

PLANING PROBLEMS

*Learning Sketches for Decomposing Planning
Problems into Subproblems of Bounded Width*

Joshua Gordon





OVERVIEW

This presentation will provide an overview of the paper "Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width" (by Dominik Drexler, Jendrik Seipp, Hector Geffner)

The paper introduces *sketches* as a general language for representing the subgoal structure of planning problems and describes how sketches can be used to decompose planning problems into subproblems of bounded width. that can be solved greedily, in polynomial time by the SIWR variant of the SIW algorithm.

The paper then addresses the problem of learning sketches automatically given a planning domain, instances of the target class of problems and the desired bound on the sketch width.

Finally the paper presents experimental results.

OUR GOAL

How sketches can encode decompositions of bounded width that can be solved greedily, in polynomial time, by the SIWR variant of the SIW algorithm.



NEXT UP

We review

Classical Planning



Sketches



Width



from Lipovetzky and Geffner (2012), Bonet and Geffner (2021), and Drexler, Seipp, and Geffner (2021).



INTRODUCTION:

Classical planners manage to solve problems that span exponentially large state spaces by exploiting problem structure.

In classical planning, the environment is represented by a set of states, actions, and a transition function that maps states to the resulting state after executing an action. The goal is to find a sequence of actions that lead from the initial state to a goal state.

Domain-independent methods make implicit assumptions about structure, such as subgoals being independent or having low width.

Domain-dependent methods, on the other hand, usually make problem structure explicit in the form of hierarchies that express how tasks decompose into subtasks



CLASSICAL PLANNING - STATE MODEL

An instance P defines a state model $S(P) = (S, s_0, G, Act, A, f)$

- S is the set of all possible states that can be reached from the initial state s_0 by applying actions from A . The ground atoms and the set of literals together make up the set of all possible states S in the state model. the ground atoms represent the different conditions that can exist and the set of literals represents the different combinations of ground atoms that can be true in a particular state.
- s_0 is the initial state of the problem.
- G is the set of goal states that satisfy the goal condition specified in P .
- Act is the set of all possible actions that can be performed in any state in S .
- $A(s)$ is the subset of Act that contains only those actions whose preconditions are true in state s .
- $f(s, a)$ is a function that maps a state s and an action $a \in A(s)$ to the successor state s' obtained by applying action a to state s .

$S(P)$ defines all possible states that can be reached from an initial state by applying actions from Act .

STATE MODLE SIMPLE EXAMPLE:

state model for controlling the temperature of a room

$$S(P) = (S, s_0, G, Act, A, f)$$

- Ground Atoms: "Room temperature is too hot", "Room temperature is too cold", "Heater is on", "AC is on"
- Set of Literals: {"Room temperature is too hot"}, {"Room temperature is too cold"}, {"Heater is on"}, {"AC is on"}, {"Room temperature is too hot" AND "AC is on"}

Initial State (s_0): {"Room temperature is too hot"}

Goal State (G): {"Room temperature is just right"} (not too hot not too cold)

Actions (Act): TurnOn(Heater), TurnOff(Heater), TurnOn(AC), TurnOff(AC)

Applicable Actions ($A(s)$): If $s = \text{"Room temperature is too hot" AND "Heater is off"}$, then $A(s) = \{\text{TurnOn(Heater), TurnOn(AC)}\}$

State Transition Function (f): If $a = \text{TurnOn(Heater)}$ and $s = \text{"Room temperature is too hot" AND "Heater is off"}$, then $f(s, a) = \text{"Room temperature is to hot" AND "Heater is on"}$



CLASSICAL PLANNING - STATE MODLE

a plan π for a planning problem instance P is a sequence of actions that can be executed in order to achieve the goal specified by P . A plan π for P is said to be *valid* if executing the actions in π starting from any initial state in P 's set of initial states leads to a state that satisfies the goal condition G .

A state s is *solvable* if there exists a plan starting at s , otherwise it is unsolvable (also called *dead-end*). Furthermore, a state s is *alive* if it is solvable and it is not a goal state. The *length* of a plan is the number of its actions, and a plan is *optimal* if there is no shorter plan.

our objective is to find *suboptimal* (*not necessarily have to be the best possible plans, but they must be feasible and reasonably good.*) plans for collections of instances over fixed domains denoted as QD or simply as Q .

NEXT UP

sketches

Classical Planning



Sketches



Width



from Lipovetzky and Geffner (2012), Bonet and Geffner (2021), and Drexler, Seipp, and Geffner (2021).

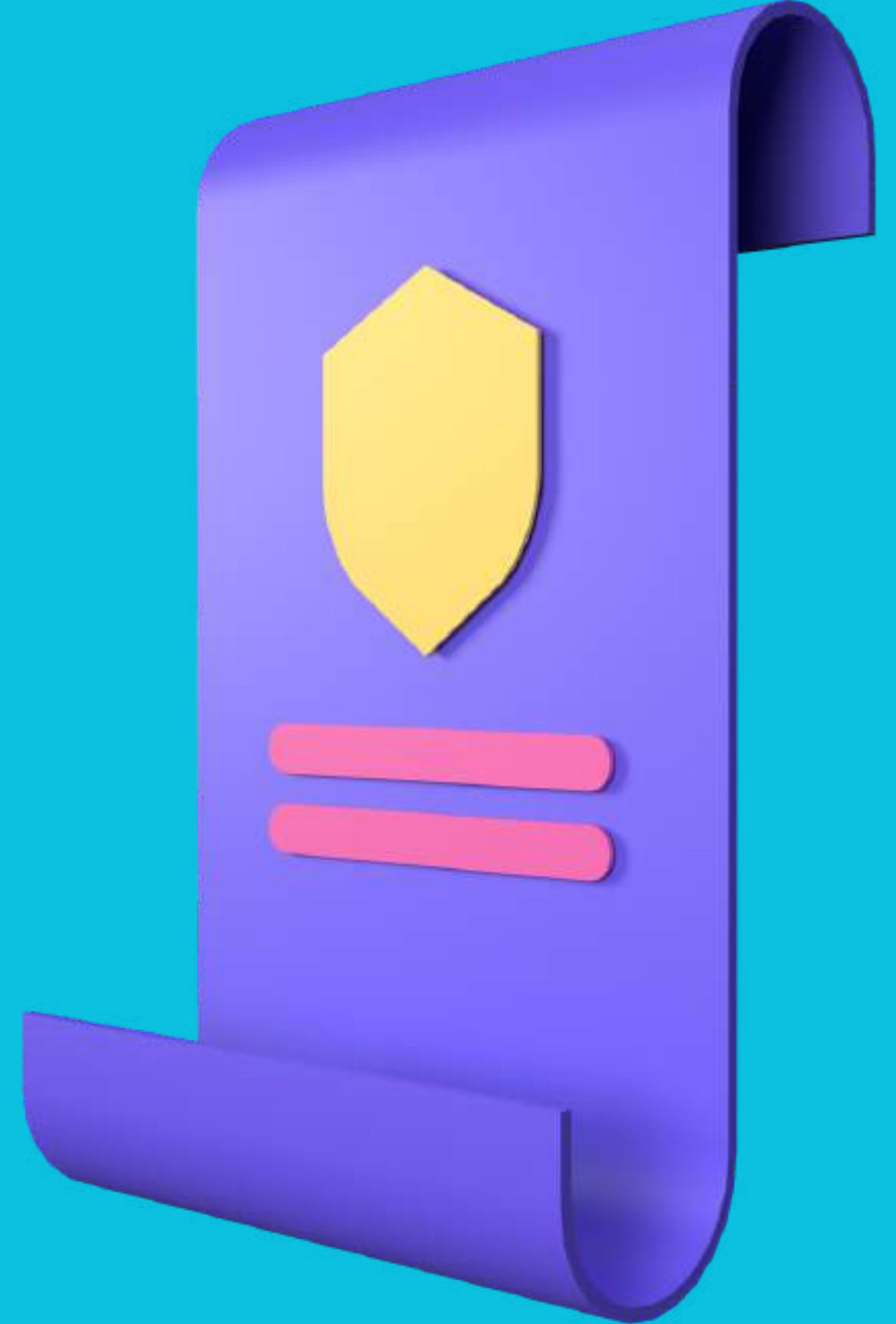


INTRODUCTION - SKETCHES:

An alternative, simpler language for representing problem structure explicitly has been introduced recently in the form of

sketches

Sketches have been introduced as a general language for representing the subgoal structure of instances drawn from the same domain.



SKETCHES:



Sketches are collections of rules of the form $C \rightarrow E$ defined over a given set of Boolean and numerical (integers ≥ 0) domain features Φ where C expresses Boolean conditions on the features, and E expresses qualitative *changes* in their values.

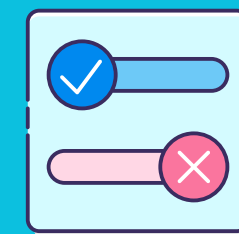
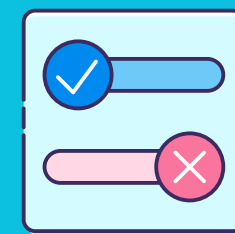
sketches can split problems into subproblems of bounded width.

Each sketch rule defines a subproblem: going from a state that satisfies C to a state that achieves the change expressed by E or a goal state.

$\Phi := \{ \text{AGE}, \text{HIGHT}, \text{SINGLE}, \text{ALLERGIES} \}$

3 4

1 6 5



FEATURE VALUATION

a feature is a function of the state over a class of problems Q .

The features considered in the language of sketches are boolean or numerical non negative integers.

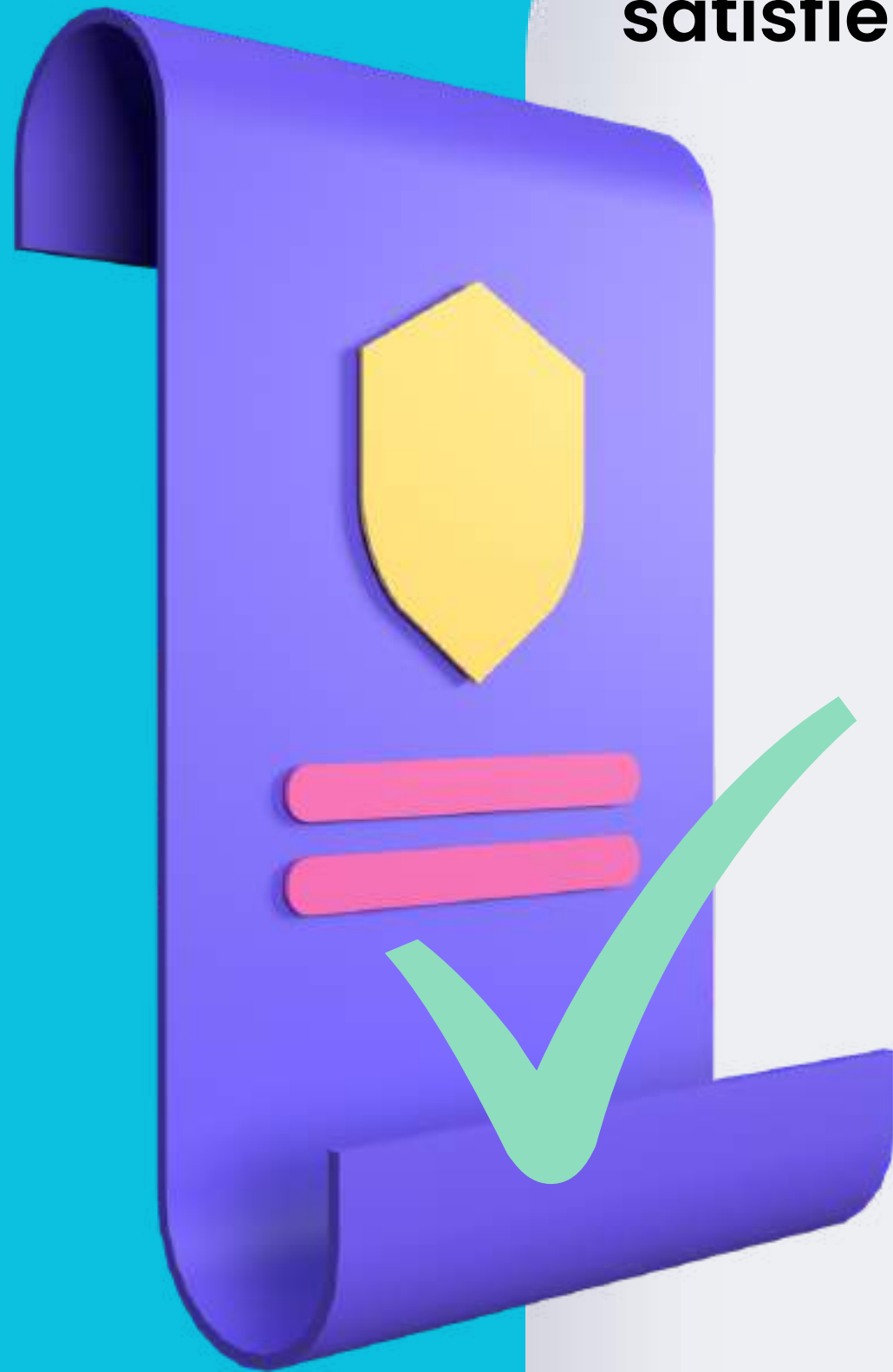
For a set of features: $\Phi = \{f_1, f_2, \dots, f_n\}$ and a state s of some instance P in Q , we denote the feature valuation determined by the state s as $f(s) := (f_1(s), f_2(s), \dots, f_n(s))$, and arbitrary feature valuations f and f' .



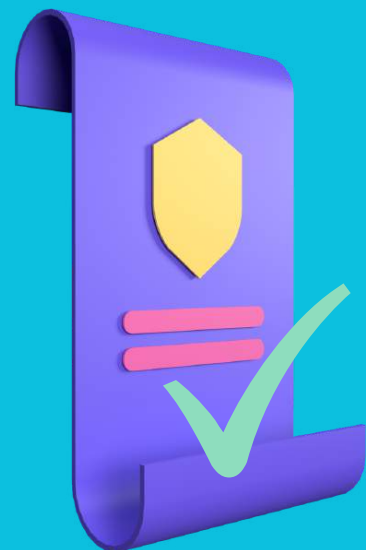
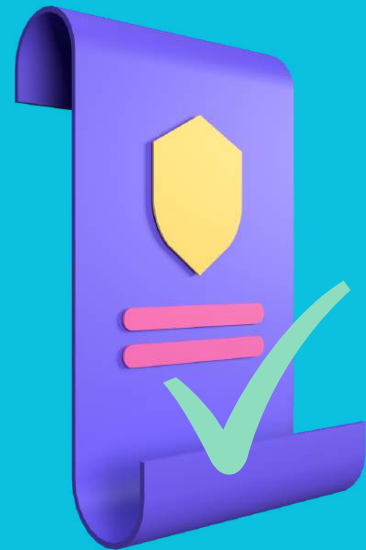
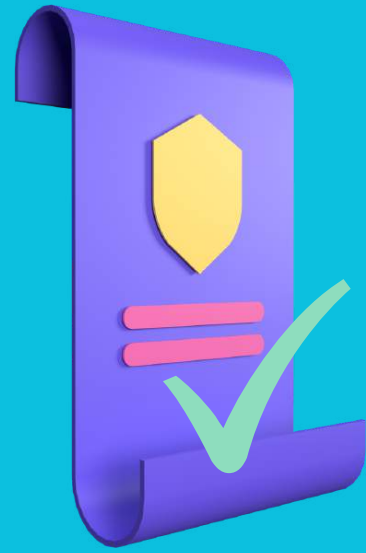
SATISFYING A SKETCH RULE

A pair of feature valuations of two states $(f(s), f(s'))$ referred to as (f, f') satisfy a sketch rule $C \rightarrow E$ iff:

- 1) C is true in f
- 2) the Boolean effects p ($\neg p$) in E are true in f'
- 3) the numerical effects are satisfied by the pair (f, f') .
(if $f \ n \downarrow$ in E , then the value of n in f' is smaller than in f)
- 4) features that do not occur in E have the same value in f and f' .



SKETCHS:

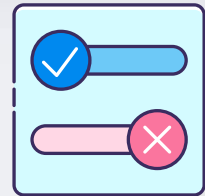


A sketch is a collection of sketch rules that establishes a “preference ordering” ‘ $<$ ’ over feature valuations where $f' < f$ if the pair of feature valuations (f, f') satisfies a rule.

If the sketch is *terminating* (will end , no loops like in voting candidates), then this preference order is a strict partial order: irreflexive and transitive.

Checking termination requires time that is exponential in the number of features.

SIMPLE EXAMPLE



3 4

6

$\Phi = \{\text{Hungry}, \text{cals in body}, \text{Time till food ready}\}$

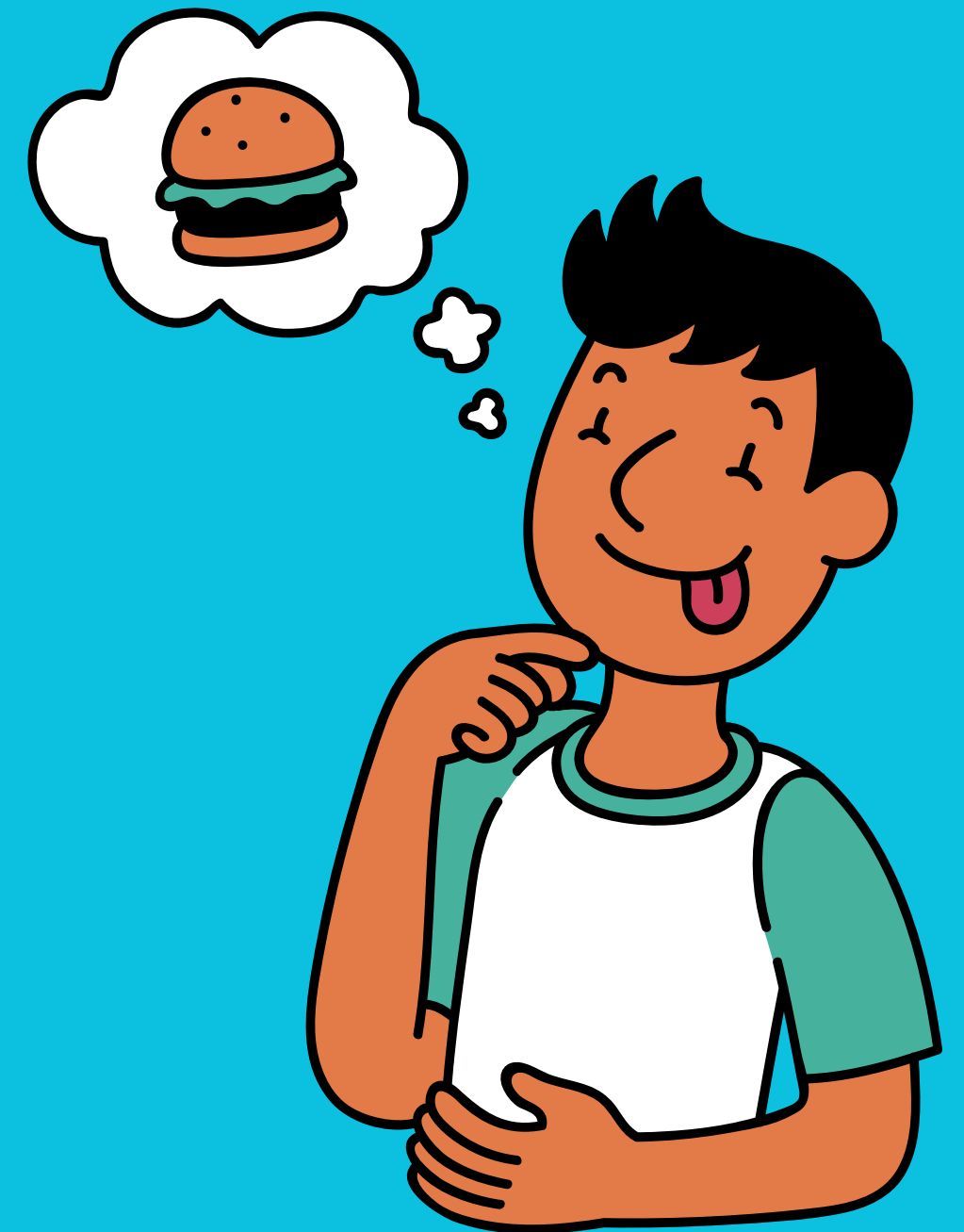
$\{\neg H, c > 0\} \rightarrow \{c \downarrow, T?\};$

$\{H, c \geq 0, T > 0\text{min}\} \rightarrow \{T \downarrow\};$

$\{T = 0\text{min}\} \rightarrow \{\neg H\};$

\neg - NOT, $>$ - gt, $<$ - lt ect... in regards to set C

\uparrow - value changes up, \downarrow - value changes down, $?$ - value can change up or down or switch boolean value.
in regards to set E





POWER OF SKETCHES



Each sketch rule defines a subproblem: going from a state that satisfies C to a state that achieves the change expressed by E or a goal state. Sketches can encode simple goal serializations, general policies, or decompositions of bounded width that can be solved greedily, in polynomial time, by the SIWR variant of the SIW algorithm.

The language of sketches is powerful, as sketches can encode everything from simple goal serializations to full general policies as well as split problems into subproblems of bounded *width*.

the language of general policies is the language of sketches but with a slightly different semantics where the subgoal states s' to be reached from a state s are restricted to be one step away from s

LIMITATIONS OF SKETCHS



Previous work has shown the computational value of sketches over benchmark domains that, while tractable, are challenging for domain-independent planners.

In this paper, the problem of learning sketches automatically is addressed given: a planning domain, some instances of the target class of problems, and the desired bound on the sketch width k , usually $k = 0, 1, 2$.

a logical formulation of the problem is presented, and yield a domain-independent planner that learns and exploits domain structure in a crisp and explicit form.

NEXT UP

Width

Classical Planning



Sketches



Width



from Lipovetzky and Geffner (2012), Bonet and Geffner (2021), and Drexler, Seipp, and Geffner (2021).



WIDTHS

the width of a sketch corresponds to the maximum number of rules that need to be applied to solve a subproblem in the sketch

0

Width-zero sketch: (General policy) where the subproblem of going from a non-goal state s to a state s' satisfying a rule $C \rightarrow E$ can always be done in one step

1

Width-1 sketch: where the problem can be broken down into subproblems that can be solved in a single step each. The solution can be obtained by solving these subproblems one after the other.

2





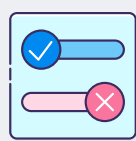
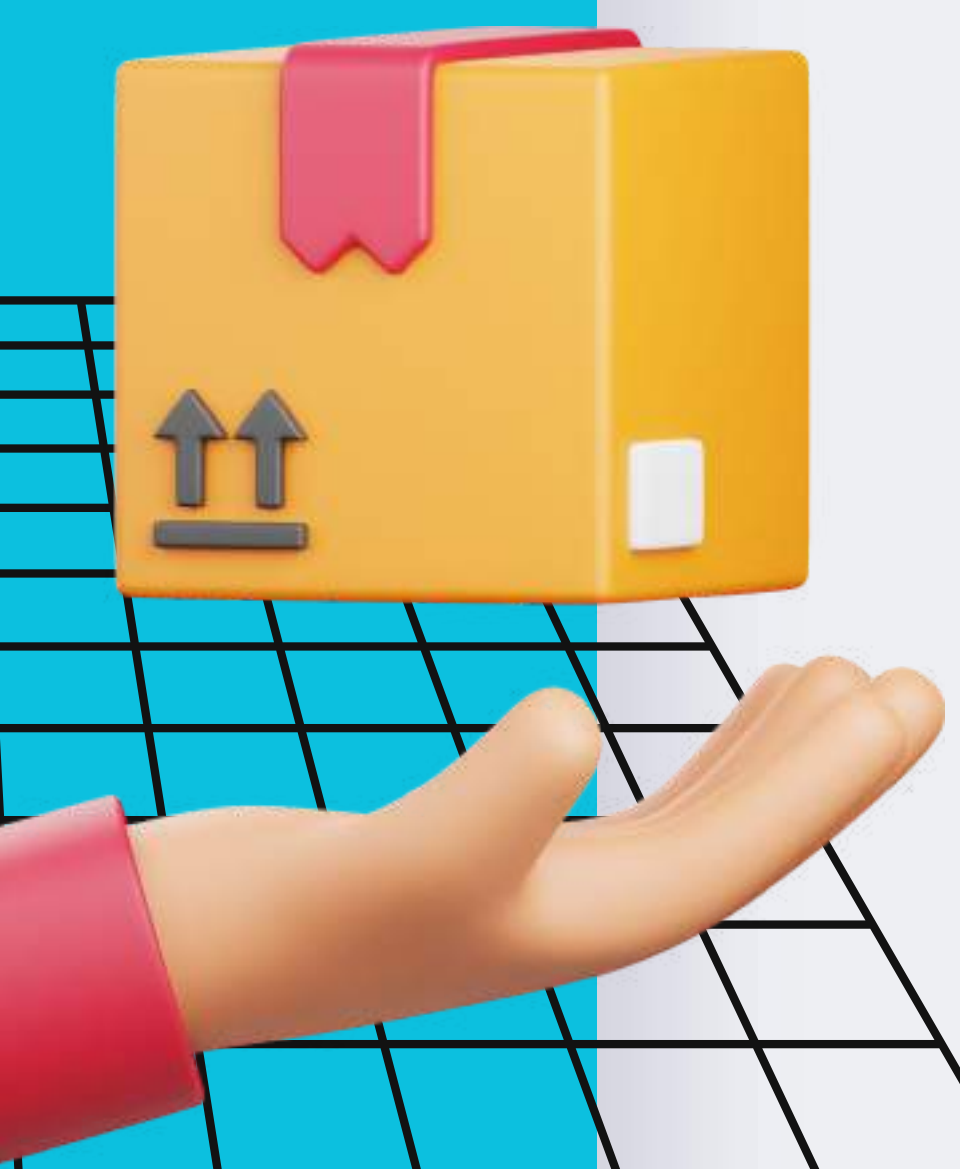
Width-2 or higher: The problem requires subproblems that may take multiple steps to solve. The solution involves solving these subproblems in a specific order.

EXAMPLE FROM PAPER

a simple domain called Delivery

an agent moves in a grid to pick up packages and deliver them to a target cell, one by one.

A general policy π for this class of instances can be expressed in terms of the set of features:


$$\Phi = \{H, p, t, n\}$$

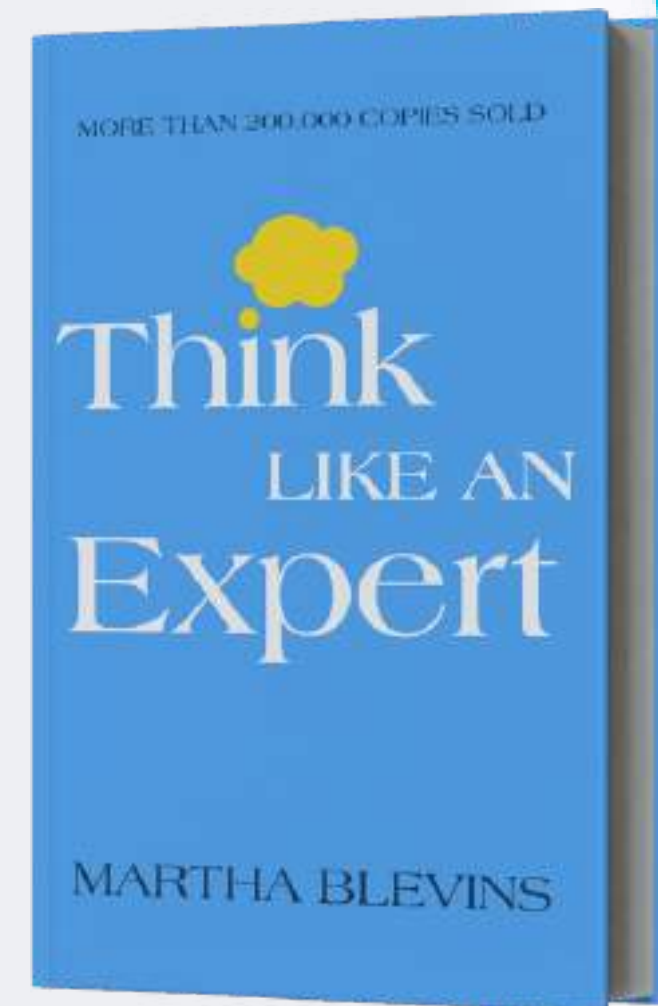
that express: “holding a package”, “distance to the nearest package”, “distance to the target cell”, and “number of undelivered packages”

SIDE NOTE:

*A domain-independent method for generating a large pool of features from the domain predicates that include these four is given by Bonet, Franc`es, and Geffner (2019).

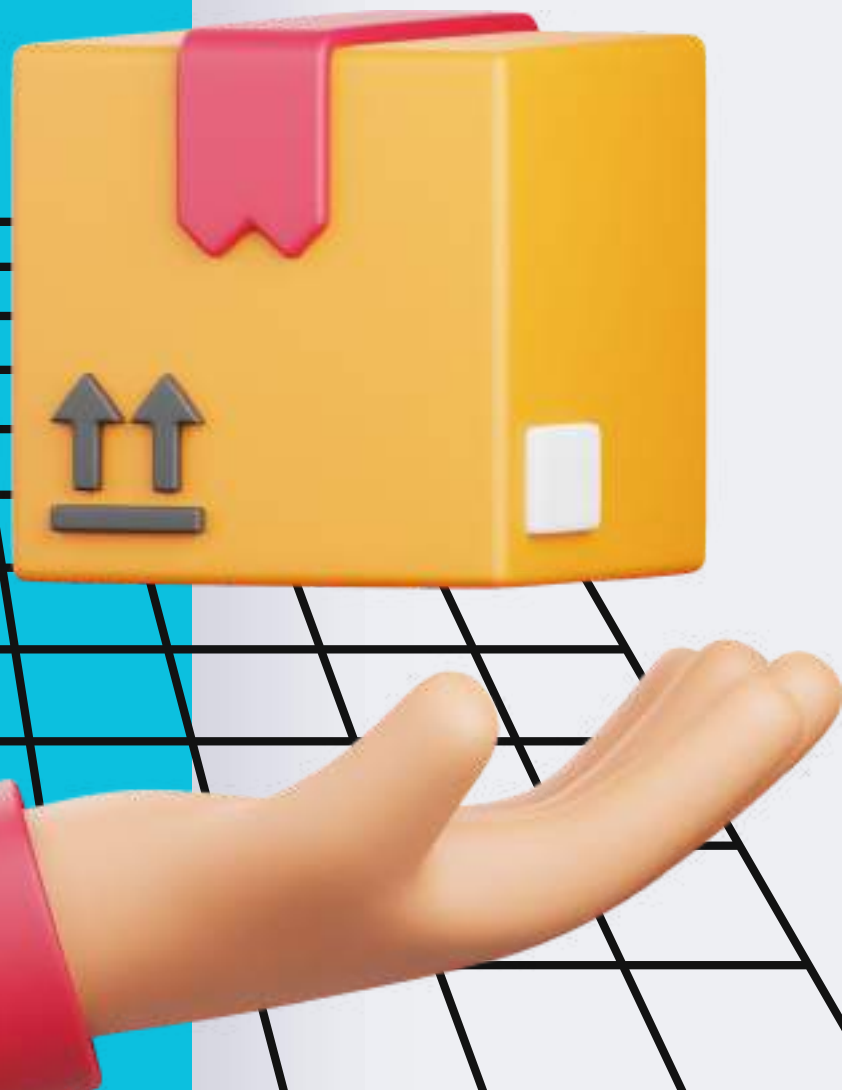
understanding what is important and what is irrelevant in regards to domain features usually requires domain experts.

it seems to imply that for this domain (Delivery) a method was generated for extracting a large pool of features that include these four features.



DELIVERY

Provided with the features in $\Phi = (\{H, p, t, n\})$, a general policy for the whole class QD of Delivery problems (any grid size, any number of packages in any location, and any location of the agent or the target) is given by the sketch:



$\{\neg H, p > 0\} \rightarrow \{p \downarrow, t?\}$; go to nearest pkg
 $\{\neg H, p = 0\} \rightarrow \{H\}$; pick it up
 $\{H, t > 0\} \rightarrow \{t \downarrow\}$; go to target
 $\{H, n > 0, t = 0\} \rightarrow \{H?, n \downarrow, p?\}$; deliver pkg

DELIVERY

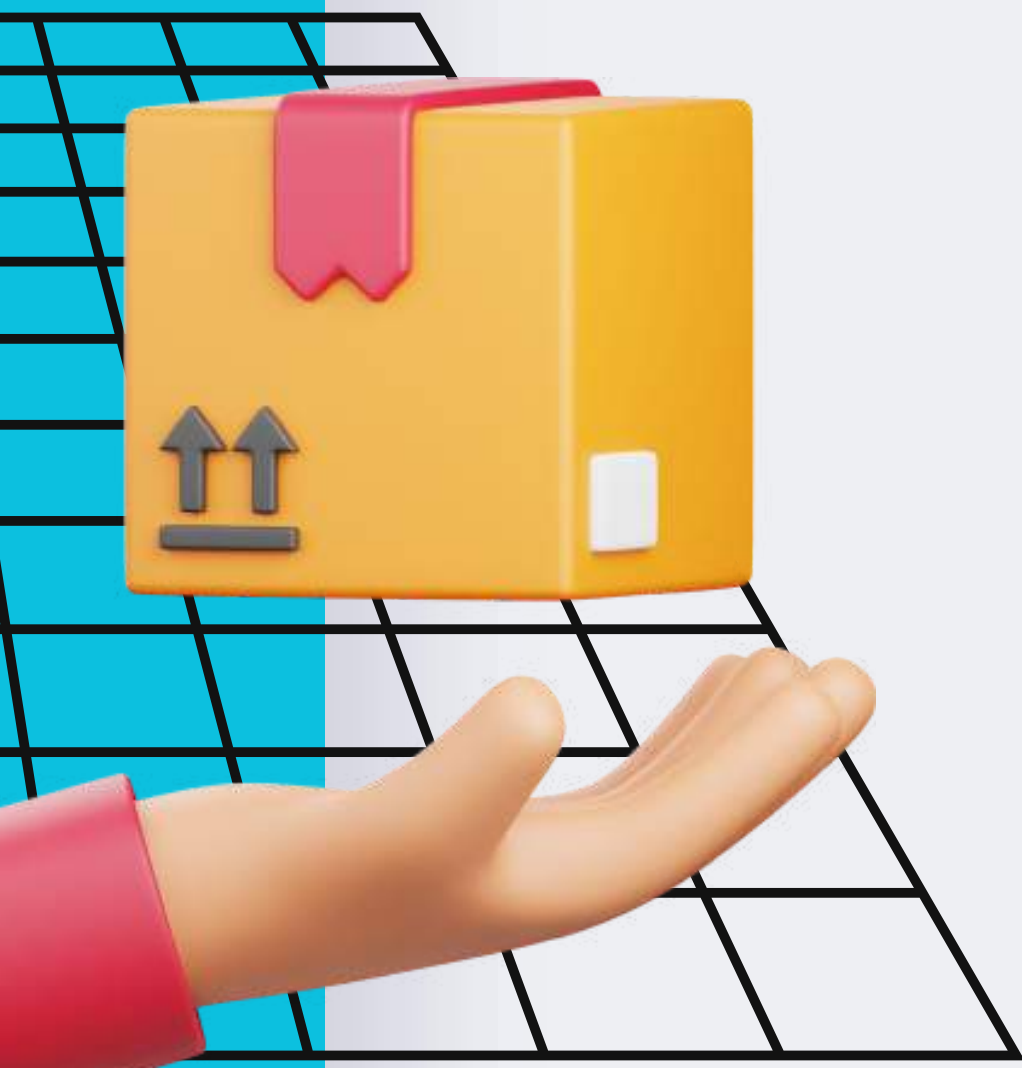
$\{\neg H, p > 0\} \rightarrow \{p \downarrow, t?\}$; go to nearest pkg

$\{\neg H, p = 0\} \rightarrow \{H\}$; pick it up

$\{H, t > 0\} \rightarrow \{t \downarrow\}$; go to target

$\{H, n > 0, t = 0\} \rightarrow \{H?, n \downarrow, p?\}$; deliver pkg

The rules say to perform any action that decreases the distance to the nearest package ($p \downarrow$), no matter the effect on the distance to target ($t?$), when not holding a package ($\neg H$); to pick a package when possible, making H true; to go to the target cell when holding a package, decrementing t ; and to drop the package at the target, decrementing n , making H false, and affecting p . Expressions $p?$ and $t?$ mean that p and t can change in any way, while no mention of a feature in the effect of a rule means that the value of the feature must not change.



DELIVERY - 0 WIDTH

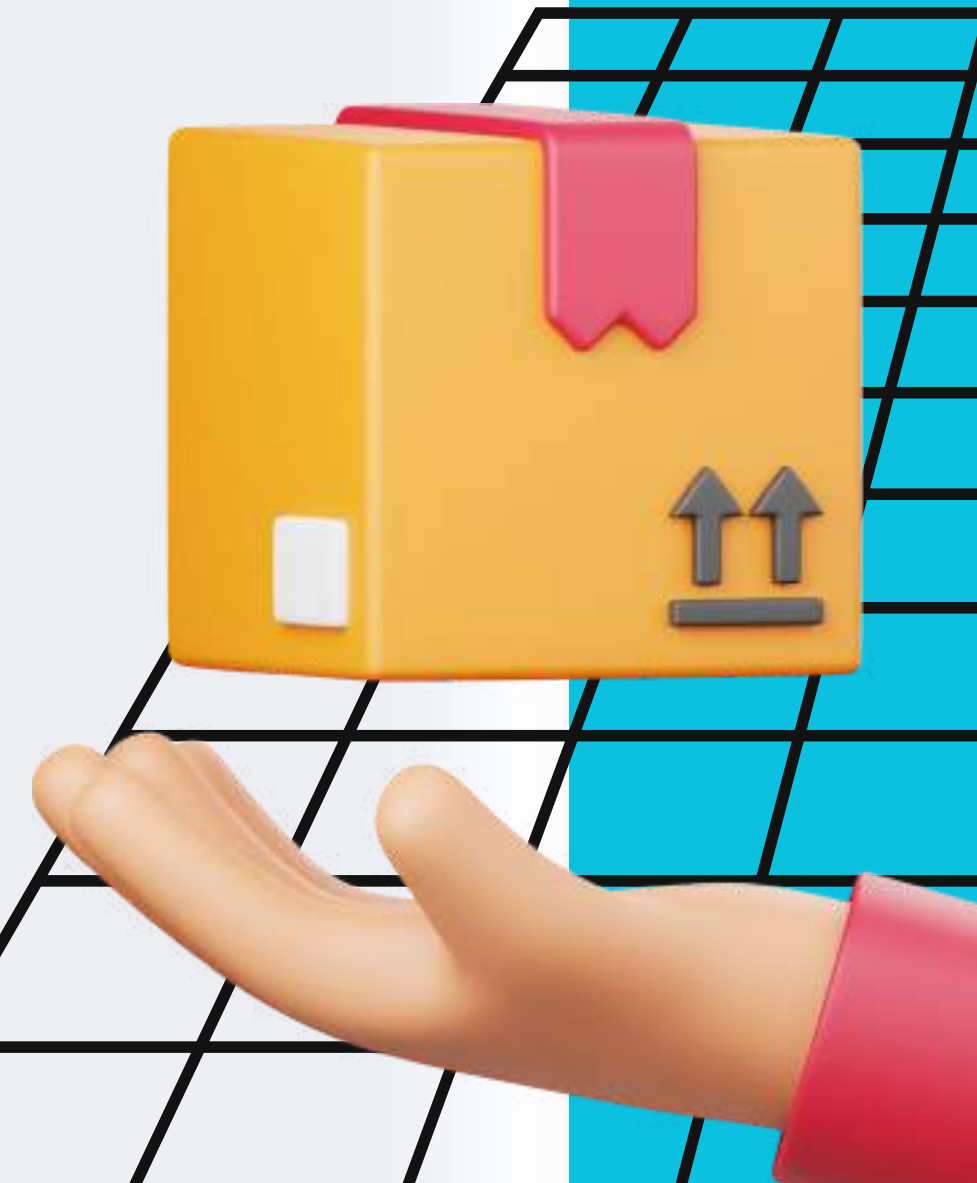
$\{\neg H, p > 0\} \rightarrow \{p \downarrow, t?\}$; go to nearest pkg
 $\{\neg H, p = 0\} \rightarrow \{H\}$; pick it up
 $\{H, t > 0\} \rightarrow \{t \downarrow\}$; go to target
 $\{H, n > 0, t = 0\} \rightarrow \{H?, n \downarrow, p?\}$; deliver pkg

0

width-zero sketch

One can show that the general policy above solves any instance of Delivery, or alternatively, that it represents a width-zero sketch, where the subproblem of going from a non-goal state s to a state s' satisfying a rule $C \rightarrow E$ can always be done in one step, leading greedily to the goal ($n = 0$).

A pair of states (s, s') satisfies a rule $C \rightarrow E$ when the feature values in s satisfy the conditions in C , and the change in feature values from s to s' is compatible with the changes and no-changes expressed in E .



DELIVERY - 2 WIDTH

A width-2 sketch can be defined instead by means of a single rule and a single

feature $\Phi = \{n\}$:

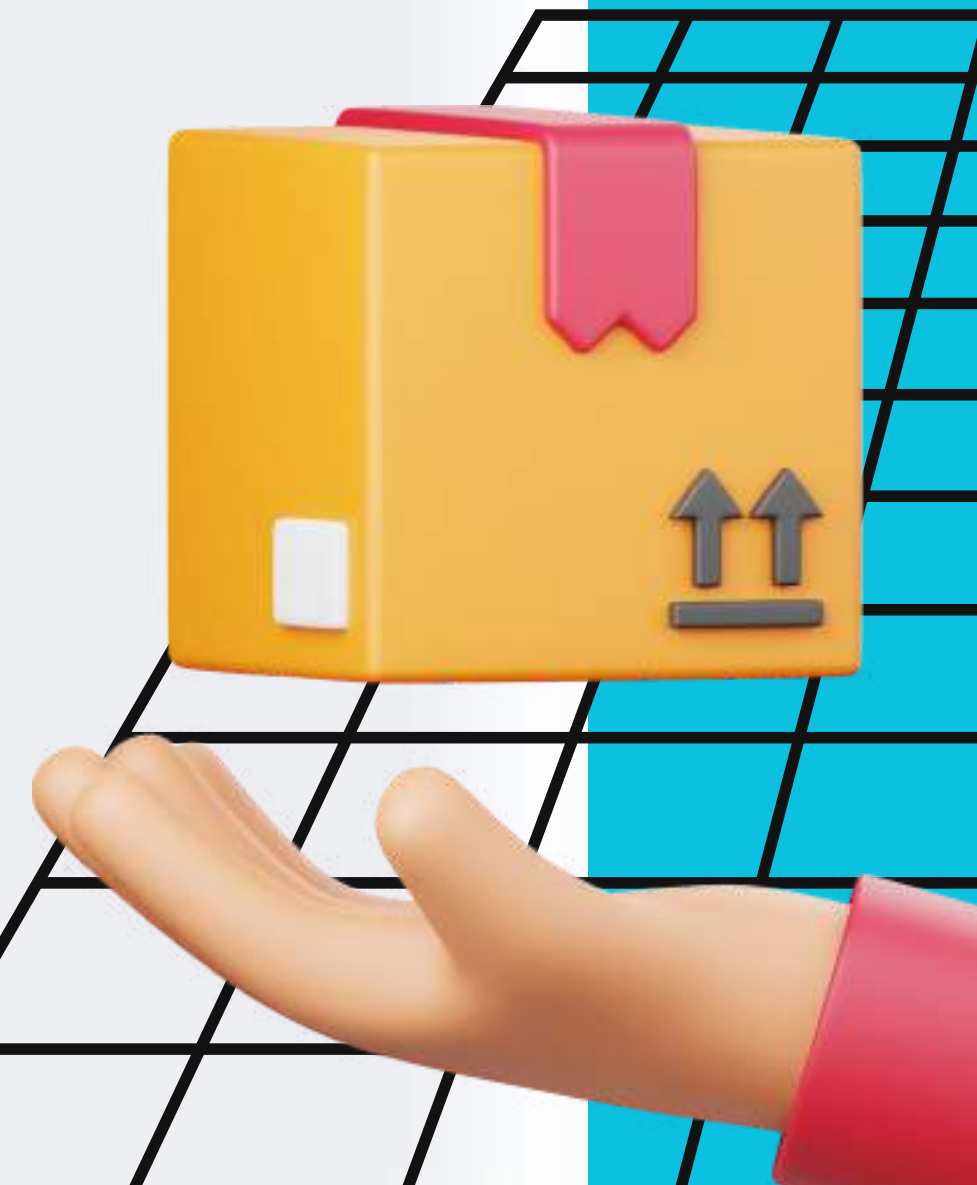
$\{n > 0\} \rightarrow \{n \downarrow\}$; deliver pkg

2

width-Two sketch

The subproblem of going from a state s where $n(s) > 0$ holds, to a state s' for which $n(s') < n(s)$ holds, has width bounded by 2.

As these subproblems, in the worst case, involve moving to the location of a package, picking it up *and* moving to the target, and dropping it.



DELIVERY 1-WIDTH

Halfway between the width-0 sketch represented by general policies, and the width-2 sketch above, is the width-1 sketch defined with two rules and two features:

$$\Phi = \{H, n\}$$

$\{\neg H\} \rightarrow \{H\}$; pick pkg

$\{H, n > 0\} \rightarrow \{H?, n \downarrow\}$; deliver pkg

The first rule captures the subproblem of getting hold of a package, which may involve moving to the package and picking it up, with width 1;

the second rule captures the subproblem of delivering the package being held, which may involve moving to the target and dropping it there, also with width 1.

Since every non-goal state is covered by one rule or the other, the width of the sketch is 1, which means that any instance of Delivery can be solved greedily by solving subproblems of width no greater than 1 in linear time.

1

width-One sketch



WIDTH

in some sense width measures how "complicated" a solution to a planning problem may be.

Problems with lower width are generally easier to solve than problems with higher width because they require fewer actions to be executed simultaneously and can therefore be solved more efficiently.

The width k of the sketch relative to a class of problems bounds the width of the resulting sub-problems that can be solved greedily in time and space that are exponential in k .

in the Delivery problem, the width-1 sketch can be solved greedily in linear time because the sub-problems involved have a maximum width of 1. However, the width-2 sketch would require sub-problems with a maximum width of 2 to be solved greedily, which would require more time and space.





WIDTH-BASED SEARCH

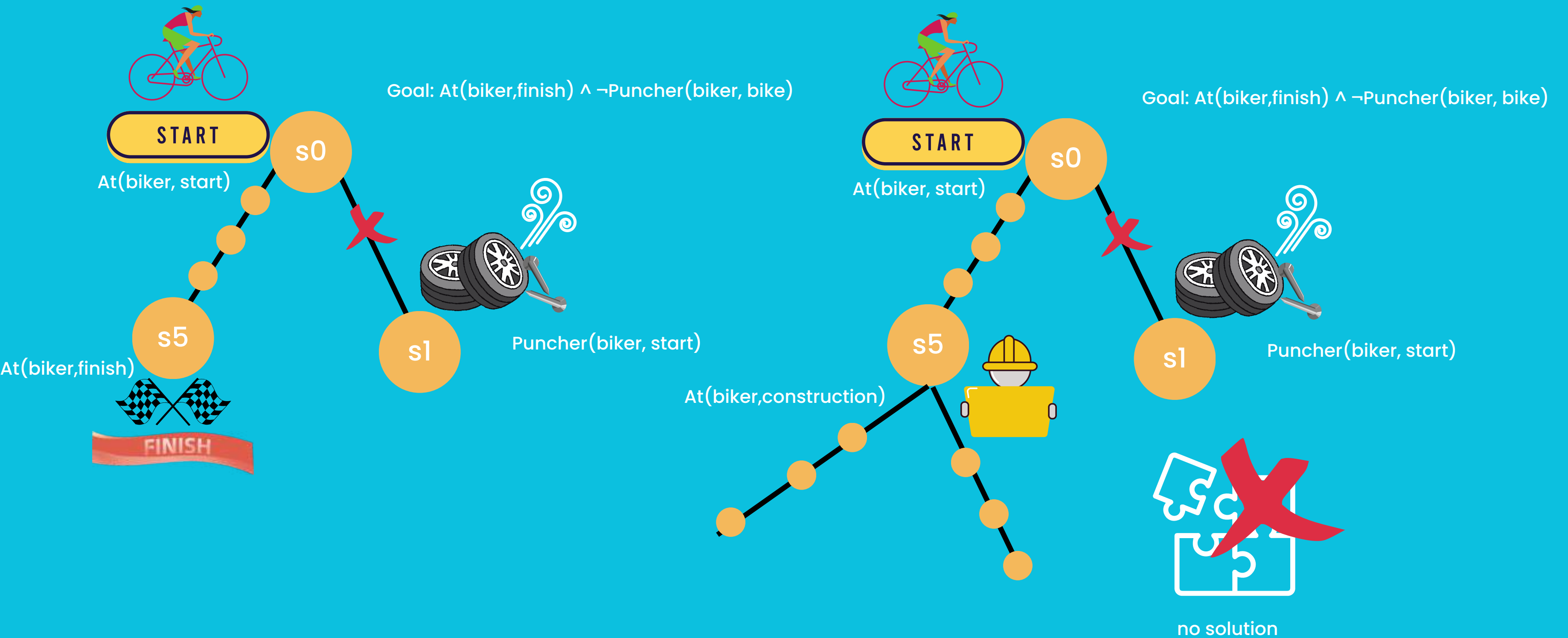
The simplest width-based search method for solving a planning problem P is $IW(1)$. It is a standard breadth-first search (BFS) in the rooted directed graph associated with the state model $S(P)$ with one modification:

$IW(1)$ starts by generating all possible successor states of the initial state s_0 . It then examines each successor state and prunes any that do not make an atom true for the first time in the search. This means that if a newly generated state does not satisfy any new goal condition, it is pruned from further consideration.

The algorithm then continues to generate successor states from the remaining states until it finds a state that satisfies all goal conditions specified in P . If no such state can be found, then $IW(1)$ concludes that P is unsolvable.

$IW(1)$ is an efficient width-based search method for solving planning problems because it prunes states that do not satisfy any new goal conditions early on in the search process. This reduces the number of states that need to be explored and can lead to faster convergence to a solution.

IW(1)





IW(K) SEARCH

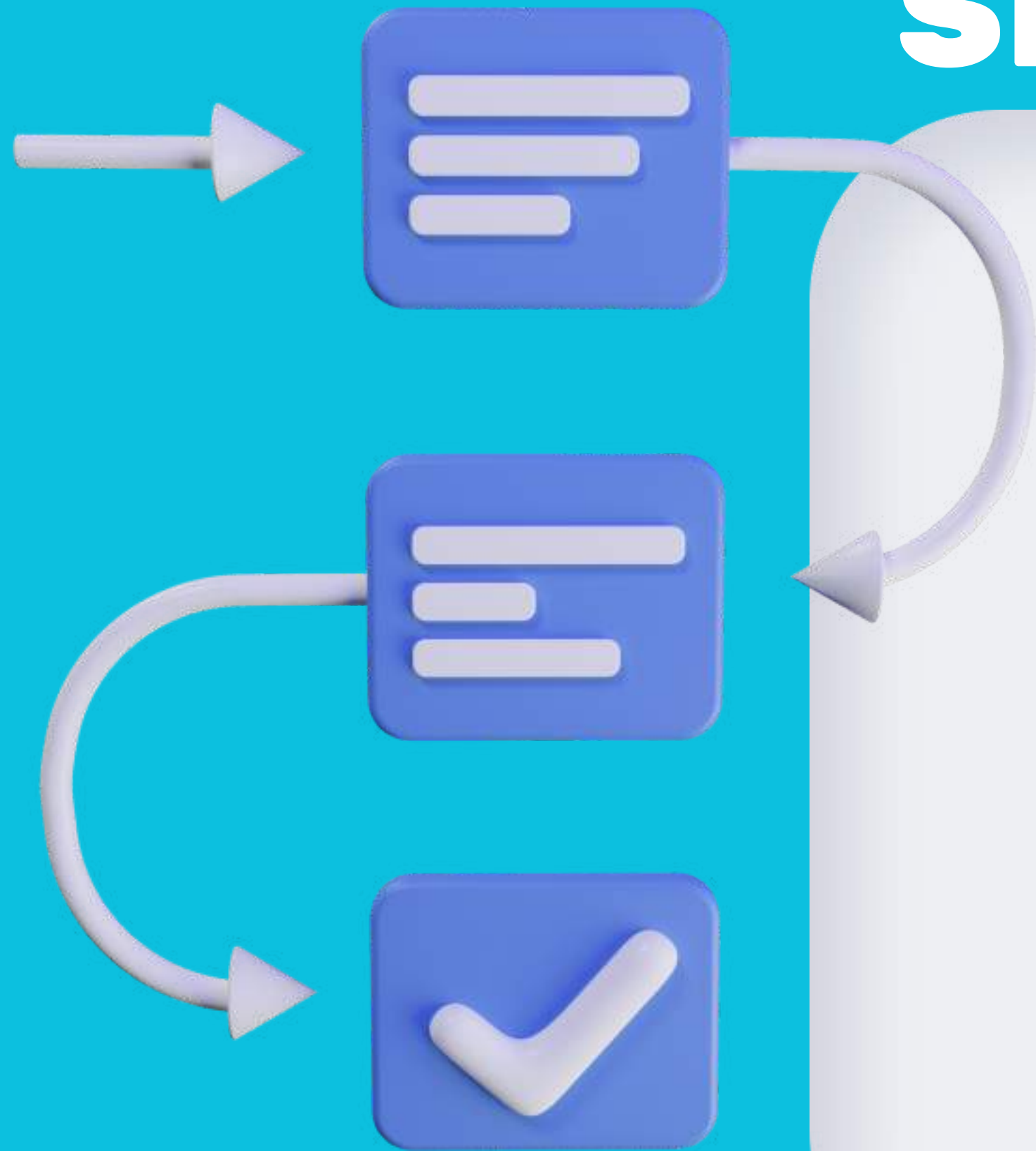
IW(k) search method is a width-based search algorithm used to solve planning problems. The algorithm is similar to IW(1), but with one key difference: it prunes a state if a newly generated state does not make a collection of up to k atoms true for the first time.

IW(k) starts by generating all possible successor states of the initial state s_0 . It then examines each successor state and prunes any that do not make a collection of up to k atoms true for the first time. This means that if a newly generated state does not satisfy any new goal condition involving up to k atoms, it is pruned from further consideration.

The algorithm then continues to generate successor states from the remaining states until it finds a state that satisfies all goal conditions specified in P. If no such state can be found, then IW(k) concludes that P is unsolvable.

IW(k) is an efficient width-based search method for solving planning problems because it prunes states that do not satisfy any new goal conditions involving up to k atoms early on in the search process. This reduces the number of states that need to be explored and can lead to faster convergence to a solution. However, as k increases, the number of states that need to be explored may also increase, which can lead to longer search times and increased memory usage.

SIW ALGORITHM



the SIW algorithm (Lipovetzky and Geffner 2012)

starts at the initial state $s = s_0$ of P , and performs an IW search from s to find a shortest path to a state s' such that $\#g(s') < \#g(s)$

where $\#g(s)$ counts the number of unsatisfied top-level goals of P in state s . If s' is not a goal state, s is set to s' and the loop repeats.



WIDTH OF CLASSICAL PLANNING

The width $w(P)$ of a classical planning problem P is the minimum k for which there exists a sequence t_0, t_1, \dots, t_m of atom tuples t_i (where $0 \leq i \leq m$) from P , each consisting of at most k atoms, such that:

1. t_0 is true in the initial state s_0 of P
2. any optimal plan for t_i can be extended into an optimal plan for t_{i+1} by adding a single action, $i = 1, \dots, m-1$
3. if π is an optimal plan for t_m , π is an optimal plan for P .

If a problem P is unsolvable, $w(P)$ is set to the number of variables (box1, box2 ... boxc, table, pickup, putdown ect in blocks world domain) in P , and if P is solvable in at most one step, $w(P)$ is set to 0 (Bonet and Geffner 2021). Chains of tuples $\theta = (t_0, t_1, \dots, t_m)$ that comply with **conditions 1–2** are called *admissible*, and the size of θ is the size $|t_i|$ of the largest tuple in the chain.



WIDTH OF CLASSICAL PLANNING

The width $w(P)$ is then the minimum size of an admissible chain for P that is also optimal (**condition 3**). Furthermore, the width of a conjunction of atoms T (or arbitrary set of states S' in P) is the width of a problem P' that is like P but with the goal T (resp. S').

For problems with conjunctive goals, the SIW algorithm (Lipovetzky and Geffner 2012) is introduced as we saw before.

The $IW(k)$ algorithm expands up to N^k nodes, generates up to bN^k nodes, and runs in time and space $O(bN^{(2k-1)})$ and $O(bN^k)$. where N refers to the number of atoms in the planning problem P , b is the maximum number of successors of any state in the search space and k is a parameter that specifies the maximum width of a planning problem that can be solved optimally by the $IW(k)$ algorithm.

$IW(k)$ is guaranteed to solve P optimally if $w(P) \leq k$. If the width of P is not known, the IW algorithm can be run instead which calls $IW(k)$ iteratively for $k = 0, 1, \dots, N$ until the problem is solved, or found to be unsolvable.



BACK TO SKETCHS

A sketch is a collection of sketch rules that establishes a “preference ordering” ‘ $<$ ’ over feature valuations where $f' < f$ if the pair of feature valuations (f, f') satisfies a rule.

This preference ordering, is used to determine the order in which subgoals are to be pursued during the search for a solution. If a sketch is terminating, then the preference order is a strict partial order, which means that it is irreflexive (no feature can be less preferred than itself) and transitive (if f_1 is less preferred than f_2 , and f_2 is less preferred than f_3 , then f_1 is less preferred than f_3).

Checking termination requires time that is exponential in the number of features (Bonet and Geffner 2021).

Checking termination is important because if the sketch is not terminating, the resulting preference order may not be a strict partial order, which can cause the planner to explore unnecessary or even infinite search spaces. if the sketch is not terminating, the planner may not converge to a solution or may converge to a suboptimal solution.



BACK TO SKETCHS

in the paper the researchers do not use these orderings explicitly but the associated problem decompositions.

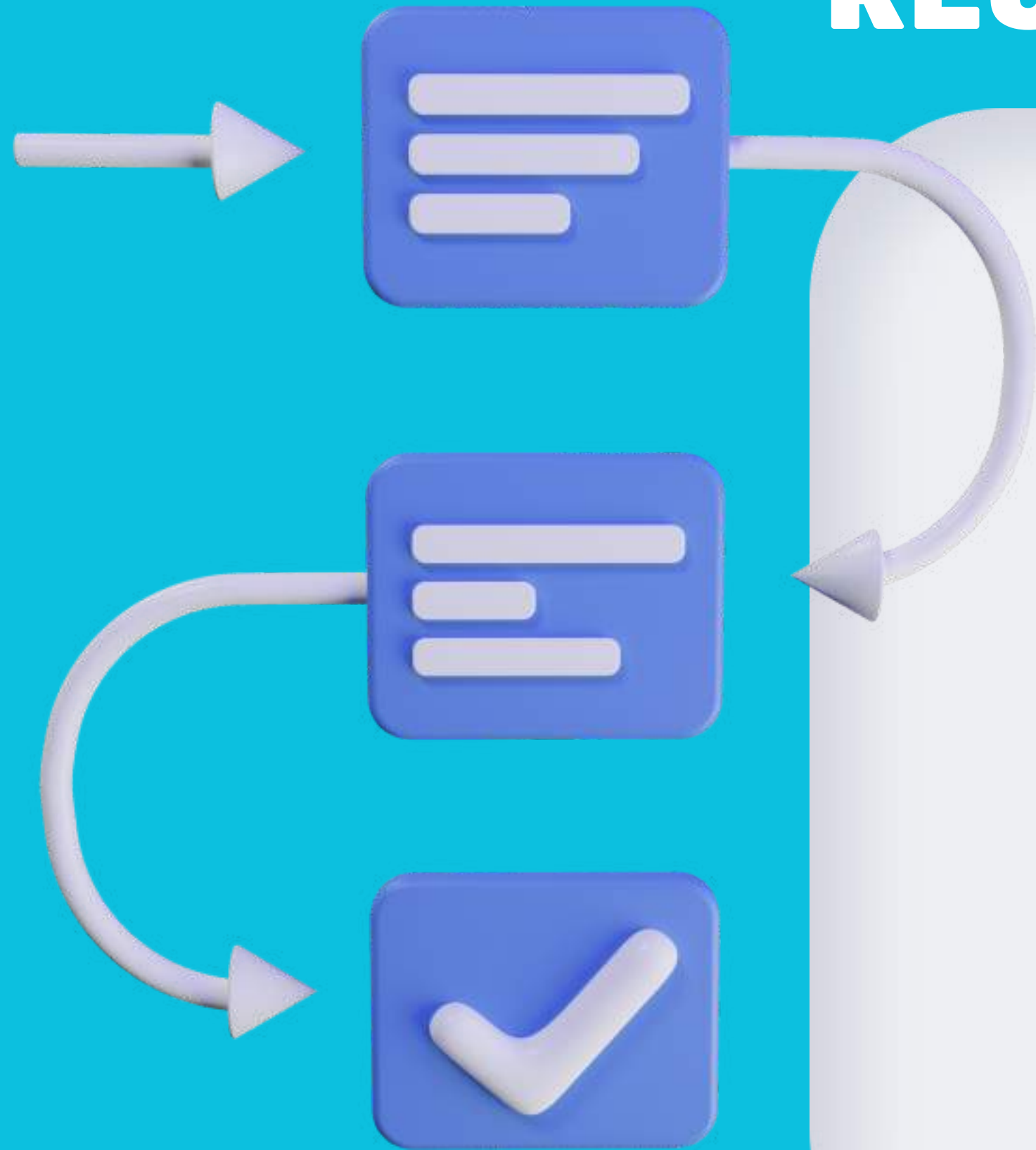
decomposition divides the original problem into subproblems that are easier to solve. The problem decomposition generates a set of subproblems that collectively cover the entire feature space.

The set of subgoal states $GR(s)$ associated with a sketch R in a state s of a problem $P \in Q$, is the set of states s' that comprise the goal states of P along with those with feature valuation $f(s')$ such that the pair $(f(s), f(s'))$ satisfies a rule in R . $GR(s)$ contains all the states that could potentially help in achieving the goal, based on the rules defined in the sketch R .

The set of states s' in $GR(s)$ that are closest to s is denoted as $G * R(s)$. This set represents the most promising subgoals to pursue next in the search for a solution, based on the preference ordering defined by the sketch R .

The researchers use the problem decomposition induced by the preference orderings to guide their search algorithm towards a solution. They focus their search on subproblems that are more likely to contain a solution based on their associated preference orderings. This allows them to search more efficiently and to avoid searching parts of the feature space that are unlikely to contain a solution.

RECAP: SIW ALGORITHM

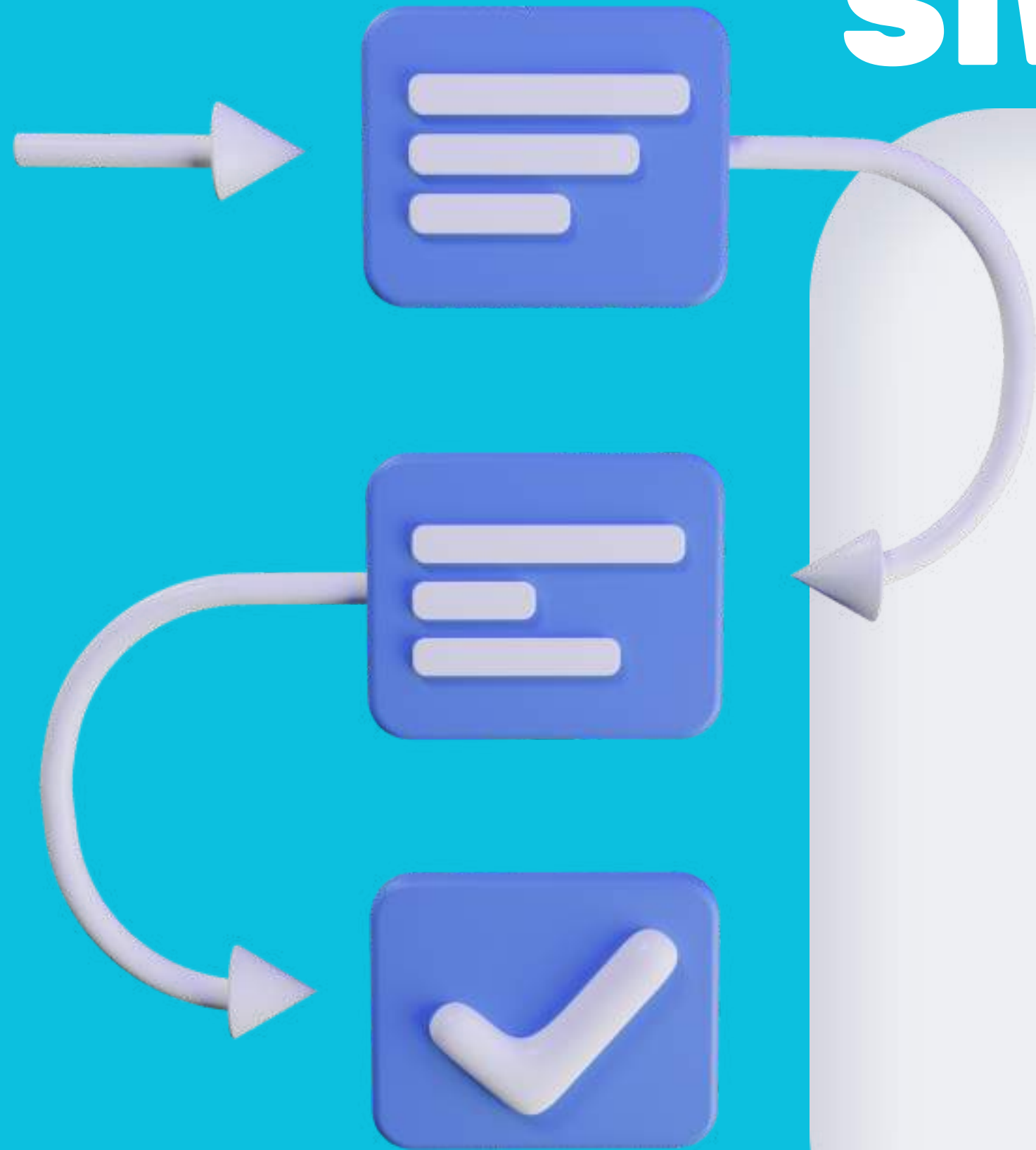


the SIW algorithm (Lipovetzky and Geffner 2012)

starts at the initial state $s = s_0$ of P , and performs an IW search from s to find a shortest path to a state s' such that $\#g(s') < \#g(s)$

where $\#g(s)$ counts the number of unsatisfied top-level goals of P in state s . If s' is not a goal state, s is set to s' and the loop repeats.

SIWR ALGORITHM



The SIWR algorithm is a variant of SIW that uses a given sketch R for solving problems P in Q .

SIWR starts at the initial state $s = s_0$ of P and then runs an IW search to find a state s' in $GR(s)$.

If s' is not a goal state, then s is set to s' , and the loop repeats until a goal state is reached.

The SIWR(k) algorithm is like SIWR but calls the procedure IW(k) internally, not IW.



SKETCH WIDTH

For bounding the complexity of these algorithms, let us assume without loss of generality that the class of problems Q is closed in the sense that if P belongs to Q so do the problems P' that are like P but with initial states that are reachable in P and which are not dead-ends.

This assumption essentially means that we only consider problems in Q that are "**well-behaved**" in the sense that they have no unreachable states or dead-ends. this assumption allows us to develop more efficient search algorithms that can be applied to a broader range of "**well-behaved**" problems.

Then the width of the sketch R over Q can be defined as follows (Bonet and Geffner 2021):

The width of sketch R over a **closed class of problems** Q is $w_R(Q) = \max_{P \in Q} w(P')$ where P' is P but with goal states $G * R(s)$ and s is the initial state of both, provided that $G * R(s)$ does not contain dead- end states.



SKETCH WIDTH

The width of sketch R over a **closed class of problems** Q is $w_R(Q) = \max_{P \in Q} w(P')$ where P' is P but with goal states $G * R(s)$ and s is the initial state of both, provided that $G * R(s)$ does not contain dead-end states.

P' is a variant of P where the goal states are defined using the sketch R .

$GR(s)$ contains all the states that could potentially help in achieving the goal, based on the rules defined in the sketch R . The set of states s' in $GR(s)$ that are closest to s is denoted as $G * R(s)$. If a state is a dead-end state, it is not included in the goal states.

The width of the sketch R over Q is then defined as the maximum width among all computed values $w(P')$ of each problem P in Q .

SIMPLE EXAMPLE

Suppose we have a **classical planning** problem where a robot needs to assemble a car. The goal state is for the car to be fully assembled.

Consider the classical planning problem build car $S(P) = (S, s_0, G, Act, A, f)$, where:

- S : The set of states, which consists of all possible configurations of the car parts and tools in the workspace.
- s_0 : The initial state, which is the empty workspace
- G : The set of goal states, which consists of all possible configurations of the completed car.
- Act : The set of actions, which consists of all possible actions that can be taken in the workspace, such as picking up a car part or using a tool.
- A : The set of preconditions for each action in regards to the state.
- f : The mapping function



SIMPLE EXAMPLE

Using this classical planning problem definition, we can then define the sketch R as a set of rules or subgoals that can help guide the planning process. For example, a subgoal in the sketch R might be to "Attach wheels to the car body" or "Install the engine in the car".

$GR(s)$ represents the set of goal states that can be reached from the current state s by following the rules in the sketch R . For example, if the current state is one where the car body is assembled but the wheels have not yet been attached, the $GR(s)$ would include all possible configurations of the car with wheels attached, based on the rules in the sketch R .

The set of states s' in $GR(s)$ that are closest to s is denoted as $G \ast R(s)$. If a state is a dead-end state, it is not included in the goal states.



SKETCH WIDTH

If the sketch width is bounded, SIWR(k) solves the instances in Q in polynomial time:

Algorithm SIWR (k) is needed here instead of SIWR because the latter does not ensure that the subproblems P' are solved optimally. This is because IW(k') may solve problems of width k non-optimally if $k' < k$.

If $w_R(Q) \leq k$ and the sketch R is terminating, then SIWR (k) solves the instances in Q in $O(bN|\Phi| + 2k - 1)$ time and $O(bNk + N|\Phi| + k)$ space, where $|\Phi|$ is the number of features, N is the number of ground atoms and b is the maximum number of successors of any state in the search space and k is a parameter that specifies the maximum width of a planning problem that can be solved optimally by the IW(k) algorithm..

For these bounds, the features are assumed to be linear in N ; namely, they must have at most a linear number of values in each instance, all computable in linear time.

REMINDER – OUR GOAL

How Sketches can encode decompositions of bounded width that can be solved greedily, in polynomial time, by the SIWR variant of the SIW algorithm.



SKETCH SATISFYING WIDTH

The paper then defines a Sketch s-width (satisfying width) in order to bypass **condition 3** in regards to classical planning definition

1. t_0 is true in the initial state s_0 of P
2. any optimal plan for t_i can be extended into an optimal plan for t_{i+1} by adding a single action, $i = 1, \dots, n-1$
3. if π is an optimal plan for t_m , π is an optimal plan for P .

the standard notion of width requires optimal plans for subproblems, while the satisfying width only requires admissible plans.

This weaker notion of width allows for more efficient search algorithms that do not require optimal solutions at every step.

By bounding the satisfying width of a sketch, the search algorithm using that sketch can find a solution within a certain number of admissible steps, without requiring optimality at every step.



SKETCH SATISFYING WIDTH

The paper then addresses an additional constraint:

"Let us finally say that a sketch R is state acyclic in Q when there is no sequence of states s_1, \dots, s_n over a problem in Q , $s_{i+1} \in GR(s_i)$, such that $s_n = s_j$, $j < n$. Bonet and Geffner (2021) showed that termination implies feature acyclicity, and it is direct to show that feature acyclicity implies state acyclicity. we can then rephrase our definition of a sketch width as:"

If $ws R(Q) \leq k$ and the sketch R is acyclic in Q , then $SIWR(k)$ solves the instances in Q in $O(bN|\Phi|+2k-1)$ time and $O(bNk + N|\Phi|+k)$ space, where $|\Phi|$ is the number of features, N is the number of ground atoms, and b is the branching factor



NEXT UP

Learning sketches

Classical Planning



Sketches



Width



from Lipovetzky and Geffner (2012), Bonet and Geffner (2021), and Drexler, Seipp, and Geffner (2021).

LEARNING SKETCHES: FORMULATION

The paper then turns to the problem of learning sketches given a set of instances P the target class of problems Q and the desired bound k on sketch width.

roughly following the approach for learning general policies
(Bonet, Francès, and Geffner 2019; Francès, Bonet, and Geffner 2021)

by constructing a theory \mathcal{T} , bound k , a bound m on the number of sketch rules, and a finite pool of features F obtained from the domain predicates and a fixed grammar.



LEARNING SKETCHES: FORMULATION

The proof for the following was provided in a different paper:

terminating sketch R with rules $C_i \rightarrow E_i$, $i = 1, \dots, m$, over features $F \in \mathcal{F}$, has sketch width $w_R(P^*) \leq k$ over the closed class of problems P^* iff the theory $T_{k,m}(P, F)$ is satisfiable and has a model where the rules that are true are exactly those in R



EXPERIMENTS



they then learn sketches for several tractable classical planning domains from the International Planning Competition (IPC).

To learn a sketch, we use two Intel Xeon Gold 6130 CPUs, holding a total of 32 cores and 384 GiB of memory and set a time limit of seven days (wall-clock time). For evaluating the learned sketches, we limit time and memory by 30 minutes and 8 GiB. Our source code, benchmarks, and experimental data are

available online (Drexler, Seipp, and Geffner 2022).

INCREMENTAL LEARNING

"Instead of feeding all instances to the ASP * (Answer Set Programming is a declarative programming paradigm used for solving complex combinatorial problems. It allows expressing a problem in terms of rules and constraints, and finding all possible solutions that satisfy these rules and constraints. ASP implementations are software tools that execute and solve ASP programs.)

at the same time, we incrementally add only the smallest instance that the last learned sketch fails to solve. In detail, we order the training instances by the number of states in increasing order, breaking ties arbitrarily.

The first sketch is the empty sketch $R_0 = \emptyset$. Then, in each iteration $i = 1, 2, \dots$, we find the smallest instance I in the ordering which the sketch R_{i-1} obtained in the previous iteration results in a subproblem of width not bounded by k or in a cycle $s_1, \dots, s_n = s_1$ of subgoal states $s_{i+1} \in GR(s_i)$ with $i = 1, \dots, n - 1$."

These tests can be performed because the training instances are small. If such an I exists, we use it as the only training instance if it is the largest among all previous training instances, otherwise we add it and proceed with the next iteration. If no such I exists, the sketch solves all training instances, and we return the sketch.



SOME TARGET DOMAINS EXPLANATIONS



blocks on

In contrast to Blocksworld domain, Blocks-on tasks just require to place a single specific block on top of another block



Childsnack

there is a kitchen, a set of tables, a set of plates, a set of children, all sitting at some table waiting to be served a sandwich. Some children are allergic and they must be served a gluten-free sandwich



Miconic

here are passengers, each waiting at some floor, who want to take an elevator to a target floor



Gripper

in Gripper there are two rooms a and b. A robot can move between a and b, and pick up and drop balls. The goal is to move all balls from room a to b.

LEARNING RESULTS



Table 1 shows results for the learning process. We learn sketches of width 0 in 6 out of the 9 domains, of width 1 in all 9 domains, and of width 2 in 8 out of 9 domains. In all cases, very few and very small training instances are sufficient for learning the sketches as indicated by the number of instances and the number of states that are actually used (columns $|P_i|$ and $|S|$)

The choice of bound k has a strong influence because both the solution space and the number of subgoals (tuples) grows with k . Also, the number of rules decreases as the bound k increases. The most complex features overall have a complexity 7 (requiring the application of 7 grammar rules), the highest number of features in a sketch is 4, and the highest number of sketch rules is 5, showing that the learned sketches are very compact.

| Domain | $w = 0$ | | | | | | | | $w = 1$ | | | | | | | | $w = 2$ | | | | | | | |
|--------------|---------|-----|---------|-------|-----------------|---|----------|-------|---------|------|---------|-------|-----------------|---|----------|-------|---------|-----|---------|-------|-----------------|---|----------|-------|
| | M | T | $ P_i $ | $ S $ | $ \mathcal{F} $ | C | $ \Phi $ | $ R $ | M | T | $ P_i $ | $ S $ | $ \mathcal{F} $ | C | $ \Phi $ | $ R $ | M | T | $ P_i $ | $ S $ | $ \mathcal{F} $ | C | $ \Phi $ | $ R $ |
| Blocks-clear | 1 | 3 | 1 | 22 | 233 | 2 | 2 | 2 | 1 | 4 | 1 | 22 | 233 | 4 | 1 | 1 | 1 | 3 | 1 | 22 | 233 | 4 | 1 | 1 |
| Blocks-on | 26 | 14k | 1 | 22 | 1011 | 7 | 3 | 3 | 9 | 105 | 1 | 22 | 1011 | 4 | 2 | 2 | 13 | 146 | 1 | 22 | 1011 | 4 | 1 | 1 |
| Childsnack | — | — | — | — | — | — | — | — | 122 | 228k | 3 | 792 | 629 | 6 | 4 | 5 | — | — | — | — | — | — | — | — |
| Delivery | — | — | — | — | — | — | — | — | 17 | 521 | 1 | 96 | 474 | 4 | 2 | 2 | 3 | 18 | 1 | 20 | 287 | 4 | 1 | 1 |
| Gripper | 2 | 19 | 1 | 28 | 301 | 4 | 2 | 3 | 3 | 60 | 1 | 28 | 301 | 4 | 2 | 2 | 7 | 48 | 1 | 28 | 301 | 4 | 1 | 1 |
| Miconic | — | — | — | — | — | — | — | — | 1 | 5 | 1 | 32 | 119 | 2 | 2 | 2 | 2 | 6 | 1 | 32 | 119 | 2 | 1 | 1 |
| Reward | 3 | 46 | 2 | 26 | 916* | 6 | 2 | 2 | 1 | 4 | 1 | 12 | 210 | 2 | 1 | 1 | 10 | 95 | 1 | 48 | 427 | 2 | 1 | 1 |
| Spanner | 12 | 2k | 3 | 227 | 658 | 7 | 2 | 2 | 3 | 22 | 1 | 74 | 424 | 5 | 1 | 1 | 6 | 38 | 1 | 74 | 424 | 5 | 1 | 1 |
| Visitall | 3 | 54 | 2 | 36 | 722* | 5 | 2 | 2 | 1 | 1 | 1 | 3 | 10 | 2 | 1 | 1 | 1 | 1 | 1 | 3 | 10 | 2 | 1 | 1 |

Table 1: Learning step. We show the peak memory in GiB after learning (M), the time in seconds for solving the ASP in parallel on 32 CPU cores (T), the number of training instances used in the encoding ($|P_i|$), the total number of states considered in the encoding ($|S|$), the number of Boolean and numerical features ($|\mathcal{F}|$) where * denotes that the distance feature was included, the largest complexity of a feature $f \in \Phi$ (C), the number of features ($|\Phi|$), and the number of sketch rules ($|R|$). We use “—” to indicate that the learning procedure failed because of insufficient resources.

SEARCH RESULTS



Table 2 shows the results of SIWR searches on large, unseen test instances using the learned sketches. As a reference point, we include the results for the domain-independent planners LAMA and Dual-BFWS.

For Childsnack, Gripper, Miconic and Visitall, we use the Autoscale 21.11 instances for testing and for the other domains, where no Autoscale instances are available, we generate 30 instances ourselves, with the number of objects varying between 10 and 100. Table 2 shows that the learned sketches of width $k = 1$ yield solutions to the 30 instances of each of the domains. Some of these domains, such as Childsnack, Spanner and Visitall, are not trivial for LAMA and Dual-BFWS, which only manage to solve 9, 0, and 29 instances, and 5, 0, and 25 instances, respectively. In all cases, the maximum effective width of the SIWR search is bounded by the width parameter k , showing that the properties of the learned sketches generalize beyond the training set.

| Domain | $w = 0$ | | | | $w = 1$ | | | | $w = 2$ | | | | LAMA | | BFWS | |
|-------------------|---------|------|------|----|---------|----|------|----|---------|-----|------|----|------|-----|------|-----|
| | S | T | AW | MW | S | T | AW | MW | S | T | AW | MW | S | T | S | T |
| Blocks-clear (30) | 30 | 3 | 0.00 | 0 | 30 | 5 | 0.80 | 1 | 30 | 4 | 0.80 | 1 | 30 | 4 | 30 | 6 |
| Blocks-on (30) | 30 | 3 | 0.00 | 0 | 30 | 6 | 1.00 | 1 | 30 | 3 | 0.98 | 1 | 30 | 4 | 30 | 25 |
| Childsnack (30) | — | — | — | — | 30 | 1 | 0.10 | 1 | — | — | — | — | 9 | 2 | 5 | 658 |
| Delivery (30) | — | — | — | — | 30 | 1 | 1.00 | 1 | 30 | 4 | 1.66 | 2 | 30 | 1 | 30 | 1 |
| Gripper (30) | 30 | 4 | 0.00 | 0 | 30 | 3 | 0.50 | 1 | 30 | 656 | 2.00 | 2 | 30 | 1 | 30 | 6 |
| Miconic (30) | — | — | — | — | 30 | 5 | 0.53 | 1 | 30 | 132 | 2.00 | 2 | 30 | 7 | 30 | 25 |
| Reward (30) | 30 | 4 | 0.00 | 0 | 30 | 2 | 1.00 | 1 | 30 | 1 | 1.00 | 1 | 30 | 2 | 30 | 1 |
| Spanner (30) | 30 | 3 | 0.00 | 0 | 30 | 4 | 0.24 | 1 | 30 | 3 | 0.24 | 1 | 0 | — | 0 | — |
| Visitall (30) | 26 | 1360 | 0.00 | 0 | 30 | 20 | 0.00 | 1 | 30 | 21 | 0.00 | 1 | 29 | 213 | 25 | 833 |

Table 2: Testing step. We show the number of solved instances (S), the maximum time for solving an instance for which all algorithms find a solution, excluding algorithms that solve no instance at all (T), the average effective width (AW), and the maximum effective width (MW).

SKETCH ANALYSIS



blocks on

In contrast to Blocksworld domain, Blocks-on tasks just require to place a single specific block on top of another block



Miconic

here are passengers, each waiting at some floor, who want to take an elevator to a target floor



Gripper

in Gripper there are two rooms a and b. A robot can move between a and b, and pick up and drop balls. The goal is to move all balls from room a to b.

In this section the publishers describe and analyze some of the learned sketches and prove that they all have width bounded by k and are acyclic.

They provide proofs of these claims in an extended version of the paper (Drexler seipp, and Geffner 2022).

In Gripper there are two rooms a and b. A robot can move between a and b, and pick up and drop balls. The goal is to move all balls from room a to b. We analyze the three learned sketches for the bounds $k = 0, 1, 2$.



learned sketch $R2\Phi$ for $k = 2$ has

features $\Phi = \{g\}$

where g is the number of well placed balls, and a single rule

$r1 = \{\} \rightarrow \{g \uparrow\}$

saying that increasing the number of well-placed balls is good.

These subproblems have indeed width bounded by 2.

learned sketch $R1\Phi$ for $k = 1$ has

features $\Phi = \{ga, g\}$

where ga is the number of balls in room a and g is the number of balls that are either in room a or b.

The rules are

$r1 = \{\} \rightarrow \{g?, ga \downarrow\}$
 $r2 = \{\} \rightarrow \{g \uparrow\}$

where $r1$ says that picking up a ball in room a is good, and $r2$ says that dropping a ball in room b is good.

learned sketch $R0\Phi$ for $k = 0$ has

features $\Phi = \{B, c\}$

where B is true iff the robot is in room b and c is the number of carried balls, and rules

$r1 = \{c = 0\} \rightarrow \{c?, \neg B\}$
 $r2 = \{B\} \rightarrow \{B?, c \downarrow\}$
 $r3 = \{\neg B, c > 0\} \rightarrow \{B?\}$

where $r1$ says that moving to room a or picking up a ball in room a is good when not carrying a ball, $r2$ says that dropping a ball in room b is good, and $r3$ says that moving to room b while carrying a ball is good.



blocks on

The Blocks-on domain works like the standard Blocksworld domain, where a set of blocks can be stacked on top of each other or placed on the table. In contrast to the standard domain, Blocks-on tasks just require to place a single specific block on top of another block

The learned sketch $R\Phi$ for $k = 1$ has

features $\Phi = \{H, g\}$

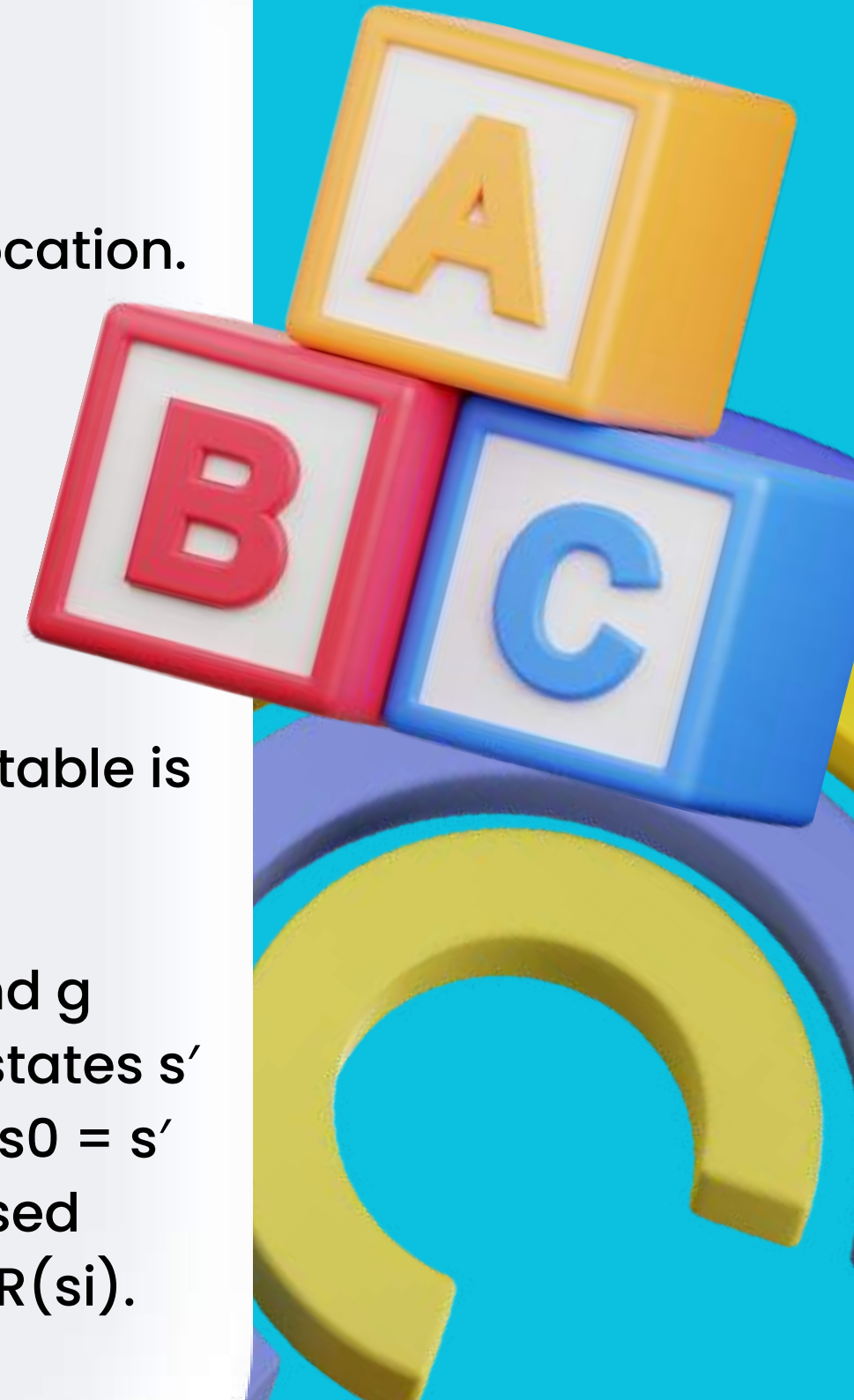
where H is true iff a block is being held and g is the number of blocks that are at their goal location.

The sketch rules are

$$r1 = \{\} \rightarrow \{\neg H, g \uparrow\}$$
$$r2 = \{H\} \rightarrow \{\neg H, g?\}$$

where $r1$ says that well-placing the block mentioned in the goal or unstacking towers on the table is good, and $r2$ says that not holding a block is good.

Starting in states s_0 where H is false, rule $r1$ looks for subgoal states s_1 where H is false and g increased. Starting in states s'_0 where H is true, rule $r2$ is also active which looks for subgoal states s'_1 where H is false and g has any value. From such states s'_1 , however, rule $r1$ takes over with $s_0 = s'_1$, and no subgoal state where H is true is reached again. If such states need to be traversed in the solution of the subproblems, they will not be encountered as subgoal states $s_{i+1} \in GR(s_i)$.



In Miconic (Koehler and Schuster 2000), there are passengers, each waiting at some floor, who want to take an elevator to a target floor.



Miconic

The learned sketch $R\Phi$ for $k = 1$ has

features $\Phi = \{b, g\}$

where b is the number of boarded passengers and g is the number of served passengers. The sketch rules are

$$r1 = \{\} \rightarrow \{b?, g \uparrow\}$$
$$r2 = \{\} \rightarrow \{b \uparrow, g?\}$$

where $r1$ says that moving a passenger to their target floor is good, and $r2$ says that letting some passenger board is good.

The learned sketches showed have been proved to all have width bounded by k and are acyclic.

provided proofs of these claims are in an extended version of the paper (Drexler, Seipp, and Geffner 2022).



CONCLUSIONS FROM THE PAPER

We have developed a formulation for learning sketches automatically given instances of the target class of problems Q and a bound k on the sketch width. The work builds on prior works that introduced the ideas of general policies, sketches, problem width, and description logic features.

The learning formulation guarantees a bounded width on the training instances but the experiments show that this and other properties generalize to entire families of problems Q , some of which are challenging for current domain-independent planners. The properties ensure that all problems in Q can be solved in polynomial time by a variant of the SIW algorithm. *This is possibly the first general method for learning how to decompose planning problems into subproblems with a polynomial complexity that is controlled with a parameter.*

Three limitations of the proposed learning approach are:

- 1) it cannot be applied to intractable domains *(computationally infeasible or very difficult to solve)
- 2) it yields large (grounded) ASP programs that are often difficult to solve
- 3) it deals with collections of problems encoded in PDDL-like languages, even if the problem of learning subgoal structure arises in other settings such as RL.

Three goals for the future are to improve scalability, to deal with non-PDDL domains, taking advantage of recent approaches that learn such domains from data, and to use sketches for learning hierarchical policies.

