocks and describe the goal by stating the desired pattern of "cell colors" as $blue(3,1)$, $red(2,1)$, $blue(1,1)$, etc., which can be encoded compactly in STRIPS without existential variables, but represents a different, less general problem.

In other cases, it is simpler to come up with equivalent and compact encodings that avoid existential variables. The goal of placing n packages p_i into n different target locations l_k, for instance, can be expressed with existential quantification and equality as

$$\exists z_1, \ldots, z_n. \left(\bigwedge_{j \neq i} z_i \neq z_j \wedge \bigwedge_i target(z_i) \wedge at(p_i, z_i) \right).$$

The quantification and the compilation can be avoided here by using a $clear$ predicate over target locations that becomes a precondition of the action of dropping a package in such locations. The resulting encoding is *equivalent* to the one obtained by compiling the previous formula in the sense that both encodings result in the same set of plans (dummy actions aside), and it has the advantage that its size does not grow exponentially with n. This compact encoding, however, is not only less clean and direct than the one with quantification, but it is also certainly *not equivalent* from a *computational* point of view. To see this, consider the standard delete-relaxation heuristics used in classical planning [McDermott, 1999; Bonet and Geffner, 2001] and a situation where none of the packages is at a target location. In the compilation, the heuristic would estimate the cost of taking each package to a *different* target location, while in the alternative compact encoding with no variables, the heuristic would estimate something different: namely, the cost of taking each package to its *closest* target location. These estimates will be different when several packages have the same closest target location. This is easy to see for $n = 2$ where the compilation results in a DNF goal with terms $at(p_1, l_1) \wedge at(p_2, l_2)$ and $at(p_1, l_2) \wedge at(p_2, l_1)$, but without the terms $at(p_1, l_1) \wedge at(p_2, l_1)$ or $at(p_1, l_2) \wedge at(p_2, l_2)$ where both packages end up at the same location. The encoding with $clear$ predicates does not account for this difference, as both target locations will remain $clear$ in the delete relaxation as long as they are $clear$ in the state on which the heuristic is evaluated.

The last example illustrates that while compiling away the variables in the goal has an exponential cost, *there are also hidden computational costs in the avoidance of existential variables* to prevent the exponential blow-up. This point can be made more general and compelling once that we notice that existential variables in the goal allow us to encode naturally and directly constraint satisfaction problems [Dechter, 2003]. Indeed, a binary CSP with variables $X_1, \ldots, X_n$ with domain D and constraints $R_{i,j}$ among all pairs of variables X_i and X_j can be naturally and directly encoded as a STRIPS problem with no actions at all, initial facts $R_{i,j}(a, b)$ for each value pair $\langle a, b \rangle$ in the domain allowed by the constraint $R_{i,j}$, and an existentially quantified goal:

$$\exists z_1, \ldots, z_n. \bigwedge_{i,j} R_{i,j}(z_i, z_j).$$

The compilation-based approach would convert such a quantified goal into a grounded DNF goal whose terms are all possible joint valuations of the CSP. Definitely, this is not an effective way of solving the CSP, yet the delete-relaxation heuristic for the compilation is *exact*: it will yield a heuristic value of 0 for the initial state iff the CSP is actually solvable. Alternative STRIPS encodings of the CSP that do not involve variables and are compact are certainly possible, yet such compact, polynomial encodings cannot yield a (polynomial) delete-relaxation heuristic equivalent to the heuristic obtained from the compilation unless $\mathsf{P} = \mathsf{NP}$.

In other words, while dealing with existential variables by compiling them away is exponential, avoiding them altogether results in poorer heuristic estimates in general. In this paper we show that there is a third option which is the *explicit handling of existentially quantified variables during the computation of the delete-relaxation heuristic.* The resulting method avoids the exponential blow-up in the compilation, and yet it delivers a heuristic that is *equivalent to the one that would be delivered by the delete-relaxation in the grounded, compiled problem.* For this, the construction of the relaxed planning graph *must take into account the first-order structure of the atoms.* Actually, the evaluation of any action precondition or goal formula involving existentially quantified variables maps into a CSP that must be solved during the computation of the heuristic. The computation of the heuristic is thus intractable in the worst case, but as we will show, it can be informed and cost-effective in practice. The consideration of the first-order structure of formulas in the heuristic computation has a close relation with the Functional STRIPS (FSTRIPS) heuristic presented in [Francès and Geffner, 2015], which is also intractable in the worst case. Even more interestingly, the proper handling of existential variables in STRIPS gives a way for compiling pure FSTRIPS into $\exists$-STRIPS as in [Haslum, 2008], but with no effect on the structure and informativeness of the resulting heuristic.

The rest of the paper is organized as follows. We present the $\exists$-STRIPS language that extends STRIPS with existentially quantified variables, and formulate a version of the delete-relaxation heuristic for such a language. We then evaluate the resulting native $\exists$-STRIPS planner, which we call `ES`, by comparing it with state-of-the-art STRIPS planners run on existentially-quantified encodings and on alternative, compact propositional encodings that avoid the compilation, and also with an extension of the FSTRIPS planner `FS0`.

2 $\exists$-STRIPS Language

We briefly review the language and semantics of STRIPS with existential quantification in action preconditions and goals. The classical planning model $\mathcal{S} = \langle S, s_0, S_G, A, f \rangle$ is made up of a finite set of states S, an initial state s_0, a set of goal states S_G, and actions $a \in A(s)$ that deterministically map one state s into another $s' = f(a, s)$, where $A(s)$ is the set of actions applicable in s. A solution or plan is a sequence of actions $a_0, \ldots, a_m$ that generates a state sequence

$s_0, s_1, \ldots, s_{m+1}$ such that $a_i \in A(s_i)$, $s_{i+1} = f(a_i, s_i)$, and $s_{m+1} \in S_G$.

A classical planning problem P defines a classical model in compact form. In STRIPS, P is a tuple $\langle F, I, O, G\rangle$, where F is the set of atoms, I is the set of atoms characterizing the initial situation, O is the set of actions, and G is the set of goal atoms. The actions a in STRIPS are given by three sets of atoms: $Pre(a)$, $Add(a)$, and $Del(a)$.

A STRIPS problem $P = \langle F, I, O, G\rangle$ defines a classical model $\mathcal{S}(P) = \langle S, s_0, S_G, A, f\rangle$ where the states in S are subsets of F that represent truth valuations where an atom p is true in s iff $p \in s$. The other elements in $\mathcal{S}(P)$ are $s_0 = I$, S_G that is given by the states that satisfy G, $A(s)$ that stands for the actions $a \in O$ whose preconditions $Pre(a)$ are true in s, and $f(a, s)$ that stands for $s' = (s \setminus Del(a)) \cup Add(a)$.

STRIPS is used as a propositional language even if it was originally a first-order language [Fikes and Nilsson, 1971]. Indeed, while in PDDL, STRIPS problems are given in a language featuring action schemas, and predicate and variable symbols, modern planners ground all these schemas using the object names and their types if available. A resulting ground atom like $on(a, b)$ is then treated propositionally, but the addition of *existential quantification* to the language requires that we look into the structure of such atoms.

In $\exists$-STRIPS, a problem is also a tuple $P = \langle F, I, O, G\rangle$, with the same semantics given by the state model $\mathcal{S}(P)$, but with *one difference:* in addition to ground atoms, preconditions and goals may feature e-formulas. An *e-formula* is a first-order formula of the form $\exists x_1, \ldots, x_n. \phi$, where ϕ is a conjunction of atoms $p(t_1, \ldots, t_n)$ with the terms t_i being constant symbols or variable symbols x_i from the quantification prefix. The e-formula $\exists x_1, \ldots, x_n. \phi$ is true in a state s if the variable symbols x_i can be consistently replaced by constant symbols c_i so that the resulting grounded formula ϕ' is true in s. This definition extends naturally when the quantified variables x_i and the constant symbols c_i in the instance have types — in such a case, variables have to be replaced by constants of a compatible type. When x is a tuple of variables, we write the e-formula $\exists x. \phi$ as $\exists x. \phi[x]$ so that the ground conjunction of atoms ϕ' that results from the substitution $x \mapsto c$ in ϕ can be written as $\phi[x \mapsto c]$ or simply $\phi[c]$. For example, for the formula $\exists x_1, x_2, x_3.\ p(x_1, x_2) \wedge p(x_2, x_3)$, if ϕ is $p(x_1, x_2) \wedge p(x_2, x_3)$, $\phi[x_1, x_2, x_3 \mapsto c, d, c]$ denotes the variable-free formula $p(c, d) \wedge p(d, c)$.

As discussed in the introduction, determining if an e-formula holds in a state s is an NP-complete problem. This means that plan verification in $\exists$-STRIPS is NP-complete, and that any heuristic h able to distinguish between goal and non-goal states (i.e., such that $h(s) = 0$ iff s is a goal state) will be intractable. This however does not imply that such heuristics cannot be cost-effective.

3 $\exists$-STRIPS Heuristics

Adapting the relaxed planning graph (RPG) heuristic from STRIPS to $\exists$-STRIPS is simple, even if the resulting heuristic is no longer polynomial. Recall that the RPG heuristic $h(s)$ for STRIPS is given by the number of actions in a plan $\pi^+(s)$ obtained from the delete-free relaxation by following a 2-step procedure [Hoffmann and Nebel, 2001]. In a first forward phase, the procedure incrementally builds a layered graph with propositional layers P_i and action layers A_i. Starting with $i = 0$ and with a proposition layer P_0 that contains the ground facts that hold in s, action a makes it into layer A_i if each precondition atom in $Pre(a)$ is in P_i, and similarly an atom p makes it into layer P_{i+1} if there is an action $a \in A_i$ with p in $Add(a)$. The procedure stops on the first propositional layer P_i such that the goals G are in P_i, or $G \notin P_i$ and $P_i = P_{i+1}$. In addition, an action $a \in A_i$ is deemed a supporter of an atom p if p appears in layer P_{i+1} for the first time and $p \in Add(a)$. The heuristic $h(s)$ is set to ∞ when the graph construction finishes without reaching the goal G, and is otherwise given by the number of different actions in a plan $\pi^+(s)$ for the delete-relaxation obtained backward from the goal. The plan includes a supporter for each goal atom that is not in s, and recursively, a supporter for each precondition of such supporter that is not in s either. The relaxed plan and the heuristic are not unique, as an atom may have more than one support.

Extending the RPG heuristic from STRIPS to $\exists$-STRIPS simply requires to define how to deal with e-formulas in preconditions and goals. While a ground atom p is deemed to be *satisfiable* in a propositional layer P_i when p belongs to P_i, we take an e-formula $\exists x.\phi[x]$ to be *satisfiable* in P_i when there is a substitution $x \mapsto c$ such that all ground atoms in $\phi[c]$ belong to P_i. If, for instance, ϕ is $p(x_1, x_2) \wedge p(x_2, x_3)$, then the formula $\exists x_1, x_2, x_3.\ \phi$ is satisfiable in a layer P_i that contains the atoms $p(c, d)$, $p(c, e)$, and $p(d, c)$, through the substitution $[x_1, x_2, x_3 \mapsto c, d, c]$. On the other hand, the same formula would not be satisfiable in that layer if only the first two atoms were contained in P_i.

Extending the RPG construction to take into account e-formulas is simple to state but potentially complex to compute: the satisfiability test for e-formulas in a state or in a propositional layer P_i amounts to solving a CSP with variables that correspond to the variable symbols in the prefix, domains given by the sets of constant symbols (respecting type-consistency when types are used) and constraints given by the atoms $p(t_1, \ldots, t_n)$ in ϕ such that the the predicate symbols represent the tables that contain the tuples of constants $c_1, \ldots, c_n$ for atoms $p(c_1, \ldots, c_n)$ in layer P_i only. A substitution $x \mapsto c$ satisfies ϕ if c is a solution to this CSP.

For plan extraction, the supporters of an e-formula $\exists x.\phi[x]$ that is satisfied in a layer P_i with substitution $x \mapsto c$ are identified with the supporters of the atoms in $\phi[c]$. Thus the plan extraction procedure that works backward from the goal does not see e-formulas at all, just ground atoms, as in the standard RPG procedure.

We call the relaxed plans and the heuristic that result from this procedure for the $\exists$-STRIPS language $\pi^e_{\text{FF}}(s)$ and $h^e_{\text{FF}}(s)$ respectively. If instead of counting the actions in the relaxed plan we count the number of propositional layers, we get a variant of the h_{max} heuristic, h^e_{max}, that can be shown to be admissible for $\exists$-STRIPS. The heuristic $h^e_{\text{FF}}(s)$ is not admissible, in the same way that h_{FF} is not admissible for STRIPS. The consistency check for e-formulas in each layer P_i is delegated to the standard Gecode CP solver [Gecode Team, 2006].

4 Relation to Compilation and Functional STRIPS

The heuristics h^e formulated above for ∃-STRIPS are equivalent to the corresponding heuristics h for the STRIPS compilation and can be regarded as a lazy version of them. The key difference is that the compilation requires *time and space* exponential in the number of existential variables, while the heuristics for ∃-STRIPS require only exponential *time in the worst case*. The terms of the ground DNF formulas required for compiling away e-formulas $\exists x.\, \phi[x]$ in preconditions and goals correspond indeed to the conjunctions of ground atoms $\phi[c]$ for each of the possible substitutions $x \mapsto c$. The computation of the heuristics however does not apply these possible substitutions all at once, and moreover, it does not explore the space of all possible substitutions exhaustively either. Instead, it uses constraint propagation to search for a substitution that makes the e-formula $\exists x.\, \phi[x]$ satisfiable.

More interestingly, the heuristics proposed for ∃-STRIPS have a strong relation to recently-proposed heuristics for Functional STRIPS, a language that extends STRIPS with (possibly fluent) function symbols [Geffner, 2000] — a feature recently incorporated to the PDDL 3.1 language standard under the name of *object fluents*.[1] Haslum [2008] has shown how pure Functional STRIPS (with no built-in functions) can be compiled into ∃-STRIPS by adding existential variables and replacing functions by relations using the principles of Skolemization in reverse. For example, a grounded atom $f(g(c_1)) = c_2$ is replaced by the e-formula $\exists x_1 . f'(x_1, c_2) \wedge g'(c_1, x_1)$ where the relations $f'(x_1, x)$ and $g'(c_1, x)$ represent $f(x_1) = x$ and $g(c_1) = x$ respectively.

The heuristic h^c_{FF} developed recently for Functional STRIPS [Francès and Geffner, 2015] is aimed at capturing the constraints induced among state variables in preconditions and goals. In the COUNTERS domain, for instance, there is 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 $[1, n]$ interval. Given some initial values, the goal is to achieve the inequalities $X_i < X_{i+1}$, for $1 \leq i < n$. Francès and Geffner show how delete-relaxation heuristics are not informative in this problem, as they all find that the goal is reachable in the first propositional layer P_i that makes each of the goals satisfiable. Yet there is a crucial difference between making each goal satisfiable and making all goals satisfiable *at the same time*. For example, a layer in which each variable X_i can take the values 1 and 2 will make each of the goals satisfiable but will not make the *joint goal* satisfiable. For this, each variable X_i must have the value i, for $i = 1, \ldots, n$. The heuristic developed by Francès and Geffner captures such induced constraints. Interestingly, the proposed heuristics h^e for ∃-STRIPS account for such constraints too. Indeed, the problem can be represented in ∃-STRIPS in terms of atoms $val(i, k)$ to state that variable X_i has value k, and static atoms $less(k, k')$ to express that value k is less than value k'. The goal for the problem can then be encoded as

$$\exists z_1, \ldots, z_n . \bigwedge_{i=1..n} [val(i, z_i)] \wedge [\bigwedge_{i=1..n-1} less(z_i, z_{i+1})] \, .$$

[1]http://ipc.informatik.uni-freiburg.de/PddlExtension

This formula $\exists x.\, \phi[x]$ will be satisfied in a propositional layer P_k only by the substitution that replaces the variable z_i by the value i, because the heuristic accounts for the fact that the z_i variables need to be replaced consistently in all goal subformulas. This is interesting because it means that the heuristic-search planner for FSTRIPS developed by Francès and Geffner can be emulated by compiling FSTRIPS into ∃-STRIPS through Haslum's translation and applying then the new heuristics. Running other planners on such translations would not deliver the same results. It must be said , however, that Haslum's translation captures the core of FSTRIPS only, leaving out the ability to use built-in functions and global constraints, which in certain problems can make a large difference. On the other hand, the heuristic for ∃-STRIPS is simpler to describe and implement than the heuristic for FSTRIPS.

5 Experimental Results

We have developed a preliminary implementation of the CSP-based approach for handling existential quantification on top of the existing Functional STRIPS planner `FS0` [Francès and Geffner, 2015], which already integrates a CSP solver. We call the resulting ∃-STRIPS planner `ES`. The planner uses the h^e_{FF} heuristic to drive a plain greedy best first search. As a side effect of the implementation, we also obtain the ability to deal with existential variables in the underlying Functional STRIPS planner.

STRIPS Setup. We consider four families of problems, each encoded both in ∃-STRIPS and in a propositional STRIPS *manual reformulation* that avoids existential quantifiers while preserving equivalence. On these problems, we compare the performance of our `ES` planner running on the ∃-STRIPS encoding against that of the `Fast-Downward` planner (`FD`) running on both encodings. `FD` is configured with the same search strategy; namely, greedy BFS with the h_{FF} heuristic, a single queue, no EHC, and non-delayed evaluation. All planners run a maximum of 30 minutes on a cluster with AMD Opteron 6300@2.4Ghz nodes, and are allowed a maximum of 8GB of memory. The source code of `ES` plus all problem encodings are available on `gfrances.github.io/pubs`.

Domains. The four families of domains that we test are BW-PATTERN,VERTEX-COLORING, GROUPING and COUNTERS, the last two from [Francès and Geffner, 2015]. **BW-PATTERN**, discussed in the introduction, is a variation of Blocksworld with colored blocks where we want to reach *any* block configuration that displays a given color pattern, such as "a red block on a blue block on a green block", which is modeled with the goal e-formula $\exists\, b_1, b_2, b_3 .\ col(b_1, red) \wedge col(b_2, blue) \wedge col(b_3, green) \wedge on(b_1, b_2) \wedge on(b_2, b_3)$. We test no propositional encoding, as it is not trivial to reformulate this problem without existential variables.

VERTEX-COLORING illustrates how any CSP can be easily embedded in an (action-less, fluent-less) ∃-STRIPS planning problem. A classical k-VERTEX-COLORING problem with n nodes and edges E maps into an ∃-STRIPS problem with k color objects and (typed) goal e-formula $\exists z_1, \ldots, z_n \in$

Domain	N	Coverage			Plan length			Node expansions			Time (s.)		
		E-FF	P-FF	E-ES	E-FF	P-FF	E-ES	E-FF	P-FF	E-ES	E-FF	P-FF	E-ES
COUNT$_0$	12	3	**7**	**7**	-	70.7	70.7	-	542K	70.7	-	52.6	84.9
COUNT$_{RND}$	36	9	**25**	21	21.4	21.6	22.1	22.4	36.6	22.1	0.9	0.0	0.7
GROUPING	48	28	34	**47**	25.8	41.6	26.1	26.8	203K	26.2	2.1	179.5	54.6
BW-PATTERN	30	11	-	**20**	8.4	-	50.9	306.3	-	16.19K	0.6	-	60.9
V-COLORING	80	1	43	**78**	-	84.3	0	-	60.7K	0	-	87.7	0
A-COLORING	88	6	**30**	27	24.5	26.9	29.2	41.2	80.8	65.0	0.0	0.0	0.3

Table 1: ∃-**STRIPS planning**. "E-FF" is FD with FF-like configuration on ∃-STRIPS version; "P-FF" is the same planner on a manual propositional STRIPS reformulation; "E-ES" is our ES planner on the ∃-STRIPS version; *N* is number of instances; length, node expansion and time figures are averages over instances solved by all those planners that solve at least 5 instances.

Domain	N	Coverage			Plan length			Node expansions			Time (s.)		
		P-FF	E-ES	EF-FS	P-FF	E-ES	EF-FS	P-FF	E-ES	EF-FS	P-FF	E-ES	EF-FS
COUNT$_0$	12	7	7	**11**	70.7	70.7	70.7	542K	70.7	70.7	52.6	84.9	5.6
COUNT$_{RND}$	36	25	21	**30**	93.0	86.6	86.2	6.9K	87.2	86.2	15.8	117.9	11.2
GROUPING	48	34	47	**48**	42.4	23.8	24.3	150K	23.8	24.3	146.8	25.9	5.8
BW-PATTERN	30	-	20	**29**	-	45.1	5.7	-	9.28K	6.2	-	38.8	0.2
V-COLORING	80	43	**78**	**78**	84.3	0	0	60.7K	0	0	87.7	0	0
A-COLORING	88	30	27	**38**	71.9	94.9	77.9	353.3	1.82K	609.6	0.1	30.9	4.1

Table 2: ∃-**STRIPS *vs.* Functional STRIPS planning**. "P-FF" is FD with FF-like configuration on a manual propositional STRIPS reformulation; "E-ES" is ES on ∃-STRIPS version; "EF-FS" is the extension FS of FS0 run on FSTRIPS encodings (with existential variables, when necessary). P-FF and E-ES columns are from Table 1. *N* is number of instances; length, node expansion and time figures are averages over instances solved by the three planners.

COLOR $\bigwedge_{(i,j)\in E} z_i \neq z_j$. The initial (and only) state of this planning problem will be detected as a goal iff the graph is k-colorable. We also test **AGENT-COLORING**, a variation of the domain where an agent moves around the graph picking up and dropping paint cans to paint the vertices "manually". We use two sets of instances: first, random instances ($100 \leq n \leq 500$, $10 \leq k \leq 30$) where the graph results from adding edges at random to a uniform spanning tree to reach a certain graph density; second, standard instances from a public compilation from the literature,[2] of which we have pruned those instances reported as not solvable in less than 1 hour by state-of-the-art coloring methods *or* having a chromatic number $\chi > 10$. From the remaining instances, we generate planning problems with a number of colors $k \in \{\chi, \chi+1\}$.

In **GROUPING**, several colored blocks are randomly placed on a grid and need to be moved so that two blocks end up on the same cell iff they have the same color. The original FSTRIPS formulation features a goal formula with an atom $loc(b_i) \bowtie loc(b_j)$ for each pair of blocks b_i, b_j, where $\bowtie$ is $=$ ($\neq$) if the two blocks have the same (different) color. This translates directly to ∃-STRIPS, so that the goal of a problem with red blocks a and b and a blue block c is $\exists\ l, l'.\ l \neq l' \wedge at(a,l) \wedge at(b,l) \wedge at(c,l')$.

Finally, **COUNTERS** is the problem described in Section 4 where we increase or decrease the value of certain integer variables X_i until the inequalities $X_1 < \ldots < X_n$ hold. We use two variations, labeled **COUNT$_0$** and **COUNT$_{RND}$**, where variables are initially set to 0 and to random values, respectively. The ∃-STRIPS encoding is the result of applying Haslum's compilation technique to the original FSTRIPS encoding, save for the substitution of the "<" built-in predicate for a $less$ predicate with extensional denotation.

[2]See https://sites.google.com/site/graphcoloring/.

Results of STRIPS Planners. Table 1 shows the results of the ES planner running on ∃-STRIPS ("E-ES") as well as of Fast-Downward on both the ∃-STRIPS encoding ("E-FF") and the manual propositional reformulation ("P-FF"). We next highlight the main conclusions that can be drawn from these results.

First, *the exponential time and space required to compile away ∃-STRIPS existential variables has a huge impact on domain coverage.* In general, FD tends to either time or memory out during preprocessing. As an example, the planner times/memories out on GROUPING instances with more than 3 colors, and on COUNTERS instances with more than 9 variables. The only domain with acceptable coverage is GROUPING, but this is just because the benchmark set includes many instances with only 2 existential variables.

Second, *the heuristic that can be derived from the propositional reformulations is less informed than the one computed from the compiled ∃-STRIPS encoding.* In GROUPING or in COUNT$_0$, for instance, the average number of node expansions is orders of magnitude higher on the propositional reformulation, because the delete-relaxation does not account for certain constraints that are captured on the ∃-STRIPS version, such as (in GROUPING) the constraint that all blocks of the same color must be on the same position *at the same time*. In spite of this, however, FD has much better performance on these propositional reformulations, because it avoids the exponential time and space required to compile away the existential variables and to process the result of the compilation.

Third, *the constraint-based approach of* `ES` *to handle existential variables avoids the exponential time and space penalty of the compilation approach without making the resulting heuristics less informed*, making it a more effective strategy, and resulting in consistently higher coverages.

Fourth, *when compared to the performance of* `FD` *on the propositional reformulations,* `ES` *produces better coverage on two cases and worse coverage on two other cases*, excluding the BW-PATTERN domain that is not simple to encode propositionally. The average number of expanded nodes is in general much lower for `ES`, up to one or two orders of magnitude in some cases, but at the same time node expansion rate is also around between one and two orders of magnitude faster in `FD`. Upon inspection, `ES` suffers a significant overhead caused by the lack of multivalued variables in ∃-STRIPS, which results in CSPs with too many Boolean variables that could well be replaced by fewer multivalued variables. This shortcoming does not appear in Functional STRIPS, which in addition benefits from built-in functions and constraints.

Comparison with Functional STRIPS. As a side effect of our implementation of `ES` on top of the `FS0` planner, and following the previous observation, we have also developed an extension of `FS0`, which we call `FS`, that is able to handle FSTRIPS with existential variables by following the same ideas that we have described for ∃-STRIPS. In addition, and thanks to the compilation technique described in Section 4, this new `FS` planner can handle nested functional terms, which the original FSTRIPS planner `FS0` cannot. Table 2 compares `FD` on the propositional STRIPS reformulations ("`P-FF`", which is the best option for `Fast-Downward` according to Table 1, leaving aside the reformulation effort) with `ES` on ∃-STRIPS encodings ("`E-ES`") and with `FS` on equivalent FSTRIPS encodings that exploit both functions and existential variables ("`EF-FS`"). In some cases (COUNTERS, GROUPING), these are pure FSTRIPS encodings that do not need existential quantification, but in the remaining domains (BW-PATTERN, VERTEX-COLORING, AGENT-COLORING) existential quantification is necessary.

When compared to `ES` on ∃-STRIPS encodings, `FS` on the FSTRIPS encodings shows higher node expansion rates and even a more informed heuristic in BW-PATTERN. This is because (1) FSTRIPS encodings usually need fewer grounded actions, (2) the CSPs used during the heuristic computation have significantly fewer variables (e.g. quadratically so in GROUPING) (3) they benefit from custom constraint propagators for FSTRIPS built-in symbols, such as the "<" symbol in COUNTERS, and (4) certain support extraction tie-breaking mechanisms are easily implemented for FSTRIPS, but have not yet been implemented in the current version of the `ES` planner, which might be harming the heuristic accuracy in BW-PATTERN.

Overall, the novel `FS` planner working on an extension of Functional STRIPS with existential quantification definitely pays off, offering a coverage which consistently dominates the other two planners, and plan length, average number of expanded nodes and overall running time figures which are consistently better. In particular, the extension of Functional STRIPS with existential quantification yields better results than the manual propositional STRIPS reformulations *in all domains*.

6 Discussion

We have presented a novel view of existential quantification in STRIPS that relates planning to constraint satisfaction and relational databases [Gottlob *et al.*, 1999]. A goal that involves existential variables is indeed like a query; what makes it particular in the context of planning is that we are not simply evaluating the goal "query" in a static database, but can apply actions that modify the database in order to make the query satisfiable, and hence the goal true.

We have also argued that the use of existential quantification for leaving some choices open in goal and preconditions allows for simpler and more concise models, while giving planners the chance to reason about those choices. Instead of compiling existential variables away with an exponential technique or avoiding them altogether, we have shown how delete-relaxation heuristics can be extended to deal with existential variables by using constraint satisfaction techniques, resulting in heuristics that are more informed than those that can be obtained from alternative propositional encodings.

This view of existential variables in STRIPS has resulted in a new ∃-STRIPS planner implemented on top of the existing Functional STRIPS `FS0` planner, *plus* a new Functional STRIPS planner that completes support and extends the original language as defined in [Geffner, 2000]. We have evaluated these planners on four families of domains encoded in STRIPS and FSTRIPS with existential variables, and when feasible and simple enough, without them. The fact that the best computational results have been achieved for the more expressive language illustrates the importance of making problem structure explicit so that it can be exploited computationally [Rintanen, 2015; Ivankovic and Haslum, 2015].

Significantly enough, there is no domain featuring existential quantifiers in any of the last three international planning competitions. This lack is arguably due to the inefficiency of the standard compilation technique, which imposes a heavy burden on both modelers and planners. We hope that this work can contribute to change this situation.

Acknowledgments

The `ES` and `FS` planners are partially built on top of the Lightweight Automated Planning Toolkit (`www.lapkt.org`) and the Fast Downward PDDL parser (`www.fast-downward.org`). The authors thank Miquel Ramírez for useful discussions and insights. This work is partially supported by the M. de Maeztu Programme (MDM-2015-0502) and by grants TIN2015-67959 and CSD2010-00034, MEC, Spain.

References

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

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

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

[Do and Tran, 2013] M. Do and M. Tran. Blocksworld: An iPad puzzle game. In *Proc. 3rd ICAPS Planning in Games Workshop*, pages 35–39, 2013.

[Fikes and Nilsson, 1971] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971.

[Francès and Geffner, 2015] G. Francès and H. Geffner. Modeling and computation in planning: Better heuristics from more expressive languages. In *Proc. 25th Int. Conf. on Automated Planning and Scheduling*, pages 70–78, 2015.

[Gazen and Knoblock, 1997] B. Gazen and C. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proc. 4th European Conf. on Planning (ECP)*, pages 221–233, 1997.

[Gecode Team, 2006] Gecode Team. Gecode: Generic constraint development environment, 2006. Available from `http://www.gecode.org`.

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

[Gottlob *et al.*, 1999] G. Gottlob, N. Leone, and F. Scarcello. On tractable queries and constraints. In *Database and Expert Systems Applications*, pages 1–15. Springer, 1999.

[Haslum, 2008] P. Haslum. Additive and reversed relaxed reachability heuristics revisited. *Proc. 2008 Int. Planning Competition*, 2008.

[Helmert, 2006] M. Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

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

[Ivankovic and Haslum, 2015] F. Ivankovic and P. Haslum. Optimal planning with axioms. In *Proc. 24th Int. Joint Conf. on Artificial Intelligence*, pages 1580–1586, 2015.

[McDermott *et al.*, 1998] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT, 1998.

[McDermott, 1999] D. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111–159, 1999.

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

[Rintanen, 2015] J. Rintanen. Impact of modeling languages on the theory and practice in planning research. In *Proc. 29th AAAI Conf. on Artificial Intelligence*, 2015.