Planning with State Constraints and its Application to Combined Task and Motion Planning

Jonathan Ferrer-Mestres
Universitat Pompeu Fabra
Barcelona, Spain
jonathan.ferrer@upf.edu

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 planning languages where action preconditions and goals are conjunctions of propositional atoms. These restrictions make the definition and computation of planning heuristics easier but limit the expressive capabilities of the resulting planners. As a result, standard planners are unable to capture the type of geometrical reasoning that is critical in robotics domains where both robots and objects have geometrical dimensions and collisions are to be avoided. Such problems are addressed in robotics by combining *task planners* that handle the symbolic reasoning part with *motion planners* that check the geometrical feasibility of the plans output by the task planners. This decomposition however may result in a lot of backtracking as the symbolic and geometrical components are not independent. The aim of this work is to provide an alternative integration of task and motion planning where the symbolic and geometrical components are addressed in combination, with neither part taking the back seat. For this, we build on the recent planner `FS0` that is able to handles an expressive variable-free, first-order planning language called Functional STRIPS where constraints, functions and numerical variables are accommodated, and extend it to handle also *state constraints* — namely, formulas that must be true in all states. We then use *functions* for encoding the geometrical dimensions and poses of objects, and *state constraints* to express that no pair of objects, including the robot, can overlap in space. We illustrate the functionality and performance of the planner over a number of examples.

Introduction

Classical AI planners are currently able to solve problems over very large state spaces. In classical planning, the initial state is fully known and actions are assumed to have deterministic effects. The main techniques rely on the use of heuristics that are derived automatically in order to guide a state-space search or on translations into propositional satisfiability (Russell and Norvig 2002; Ghallab, Nau, and Traverso 2004; Geffner and Bonet 2013). However, the restrictions in planning modeling languages that have facilitated these developments, have also limited the scope of the resulting planners. In particular, standard planners do not appear to be suitable for capturing the type of geometrical reasoning that is critical in robotics where both robots and objects have geometrical dimensions and collisions are to be avoided. Such problems are addressed instead through a combination of two types of planners: task planners that handle the high-level, symbolic reasoning part, which correspond to the AI planners, and motion planners (LaValle 2006) that plan the movements in space and handle the geometrical constraints (Cambon, Gravot, and Alami 2004; Wolfe, Marthi, and Russell 2010; Lagriffoul et al. 2012; Lozano-Pérez and Kaelbling 2014). The symbolic and the geometrical components, however, are not independent, and hence, by giving one of these two parts the secondary role of verifying feasibility, approaches based on task and motion decomposition are doomed to produce lots of backtracks. The problem of excessive backtracks is well-known in constraint satisfaction when constraints are used *passively* (Mackworth 1977; Dechter 2003). The computational solution to this problem is the interleaving of search with forms of constraint propagation that make the constraints an *active* part in the search. The situation is similar when searching for a goal in a graph: blind-search methods where the goal plays a passive role cannot scale as well as methods where the goal directs the search by means of a heuristic function.

The aim of this work is to provide an alternative integration of task and motion planning where the symbolic and geometrical components in robot planning are addressed in combination and both parts play an active role in directing the search for plans. For this, we build on the recent planner `FS0` (Francès and Geffner 2015) that handles an expressive variable-free, first-order planning language called Functional STRIPS (Geffner 2000) which naturally accommodates constraints, functions, and numerical variables *both in the specification of the problem and in the computation of the heuristics*. On top of that, we extend `FS0` and Functional STRIPS with the ability to handle *state constraints*, *i.e.* formulas that must be true in all states encountered through the execution of a plan. We use *functions* for encoding the geometrical dimensions of objects and their poses, and *state constraints* to express that no two objects, including the robot, can overlap in space. State constraints are not a standard feature of planning languages, although their convenience has been thoroughly discussed in the literature on reasoning about actions (Lin and Reiter 1994; Son et al. 2005), sometimes under the name of *plan invariants*.[1] In our case, state constraints can be thought of as a

[1] State constraints should not be confused with state invariants: a

convenient way to express preconditions that are implicit for all actions (*i.e.* actions leading to the violation of a state constraint are deemed not applicable), states to be avoided (*e.g.*, states where a formula $p \wedge q$ is true),[2] and dead-end conditions, that is conditions that if ever achieved preclude reaching the goal. Functions are then used in the planner to encode the geometrical dimensions of objects and their poses, and state constraints to express that objects should not overlap in space. The functionality and performance of the planner will be illustrated over some examples.

The rest of the paper is organized as follows. We first review the Functional STRIPS planning language, whose expressivity is key for handling "motion planning" as a particular form of "task planning", and the recent planner `FS0`, which handles a large fragment of Functional STRIPS. We then show how to extend this planner for handling state constraints. Finally, we use the resulting planner for modeling and solving planning problems with state constraints, and in particular, problems that combine motion and task planning. Preliminary experimental results are reported, and simulations of the resulting plans can be seen in videos.[3]

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, typical 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.

A FSTRIPS problem is a tuple $P = \langle F, I, O, G \rangle$ where I and G are the initial and goal formulas, O is a set of actions, and F describes the symbols and their types. The formula I along with the external procedures must define a unique initial denotation for each of the symbols in F.

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 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 *i.e.* $c^* = c$. 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 (external procedures) 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 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

state constraint *enforces* a formula to be true in all reachable states, while a state invariant is a formula that can be proven to hold in all reachable states. For example, block A not being on top of two blocks at the same time is a typical state invariant in blocksworld, whereas A not being on top of block B might be a particular state constraint that we want to enforce.

[2] State constraints ϕ used this way model the class of extended temporal goals "never ϕ" (Dal Lago, Pistore, and Traverso 2002).

[3] `www.bitbucket.org/ferrerj/ctmp`

$loc(b) = b'$ 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 $P = \langle F, I, O, G\rangle$ 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.

Example

A simple planning problem involving a set of integer variables $X_1, \ldots, X_n$ and actions that allow us to increase or decrease the value of any variable by one within the $[0, n]$ interval, can be modeled in Functional STRIPS by treating the variables X_i as 0-arity fluent functional symbols with values ranging in the $[0, n]$ interval. These X_i symbols represent the state variables in the problem. If $I = \{X_1 = 0, \ldots, X_n = 0\}$, and $G = \{X_1 < X_2, \ldots, X_{n-1} < X_n\}$, the problem is about changing the value of the X_i variables from 0 to final values that increase monotonically with i. The precondition-free action $inc(X_i)$ has the effect $X_i := \min(X_i + 1, n)$, whereas $dec(X_i)$ has the effect $X_i := \max(X_i - 1, 0)$.

A Functional STRIPS Planner

The `FS0` planner (Francès and Geffner 2015) deals with a large fragment of the Functional STRIPS language where preconditions and goal formulas must be in CNF and each clause may involve two state variables at most, with the exception of atoms expressing *global constraints* (van Hoeve and Katriel 2006; Rossi, Van Beek, and Walsh 2006). For example, the goal of arranging a set of blocks in blocksworld into a single tower can be expressed succinctly as $\texttt{alldiff}(loc(b_1), \ldots, loc(b_n))$ where `alldiff` represents the standard *all-different* constraint. Like constraint solvers, `FS0` gets a computational benefit from combining the $O(n^2)$ inequality constraints $loc(b_i) \neq loc(b_j)$ into one single `alldiff` global constraint, as this enables more powerful forms of constraint propagation that yield more informed heuristic values. We next focus on the computation of these heuristics in `FS0`, where they are used to guide a standard *greedy best-first search*.

Computation of the Heuristic

As noted by several authors, some of the heuristics that are useful in STRIPS like $h_{\max}$ (Bonet and Geffner 2001) and h_{FF} (Hoffmann and Nebel 2001) 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 (RPG)** 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, $X^0 = \{X^s\}$, s being the state whose heuristic value 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 set of possible denotation 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 $x \in w^k$. 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. A *relaxed plan* $\pi_{\text{FF}}(s)$ can be obtained then 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 $\pi_{\text{FF}}(s)$, 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`. The heuristic $h_{\max}(s)$ is given by the index k of the first layer P_k where the goal is true.

Logical Generalization: Constrained RPG. 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. A formula like $Pre(a)$ is *satisfiable* in layer 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 interpreta-**

tion of the propositional layers P_i affects the contents and computation of the RPG, which we now call **Constrained RPG (CRPG)**. 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. Heuristics analogous to h_{max} and h_{FF} (which we name h^*_{max} and h^*_{FF}) can then be obtained from such a graph in the usual way.

Polynomial Approximation: Constraint Propagation. The heuristics h^*_{max} and h^*_{FF} correctly assign infinite heuristic values to logically inconsistent goals such as $(X < 3 \wedge X > 5)$, which get finite values in the unconstrained heuristics h_{max} and h_{FF} when each goal $X < 3$ and $X > 5$ is reachable separately. The problem with such heuristics is that they are *intractable*. The two heuristics h^c_{max} and h^c_{FF} supported in the `FS0` planner are aimed at approximating these two heuristics in an efficient, *polynomial* manner. For this, it is assumed that action preconditions, conditions, and goals are conjunction of atoms, and that fluent symbols do not appear nested. Under these restrictions, checking whether the goal G is *satisfiable* in a propositional layer P_k boils down to solving a *constraint satisfaction problem* G^k whose *variables* are the state variables X appearing in G, the *domain* $D(X)$ of the variables X is X^k, and the constraints are given by the *atoms* in G. In the approximation, the goal G is deemed *satisfiable* simply if the local consistency methods do not prove that the CSP is unsatisfiable by leaving a variable with an empty domain. The local consistency methods are node and arc consistency (Dechter 2003; Rossi, Van Beek, and Walsh 2006). The h^c_{max} heuristic is defined by the index k of the first layer P_k where the goal is deemed satisfiable in this manner. Arc consistency applies only to binary constraints, *i.e.* to atoms involving two state variables. For atoms involving more variables, local consistency algorithms that depend on the type of (global) constraint are used instead. Currently, `FS0` supports just two types of global constraints, `alldiff` and `sum` (Rossi, Van Beek, and Walsh 2006), but it offers the possibility of defining new global constraints by providing their associated local consistency algorithms.

As an illustration of these heuristics, the problem above with variables X_i that must be increased or decreased from 0 until achieving the inequalities $X_i < X_{i+1}, i = 1, \ldots, n-1$, yields an h_{max} value of 1 when the value-accumulating semantics is used, or when the problem is compiled into STRIPS. On the other hand, the constrained heuristic h^*_{max} and its polynomial approximation h^c_{max} have the optimal value n. This is because while each of the goal atoms $X_i < X_{i+1}$ is reachable in one step in the RPG, their *conjunction* is satisfiable in both the constrained RPG and its polynomial approximation only after n steps.

State Constraints

By accommodating numerical variables, constraints, and functions, it is possible to express in Functional STRIPS problems that involve geometrical reasoning. For example, one can easily deal with a problem involving a square object o with side length d by using a representation in which the function $config(o)$ denotes its 2D location $\langle x, y \rangle$ which can be changed through translation actions. By a suitable discretization of the set of possible configurations, one can then express the problem of manipulating the object o from a given initial configuration into a final configuration where, for example, a certain position (x_c, y_c) must be covered by the object. For this, the goal G can be expressed as the formula $(x - d \leq x_c \leq x + d) \wedge (y - d \leq y_c \leq y + d)$ that reduces to four atoms involving two state variables x and y. Actually, the current version of `FS0` handles the negation of such goals as well, which also involves two variables, and expresses the problem where the object must be placed in a location where the target point is not covered. Thus, it is simple to capture in `FS0` problems where an object must be moved in order to comply with some *geometrical goal constraint*. What is less simple to do in Functional STRIPS is to enforce such constraints *throughout the execution* of the plans which is critical in robotics. For doing this, however, we just need to treat such constraints, not as goal constraints, but as *state constraints* (Lin and Reiter 1994; Son et al. 2005), i.e., constraints applying to all states not just goal states. The changes required in the language and in the derivation of heuristics for accommodating state constraints are minor, but the expressivity gained is significant.

Syntax and Semantics

A FSTRIPS planning problem with *state constraints* is a tuple $P = \langle F, I, O, G, C \rangle$ where the new component C stands for a set of formulas expressing the constraints. The syntax for these formulas is the same as for those encoding the goal G but their semantics is different. State constraints are used for encoding *implicit preconditions.* Namely, an action a is deemed applicable in a state s when *both* $[Pre(a)]^s = true$ *and* the state s_a that results from applying a to s is such that $c^s = true$ for every state constraint $c \in C$. In other words, an action a is *non-executable* in a state s where its precondition $Pre(a)$ holds, if its execution leads to a state s_a that violates some state constraint. In addition, the unique initial state must satisfy all the state constraints as well. As a result, if $c \in C$ is a state constraint and $s_0, \ldots, s_n$ is the sequence of states generated by a plan that solves P, then c will be true in all the states $s_i, i = 0, \ldots, n$.

Heuristics

The introduction of state constraints affects the definition of the heuristics h^*_{max} and h^*_{FF} obtained from the constrained RPG, and the polynomial approximations h^c_{max} and h^c_{FF} supported in the `FS0` planner. The changes, however, are minor. First, recall that a layer P_k encodes sets X^k of possible values for each state variable X, which in turn define sets V^k of possible interpretations over the language. An action a is deemed applicable in layer P_k if one such interpretation satisfies the formula $Pre(a)$; similarly, the goal G is taken to be true for the purpose of computing a constrained relaxed plan, if one such interpretation satisfies G. In the presence of state constraints, this remains the same except that interpretations in V^k that do not satisfy a state constraint are pruned first. In the polynomial approximation that leads to the heuristics

h^c_{max} and h^C_{FF}, the state constraints are not used to prune interpretations directly but just the domains X^k of the state variables X in layer k through constraint propagation.

Examples

We illustrate next the usefulness of state constraints both in terms of modeling and computation with a couple of examples, reporting their running times as well.

The **Missionaries and Cannibals** (M&C) problem has received wide attention since the early days of AI (Amarel 1968) as a toy problem that is nevertheless representative of a wider class of transportation-under-constraints problems. In its standard version, the problem places three missionaries and three cannibals on the left bank of a river which they all want to cross. A single boat is available that can hold only two people at a time, regardless of whether they are missionaries or cannibals. Apparently, the missionaries do not want to be outnumbered by the cannibals, be it on either bank of the river or inside of the boat, for fear of the cannibals exercising their defining inclination.[4] The goal is to find an appropriate schedule of river crossings that transports everyone to the right bank of the river in a safe manner.

We model a generalization of the problem for n missionaries and n cannibals ($n \geq 3$) on a complete graph. There are fixed symbols $l_1, \ldots, l_m$ representing the locations, and state variables $nc(l)$ and $nm(l)$ representing the number of cannibals and missionaries at each location l. In addition, the 0-ary functional symbol X represents the current location of the boat. The actions $move(c, m, l)$, with $0 \leq c, m \leq 2$, $c+m \in \{1, 2\}$, move c cannibals and m missionaries in one boat trip from the current location to l. Their preconditions are $c \leq nc(X)$ and $m \leq nm(X)$, and their effects are $X := l$, $nc(l) := nc(l)+c$, $nc(X) := nc(X)-c$, and analogously for the missionaries. Finally, the restriction on cannibals not outnumbering missionaries in a location l is modeled by the binary state constraint $nm(l) \geq nc(l) \vee nm(l) = 0$. The "inside of the boat" restriction is encoded as part of the $move$ action.

As a second example, consider a **simple navigation with geometrical obstacles** problem in which an $n \times m$ grid contains a robot that has to reach a goal cell while avoiding obstacles. For simplicity, we assume a point robot with no geometry, and obstacles o with rectangular shape which can be represented by a couple of coordinate points (x_o, y_o) and (x'_o, y'_o), with $x_o < x'_o$ and $y_o < y'_o$ (obstacles having other shapes can be thought of as a combination of smaller rectangles). The location of the robot is represented by two 0-arity fluent functional symbols x and y with values in $[1, n]$ and $[1, m]$. The actions $move(dx, dy)$ with $dx, dy \in [-1, 1]$ move the agent to adjacent locations, including diagonals, with effects $x := max(0, min(x + dx, n))$ and $y := max(0, min(y + dy, m))$. Avoidance of obstacles o in any plan can then be represented succinctly through the state constraint $\neg(x_o \leq x \leq x'_o \wedge y_o \leq y \leq y'_o)$, resulting in a problem with as many state constraints as obstacles, and where each of the state constraints involves the same two state variables x and y.

[4] A historically more accurate version of the problem has it that it is the cannibals that do not want to be outnumbered by the missionaries for fear of being converted, but we restrict our discussion to the first version for the sake of tradition.

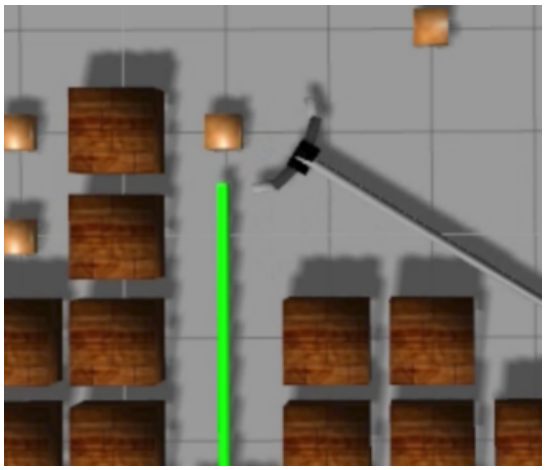

Figure 1: Arm rotating to grasp an object

We have actually tested the planner on a number of randomly generated instances of increasing size for each of these two domains.[5] In the case of the M&C domain, the planner scales up pretty well, handling problem sizes of 20-node location graphs and 12 missionaries plus 12 cannibals with relative ease: an instance with a 10-node graph and $9+9$ missionaries and cannibals is solved in 80 sec. by finding a plan of length 49 after expanding 637 nodes. An instance with 20 nodes and $12 + 12$ individuals takes 379 sec. and 183 node expansions to find a plan of length 45. Similarly, the planner handles navigation problems with a linear number of geometrical obstacles in less than 0.1 sec. for 10×10 grids and less than 15 sec. for 50×50 grids.

Task and Motion Planning Combined

The last example illustrates how problems involving geometrical constraints can be modeled in Functional STRIPS with state constraints, and solved by the `FS0` planner. We now consider a more general type of problem involving a robot that can translate, rotate (Figure 1), and pick and place objects (Figure 2). Robots and objects can have arbitrary 2D geometries, and collisions are to be avoided. In one task, which we name **moving with geometrical obstacles**, the robot must reach a goal configuration by navigating and moving the objects that obstruct the path. In the other task, which we name **tidying up**, the robot must place the objects in some goal configuration. In both cases, the 2D space of the environment is discretized according to a parameter r into a regular grid of size $r \times r$. Robots and objects have a configuration that captures triplets $\langle x, y, \theta \rangle$, where x and y capture the center-of-mass position of the object or robot within the discrete grid, and θ its orientation, with angles being discretized into 45 degrees.

[5] Problem encodings for which empirical results are discussed are available at `www.bitbucket.org/ferrerj/ctmp`.

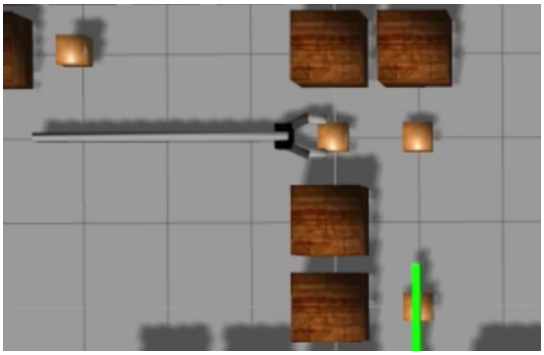

Figure 2: Gripper grasping an object

The configuration $config(o)$ of an object or robot o is represented with the functional fluent $config$. Translations and rotations of the robot change the robot configuration, and, if the robot is holding and object, they change the configuration of the object too. These changes are all expressed by means of externally defined procedures. In addition, the $pickup(o)$ action has precondition $graspable(config(r), config(o))$ where r is the robot, o the object to be picked up, and $graspable$ is a fixed predicate symbol whose denotation is externally defined too.

The *avoidance of collisions between movable and static objects* is captured by the fixed external procedures that perform the configuration updates. If, say, a translation of the robot would make it collide with a static object, the procedure updates the configuration to the particular invalid value $\perp$. The *avoidance of collisions among movable objects*, on the other hand, is handled in a pairwise fashion by the binary state constraints $non\text{-}overlap_{i,j}(config(o_i), config(o_j))$, where the fixed predicate symbol $non\text{-}overlap_{i,j}$ is defined by an external procedure that is a function of the geometries and configurations of the objects i and j. The predicate is true when there is no grid cell occupied by both objects given their fixed geometries and current configurations. For the sake of performance, all reasoning involving object configurations is precompiled into extensional form, meaning that all the fixed external functions are extensionally stored by means of tables that are indexed by the configuration identifiers.

Illustrations showing the initial, goal, and intermediate configurations resulting from plans computed by `FS0` on the "moving with geometrical obstacles" problem are shown in Figure 3. Movable objects are depicted by little squares, static objects by brown boxes, the robot is stick-shaped with a gripper and its final configuration is shown in green.

FSTRIPS Model Details

We provide a few additional details on the model of the problems in FSTRIPS. Besides the $config$ symbol, two auxiliary fluent functional symbols $handempty$ and $held$ are used, the first boolean, the second denoting the object being held. The functions used to capture the changes in the configurations are $translated$, $rotated$ and $o\text{-}rotated$ (standing for the rotation of the object being held). The first function is used in the effects of the translation action, the second in the rotation action, and the third in the rotation action with an object. More precisely, $translated(o, c, d)$ is the configuration that results from applying an *atomic translation* to the robot or object o when in configuration c, where d is a direction; i.e., one of the the fixed symbols N, NE, E, SE, S, SW, W, NW. In turn, $rotated(o, c, d)$ is the result of applying an *atomic rotation* with d being one of the fixed symbols cw and ccw (*clockwise*, *counterclockwise*). Finally, $o\text{-}rotated(o, c, d)$ denotes the resulting configuration of the obstacle o held by the robot when the robot performs a rotation in direction d. By *atomic* translations and rotations we refer to step-wise operations that translate or rotate the object to the next cell or angle respectively, according to the chosen discretization. Having the objects move only one step at a time is necessary to ensure that no collisions happen along trajectories, while keeping the total number of (grounded) actions small and independent of the total number of configurations.

The problem model requires 6 action types `translate`, `rotate` (when the robot is holding no object), `pick-up`, `place`, `translate-with-object`, and `rotate-with-object` (when the robot is holding an object). We omit the full specification and limit our description to the action `rotate-with-object`(o, d), where $d \in \{cw, ccw\}$, and which has a single precondition $held = o$ and two effects: $conf(r) := rotated(r, conf(r), d)$ and $conf(o) := o\text{-}rotated(o, conf(o), d)$. All action preconditions are CNF formulas with clauses involving a single state variable, with the sole exception of the precondition of the `pick-up` action which contains a clause making use of the $graspable$ predicate that involves two state variables.

Experimental Results

We report next the empirical results of the `FS0` planner when run on a number of problems from the MOVING-GEOM-OBSTACLES and TIDYING-UP domains.[6] The results are from running the planner on an AMD Opteron 6300 machine with a 2.4Ghz clock, with a time bound of 30 minutes and a memory bound of 8GB.

Table 1 shows the per-instance results of the planner, focusing on plan length (*i.e.* number of actions in the plan), total number of expanded nodes along the search, and total runtime of the plan search. In both domains, the complexity of the problem clearly grows with both the discretization resolution and the number of movable objects as expected. In general, the higher the value of any of these two parameters, the longer the plans, because steps are smaller and more objects may have to be moved.

MOVING-GEOM-OBSTACLES problems with 10×10 resolution grids are solved by `FS0` with ease, even with 5 obstacles. The number of node expansions during the greedy best-first search grows with the number of objects but is low, suggesting that the heuristic is informative. Grid resolutions of 30×30 and 50×50 pose a more significant challenge, but nevertheless the planner solves the instances with $1 - 3$ objects with a low number of node expansions.

[6] Simulations of the obtained plans can be seen at `www.bitbucket.org/ferrerj/ctmp`.

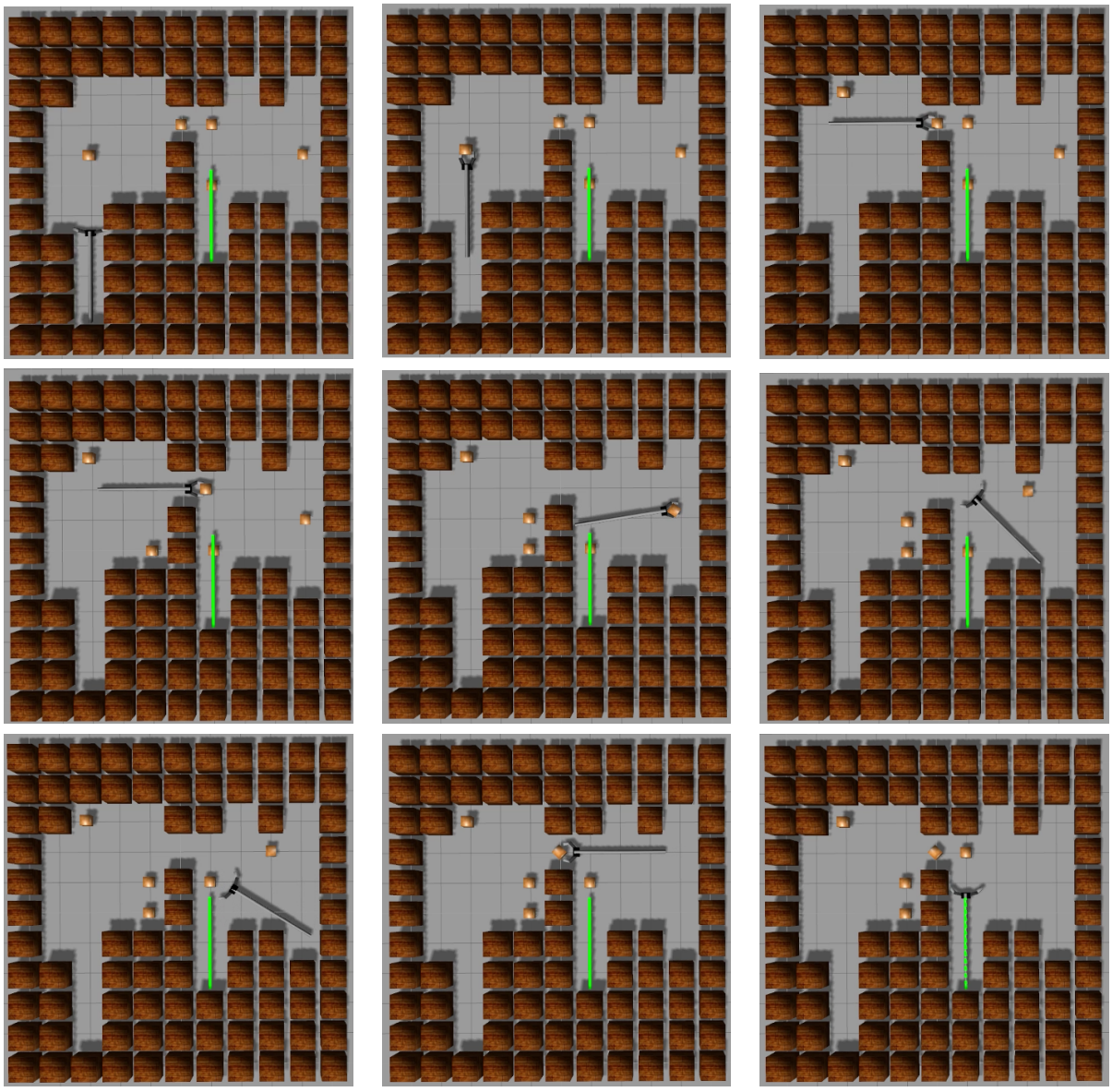

Figure 3: The MOVING-GEOM-OBSTACLES domain in which a robot has to reach a target position but a number of obstacles need to be picked up and moved around so that they stop blocking the robot's path to the goal. Snapshots from top left to bottom right show several steps of the solution, in which the robot reaches the goal configuration after clearing the path.

Problem			FSTRIPS model				Solution			
Type	Res.	Obj.	vars	configs	actions	constr.	length	#Exp	Time (s.)	#Exp per second
MGO	10×10	1	4	212	22	3	18	19	0.16	118.75
MGO	10×10	2	6	212	34	6	19	21	0.45	46.67
MGO	10×10	3	8	212	46	10	23	33	1.13	29.20
MGO	10×10	4	10	212	58	15	41	169	8.10	20.86
MGO	10×10	5	12	212	70	21	59	397	10.56	37.59
MGO	30×30	1	4	894	22	3	44	83	8.35	9.94
MGO	30×30	2	6	894	34	6	49	55	16.39	3.36
MGO	30×30	3	8	894	46	10	55	119	55.26	2.15
MGO	30×30	4	10	894	58	15	-	1475	>1800	0.82
MGO	30×30	5	12	894	70	21	-	2139	>1800	1.19
MGO	50×50	1	4	2120	22	3	74	218	110.43	1.97
MGO	50×50	2	6	2120	34	6	78	80	72.14	1.11
MGO	50×50	3	8	2120	46	10	83	222	548.75	0.40
MGO	50×50	4	10	2120	58	15	-	669	>1800	0.37
MGO	50×50	5	12	2120	70	21	-	1458	>1800	0.81
TU	10×10	2	6	226	34	6	12	14	0.06	233.33
TU	10×10	3	8	227	46	10	35	63	0.43	146.51
TU	30×30	2	6	908	34	6	54	143	13.24	10.80
TU	30×30	3	8	909	46	10	112	6564	164.50	39.90
TU	50×50	2	6	2134	34	6	47	154	5.51	27.95
TU	50×50	3	8	2135	46	10	121	17449	1675.72	10.41

Table 1: Performance of `FS0` on MOVING-GEOM-OBSTACLES (MGO) and TIDYING-UP (TU) instances, described in the text. Instances differ on the resolution of the environment discretization (col. 2) and on the number of movable objects (col. 3). Columns 4 to 7 report information about the characteristics of the FSTRIPS encoding, namely the number of state variables, the number of object configurations, the number of (grounded) actions, and the number of state constraints. The four rightmost columns respectively report (1) the length of the plan (a dash denotes that no plan was found), (2) the number of nodes expanded along the search, (3) the total runtime until the plan is found, and (4) the rate at which nodes are expanded. Simulations of the execution of the computed plans can be visualized at `www.bitbucket.org/ferrerj/ctmp`.

In turn, TIDYING-UP problems are also solved by expanding few nodes in the simpler cases, although the heuristic is less informative for the higher resolutions and 3 objects.

In general, the rate of nodes expanded per second decreases sharply with the resolution and number of objects. This is indicative of the cost of computing the heuristic, which is affected by the number of state variables associated with the objects and by the total number of configurations, which in turn depends on the chosen resolution. Indeed, the number of object configurations in a grid discretized with resolution $r \times r$ and k possible orientations is in the order of $k \cdot r^2$, although many of these configurations will not be feasible, as they overlap with an static object in the map. The empirical results show the feasibility of the approach, but work is still needed to improve scalability further. In particular, one possibility to address the blowup that our approach incurs when increasing the resolution or scaling up to a 3D representation would be the use of an encoding along the lines of the "navigation with geometrical obstacles" example presented above, where configurations are modeled in terms of three distinct state variables x, y and θ instead of one. Particular attention should be paid to how this type of change affects the quality of the resulting heuristics. Another issue that deserves further consideration is the impact of the precompilation of all external procedures into extensional form. This offers an advantage on small-scale domains, but it is not clear how useful it is on larger domains.

Related Work

Combined task and motion planning for grasping and manipulation is an open research problem at the intersection of planning and robotics. The aSyMov planner (Cambon, Gravot, and Alami 2004; Gravot, Cambon, and Alami 2005), for instance, already aimed at integrating both symbolic and geometric reasoning at each step of the planning process. External procedures have also been used to avert the difficulty of performing geometric reasoning within a logically-oriented planning language (Dornhege et al. 2010), as well as to predict the effects of actions involving complex physics (Kunze and Beetz 2015).

Hierarchical approaches to the problem have also been explored in (Kaelbling and Lozano-Pérez 2011), where a hierarchical regression-based schema is developed that combines task and motion planning, and in (Wolfe, Marthi, and

Russell 2010), where Hierarchical Task Networks are used to tackle robotic manipulation problems by modeling the bottom actions of the hierarchy with motion planning. Manipulation planning problems are also tackled through symbolic planning in (Nebel, Dornhege, and Hertle 2013).

An alternative approach is that presented in (Srivastava et al. 2014), where off-the-shelf classical and motion planners are integrated through the use of a planner-independent interface layer. The errors in the motion planning goals have to be identified and fed back to the symbolic planner in the form of logic predicates, the main challenge being the proper identification of the offending atoms that prevent the enactment of specific high-level actions. (Garrett, Lozano-Pérez, and Kaelbling 2014) computes an heuristic that takes the geometrical information into account together with the symbolic information, exploiting a conditional reachability graph as a form of a probabilistic roadmap conditioned to the object configurations. (Lagriffoul et al. 2012) integrates task and motion planning proposing a technique to reduce the geometric configuration space and thus the number of calls to the motion planner. A key difference between all these approaches and ours, however, is *the formulation of motion planning as an additional component of the task planner.*

Summary

We have proposed an alternative integration of task and motion planning where the symbolic and geometrical components are addressed in combination, with neither part taking the back seat. For this, we have built on an expressive planning language, Functional STRIPS, that supports constraints, functions, and numerical variables, and on the planner `FS0`, which supports a large fragment of this language in the specification of problems and is crucially able to exploit its expressivity in the computation of heuristics. We have extended this language and computational model with state constraints: logical formulas that must hold true in every state of a plan. In order to address motion and task planning problems, we use *functions* for encoding the geometrical dimensions of objects and their poses, and *state constraints* to express that no two objects, including the robot, can overlap in space. The experiments reported are preliminary but illustrate the feasibility of the approach. There is a lot of room for improving performance and for exploring the possibilities that are afforded by this integration of motion planning into task planning. In particular, scaling up well in the presence of large grids and many objects remains a challenge. In principle, however, there is no need for the grids to be regular: it would be more natural to use higher resolutions around the current robot configuration and lower resolutions elsewhere. Alternatively, maps obtained from random configuration sampling, as in probabilistic roadmaps, could be used instead. The strength of the integration proposed is that it is very general and independent of these choices.

References

Amarel, S. 1968. On representations of problems of reasoning about actions. *Machine intelligence* 3(3):131–171.

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

Cambon, S.; Gravot, F.; and Alami, R. 2004. aSyMov: Towards more realistic robot plans. In *Proc. of ICAPS*.

Dal Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Proc. AAAI*, 447–454.

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 *Proc. of ICAPS*, 114–121.

Dornhege, C.; Eyerich, P.; Keller, T.; Brenner, M.; and Nebel, B. 2010. Integrating task and motion planning using semantic attachments. In *Bridging the Gap Between Task and Motion Planning*.

Francès, G., and Geffner, H. 2015. Modeling and computation in planning: Better heuristics from more expressive languages. In *Proc. of ICAPS*, 70–78. AAAI Press.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2014. FFRob: An efficient heuristic for task and motion planning. In *Proc. Int. Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool.

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.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Morgan Kaufmann.

Gravot, F.; Cambon, S.; and Alami, R. 2005. aSyMov: a planner that deals with intricate symbolic and geometric problems. In *International Symposium on Robotics Research*, 100–110. Springer.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning modulo theories: Extending the planning paradigm. In *Proc. of ICAPS*.

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 *Proc. of ICAPS*.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *International Conference on Robotics and Automation (ICRA)*, 1470–1477. IEEE.

Kunze, L., and Beetz, M. 2015. Envisioning the qualitative effects of robot manipulation actions using simulation-based projections. *Artificial Intelligence*.

Lagriffoul, F.; Dimitrov, D.; Saffiotti, A.; and Karlsson, L. 2012. Constraint propagation on interval bounds for dealing with geometric backtracking. In *International Conference on Intelligent Robots and Systems (IROS)*, 957–964. IEEE.

LaValle, S. M. 2006. *Planning algorithms*. Cambridge.

Lin, F., and Reiter, R. 1994. State constraints revisited. *Journal of logic and computation* 4(5):655–677.

Lozano-Pérez, T., and Kaelbling, L. P. 2014. A constraint-based method for solving sequential manipulation planning problems. In *International Conference on Intelligent Robots and Systems (IROS)*, 3684–3691. IEEE.

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

Nebel, B.; Dornhege, C.; and Hertle, A. 2013. How much does a household robot need to know in order to tidy up? In *Proceedings of the AAAI Workshop on Intelligent Robotic Systems*.

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

Russell, S., and Norvig, P. 2002. *Artificial Intelligence: A Modern Approach*. Prentice Hall. 2nd Edition.

Son, T. C.; Tu, P. H.; Gelfond, M.; and Morales, A. 2005. Conformant planning for domains with constraints: A new approach. In *Proc. AAAI-05*, 1211–1216.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *International Conference on Robotics and Automation (ICRA)*, 639–646. IEEE.

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

Wolfe, J.; Marthi, B.; and Russell, S. J. 2010. Combined task and motion planning for mobile manipulation. In *Proc. of ICAPS*, 254–258.