Duality in STRIPS planning*

Martin Suda
Institute for Information Systems, Vienna University of Technology, Austria

Abstract

We describe a duality mapping between STRIPS planning tasks. By exchanging the initial and goal conditions, taking their respective complements, and swapping for every action its precondition and delete list, one obtains for every STRIPS task its dual version, which has a solution if and only if the original does. This is proved by showing that the described transformation essentially turns progression (forward search) into regression (backward search) and vice versa.

The duality sheds new light on STRIPS planning by allowing a transfer of ideas from one search approach to the other. It can be used to construct new algorithms from old ones, or (equivalently) to obtain new benchmarks from existing ones. Experiments show that the dual versions of IPC benchmarks are in general quite difficult for modern planners. This may be seen as a new challenge. On the other hand, the cases where the dual versions are easier to solve demonstrate that the duality can also be made useful in practice.

1 Introduction

Propositional STRIPS language is one of the favourite formalisms for describing planning tasks. A STRIPS task description consists of an initial and goal condition formed by conjunctions of propositional atoms and of a set of actions made up by a precondition, add and delete lists. Despite its simplicity, the modelling power of the STRIPS formalism already captures the complexity class PSPACE (Bylander, 1994). Also, STRIPS lies in the core of the more expressive PDDL language (McDermott, 2000) used for representing benchmarks in the International Planning Competition.

Classical search is one of the basic but also most successful approaches to determining whether a given planning task has a solution. The search may proceed either in the forward direction starting from the initial state and applying actions until a goal state is reached, or in the backward direction where the goal condition is regressed over actions to produce sub-goals until a sub-goal satisfied by the initial state is obtained. Forward search is typically termed progression, while backward search is called regression.

*This research has been conducted at Max-Planck-Institut für Informatik, Saarbrücken, Germany. The author was also supported by the ERC Starting Grant 2014 SYMCAR 639270 and the Austrian research project FWF RiSE S11409-N23.

In this paper, we show that from the computational perspective there is no real difference between progression and regression in STRIPS planning. This is very surprising because progression is working with single states only while the sub-goal conditions in regression represent whole state sets. We show this result by describing a duality mapping working on the domain of all STRIPS planning tasks. Performing regression on the original task is shown equivalent to performing progression on the dual.

The existence of the duality mapping has some additional interesting consequences. For instance, any notion originally conceived and developed with one of the search approaches in mind has a dual counterpart within the other approach. We give examples of this phenomenon in Section 5, one of them being the dual of the relevance condition, an important ingredient in pruning the regression search space. The duality can also be used to construct new algorithms from old ones and to obtain new benchmarks from existing ones. Thus a purely theoretical concept at first sight, the duality also has immediate implications for practice.

The rest of the paper is organized as follows. After giving the necessary preliminaries in Section 2, we recall the details about progression and regression relevant for our work in Section 3. The duality mapping is defined and its properties are stated and proven in Section 4. We subsequently discuss immediate theoretical implications of the duality in Section 5. Section 6 then reports on our experiments. We compare the performance of several modern planners on dual versions of IPC benchmarks and also show how a planner can be adapted with the help of the duality to solve benchmarks previously out of reach. Finally, in the concluding Section 7, we discuss the applications of the duality from a broader perspective.

Previous work. The idea of inverting the search direction in planning was already considered by Massey (1999) in his dissertation. Our main theorem can be recovered from that work, where it follows from a more general, but perhaps a less elegant result. A proof similar to the one presented here can be found in Pettersson (2005).

Problem reversal was used by Haslum (2008) to enable progression-like reachability heuristics being used for regression search. Alcázar and Torralba (2015) use the same technique to compute backward invariants of planning prob-

lems. This is done, however, within the SAS$^+$ formalism and is therefore not directly comparable to our results.

It seems that although already known, the idea of duality is not very well known among the planning community. We hope that the discussion on both its theoretical and practical implications as well as the experimental evaluation presented in this paper will trigger further research on this interesting notion.

2 Preliminaries

A propositional STRIPS *planning task* is defined as a tuple $\mathcal{P} = (X, I, G, \mathcal{A})$, where X is a finite set of *atoms*, $I \subseteq X$ is the *initial condition*, $G \subseteq X$ is the *goal condition*, and $\mathcal{A}$ a finite set of *actions*. Every action $a \in \mathcal{A}$ is a triple $a = (pre_a, add_a, del_a)$ of subsets of X referred to as the action's *precondition*, *add list*, and *delete list*, respectively.

The semantics is given by associating each planning task $\mathcal{P} = (X, I, G, \mathcal{A})$ with a *transition system* $\mathcal{T}_\mathcal{P} = (S, I, S_G, T)$, where the set of *world states* $S = 2^X$ is identified with the set of all subsets of X, the *initial state* is the subset I, the *goal states* $S_G = \{s \in S \mid G \subseteq s\}$ are those states that satisfy the goal condition G, and, finally, the *transition relation* T, which consists of state-action-state triples called *transitions*, is defined as follows:

$$T = \{s \xrightarrow{a} s' \mid pre_a \subseteq s \wedge s' = (s \cup add_a) \setminus del_a\}.$$

A planning task *has a solution* if there is a *path* in the respective transition system from the initial state to a goal state, i.e. if there is a finite sequence of transitions $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots s_{k-1} \xrightarrow{a_k} s_k$ such that $s_0 = I$ and $s_k \in S_G$. Notice that the path π is fully determined by the sequence of actions $a_1, \ldots, a_k$, which we call a *plan* for $\mathcal{P}$.

3 Progression and regression

There are two basic approaches to searching for solutions of planning tasks: progression and regression Russell and Norvig (2010). Progression, or simply forward search, proceeds systematically from the initial state and applies actions until a goal state is reached. Regression, or backward search, on the other hand, regresses the goal condition over actions to produce sub-goals until a sub-goal contained in the initial state is obtained.

In what follows we abstract away the actual search algorithm and only focus on properties of the two approaches that are important for showing their correctness. These properties depend solely on three "entry point" procedures, by which the actual search algorithm could be parameterized:

- $\texttt{start}()$, which generates a start *search node*,
- $\texttt{is\_target}(t)$, which tests whether a given search node is a *target* node, and
- $\texttt{succ}(t)$, which generates *successor* nodes t' of the given search node t.

Given a plannig task $\mathcal{P} = (X, I, G, \mathcal{A})$, the respective implementations of the procedures for progression and regression are summarized in Table 1. Let us first focus on progression. There, each search node directly corresponds to a world state, or, more specifically, to a world state reachable from the initial state. The start search node $\texttt{start}^{Pr}()$ is equal to the initial state I itself, the $\texttt{is\_target}^{Pr}(t)$ procedure tests whether the given node satisfies the goal condition, and the successor nodes $\texttt{succ}^{Pr}(t)$ are constructed by taking for every action $a \in \mathcal{A}$ for which the *applicability* condition $pre_a \subseteq t$ is satisfied the successor node $t' = (t \cup add_a) \setminus del_a$. This naturally corresponds to the definition of the transition system $\mathcal{T}_\mathcal{P}$ and so the proof of the following correctness theorem for progression becomes immediate.

Theorem 1. *A planning task $\mathcal{P} = (X, I, G, \mathcal{A})$ has a solution if and only if there exists a sequence of search nodes $t_0, \ldots, t_k$ such that $t_0 = \texttt{start}^{Pr}()$, $\texttt{is\_target}^{Pr}(t_k)$, and for every $i = 1, \ldots, k$ $t_i \in \texttt{succ}^{Pr}(t_{i-1})$.*

In the case of regression, a search node is also represented by a subset of X, but it should be viewed as a sub-goal to be met, corresponding to a set of world states that satisfy it. Here, the search nodes are manipulated in the following way. The start search node $\texttt{start}^{Re}()$ is identified with the (sub-)goal G itself, the $\texttt{is\_target}^{Re}(t)$ procedure returns true if and only if the initial state I satisfies t, and the successor search nodes $\texttt{succ}^{Re}(t)$ are generated by collecting the regressed sub-goals $t' = (t \setminus add_a) \cup pre_a$ for every action $a \in \mathcal{A}$ for which the *consistency* condition $del_a \cap t = \emptyset$ holds. The key property of regression is that in every world state s satisfying the regressed sub-goal t' (i.e., in every s such that $t' \subseteq s$) the action a is applicable ($pre_a \subseteq s$) and leads to a world state that satisfies the original sub-goal t. Consistency is needed to ensure that the action does not undo any desired atom.

Remark. Another property that is typically required, apart from consistency, is relevance. An action $a \in \mathcal{A}$ is said to be *relevant* for achieving a sub-goal t if and only if $add_a \cap t \neq \emptyset$, i.e., if when applied, it achieves a part of the sub-goal. Because relevance is only important for efficiency and not for correctness of algorithms based on regression, we set it aside for now, to keep things simple, and return to it in a later discussion.

The correctness theorem for regression has exactly the same form as the one for progression. We do not detail its proof, which is standard and basically just combines the insights mentioned above.

Theorem 2. *A planning task $\mathcal{P} = (X, I, G, \mathcal{A})$ has a solution if and only if there exists a sequence of search nodes $t_0, \ldots, t_k$ such that $t_0 = \texttt{start}^{Re}()$, $\texttt{is\_target}^{Re}(t_k)$, and for every $i = 1, \ldots, k$ $t_i \in \texttt{succ}^{Re}(t_{i-1})$.*

4 Duality

When looking at Table 1, which compares progression and regression, it is not difficult to observe certain formal similarities. For instance, the role played by the initial condition I in progression is similar to the one played by G in regression and vice versa. Similarly, the precondition pre_a and the delete list del_a of the considered action a seem to be exchanging roles in a certain way. In this section we describe an involutory mapping $\_^d : STRIPS \to STRIPS$ acting on

	progression: $\_^{Pr}$	regression: $\_^{Re}$
`start()`	I	G
`is-target`(t)	$G \subseteq t$	$t \subseteq I$
`succ`(t)	$\{\ t' \mid \exists a \in \mathcal{A}\ .\ pre_a \subseteq t \wedge t' = (t \cup add_a) \setminus del_a\ \}$	$\{\ t' \mid \exists a \in \mathcal{A}\ .\ del_a \cap t = \emptyset \wedge t' = (t \setminus add_a) \cup pre_a\ \}$

Table 1: Instantiating progression and regression for a plannig task $\mathcal{P} = (X, I, G, \mathcal{A})$.

the class of all STRIPS planning tasks that shows that the above similarities are not a coincidence and that progression and regression are more closely related than is would seem at first sight.

For an action $a = (pre_a, add_a, del_a)$ a dual action a^d is formed by exchanging the precondition and delete list: $a^d = (del_a, add_a, pre_a)$. For a set of actions $\mathcal{A}$ the set of dual actions is $\mathcal{A}^d = \{a^d \mid a \in \mathcal{A}\}$. Now, given a planning task $\mathcal{P} = (X, I, G, \mathcal{A})$ the dual task $\mathcal{P}^d$ is obtained by exchanging the initial and goal conditions while taking their complements with respect to X, and using the dual action set:

$$\mathcal{P}^d = (X, (X \setminus G), (X \setminus I), \mathcal{A}^d).$$

It is easy to see that a mapping $\_^d$ defined in this way is indeed *involutory* on the set of STRIPS planning tasks, meaning that $(\mathcal{P}^d)^d = \mathcal{P}$ for every task $\mathcal{P}$. This justifies the use of the term duality.

We can now state the central theorem about duality.

Theorem 3. *For every planning task $\mathcal{P} = (X, I, G, \mathcal{A})$ the dual task $\mathcal{P}^d$ has a solution if and only if $\mathcal{P}$ does. More specifically, a sequence of actions $a_1, \ldots, a_k$ is a plan for $\mathcal{P}$ if and only if the sequence $a_k^d, \ldots, a_1^d$ is a plan for $\mathcal{P}^d$.*

Proof. If a planning task has a solution, it can be found by both progression and regression, because they are both correct (Theorem 1 and 2). We prove this Theorem 3 by showing that regression for $\mathcal{P}$ performs exactly the same operations as progression for $\mathcal{P}^d$ when the search nodes are represented in a complemented form for the latter, i.e. when storing $X \setminus t$ in place of t. This is done in three steps corresponding to the three "entry point" procedures of Table 1.

First, we realize that

$$\texttt{start}^{Re}_{\mathcal{P}}() = X \setminus \texttt{start}^{Pr}_{\mathcal{P}^d}().$$

In words, the start search node of regression for $\mathcal{P}$, is the complement (with respect to X) of the start search node of progression for $\mathcal{P}^d$. Similarly, a search node $t \subseteq X$ is a target node in regression for $\mathcal{P}$ if and only if $(X \setminus t)$ is a target node in progression for $\mathcal{P}^d$:

$$\texttt{is\_target}^{Re}_{\mathcal{P}}(t) = \texttt{is\_target}^{Pr}_{\mathcal{P}^d}(X \setminus t),$$

which follows from the equivalence

$$a \subseteq b \leftrightarrow (X \setminus b) \subseteq (X \setminus a).$$

Finally, the successor nodes of a search node $t \subseteq X$ in regression for $\mathcal{P}$ can be computed as complements of successor nodes of $(X \setminus t)$ in progression for $\mathcal{P}^d$:

$$\texttt{succ}^{Re}_{\mathcal{P}}(t) = \{(X \setminus t_0) \mid t_0 \in \texttt{succ}^{Pr}_{\mathcal{P}^d}(X \setminus t)\}.$$

For this last point, it is sufficient to verify for every action $a \in \mathcal{A}$ that 1) the consistency condition in regression for $\mathcal{P}$ and applicability condition in progression for $\mathcal{P}^d$ are each other's dual:

$$\begin{aligned} del_a \cap t = \emptyset &\leftrightarrow del_a \subseteq (X \setminus t) \\ &\leftrightarrow pre_{a^d} \subseteq (X \setminus t), \end{aligned}$$

and, 2) regressing t over a yields the complement of applying a^d to the complement of t:

$$\begin{aligned} X \setminus ((t \setminus add_a) \cup pre_a) &= ((X \setminus t) \cup add_a) \setminus pre_a \\ &= ((X \setminus t) \cup add_{a^d}) \setminus del_{a^d}. \end{aligned}$$

With these two properties checked (by applying De Morgan's laws for sets) the theorem is proven by induction over k, the length of a solution path $\pi = s_0 \xrightarrow{a_1} s_1 \ldots s_{k-1} \xrightarrow{a_k} s_k$ and the corresponding plan $a_1, \ldots, a_k$. □

5 Implications

The most striking consequence of Theorem 3 is the discovery that in STRIPS planning there is no substantial difference between progression and regression. Indeed, any algorithm based on one of the two approaches may be effectively turned into an algorithm based on the other by simply applying the duality mapping to the input as a preprocessing and running the actual algorithm on $\mathcal{P}^d$ instead of on $\mathcal{P}$. This transformation obviously preserves the length of the shortest plan and its cost.

Given this perspective, it is now interesting to observe what are the dual counterparts of notions that were originally conceived and developed with only one of the approaches in mind and in how do they emerge "on the other side of the duality". We will now comment on some of these observations in the following subsections.

Relevance and usefulness

It was mentioned before that it is important for the efficiency of regression to only regress over actions that are relevant for the current sub-goal. Let us repeat that an action $a \in \mathcal{A}$ is relevant for t if and only if $add_a \cap t \neq \emptyset$. Regressing over an action that is not relevant for t results in a (possibly strictly) stronger sub-goal $t' \supseteq t$. We may safely discard t' from consideration, because successfully regressing t' is (possibly strictly) more difficult than successfully regressing t.[1] This way filtering out non-relevant actions helps to keep the regression search space manageable.

It is now at hand to ask what the dual notion of relevance is. For lack of a better word, we call it usefulness. We say

[1] If solution can be found from t', it can be found from t as well.

that an action $a \in \mathcal{A}$ is *useful* in a state t if and only if the add list of a is not fully contained in t. We see that usefulness is a natural property: it does not make sense to progress via a non-useful action, because it will never make more atoms true in the resulting state. The reason why usefulness is generally not mentioned in the literature is that in typical benchmarks there are seldom actions that would be applicable and yet not useful in a given state. This is in contrast with regression where consistency and non-relevance are far less correlated.

First add, then delete?

When defining the result of action application to a state, one needs to decide in which order should the add list and the delete list be considered. In particular, if a description of a planning task contains an action a such that add_a and del_a have a non-empty intersection, the result of applying a to a state s depends on this order. One can either exclude this possibility up front by requiring that for any action the add and delete lists are disjoint, or, alternatively, to decide on a canonical order of their application.

There are two remarks we can make here with respect to duality. First, if we choose the former option above, i.e., if we require that $add_a \cap del_a = \emptyset$ for any $a \in \mathcal{A}$, we should perhaps (for the sake of symmetry) also require that $add_a \cap pre_a = \emptyset$, because that is exactly the condition under which the order of applying add list and the precondition during regression of a sub-goal becomes irrelevant. Note that this condition also makes sense from the perspective of progression, because atoms mentioned in the precondition will be preserved by the action anyway (unless deleted) so they do not need to be mentioned again in the add list.

The second remark relates to the latter option, when in order to resolve the above situation a particular add-delete order is chosen as canonical. Here the duality dictates (with appeal to elegance of the theory) that adding should happen before deleting, as done in our definition in Section 2. It is because only with that order the proof of Theorem 3 goes through as presented.

Let us be more specific. In progression we, quite naturally, first check the applicability condition $pre_a \subseteq s$, before applying the effects. That is why the corresponding regression operation needs to first subtract the add list from the sub-goal, before adding the preconditions: $t' = (t \setminus add_a) \cup pre_a$. Then dualizing the last equation gives us $s' = (s \cup add_a) \setminus del_a$ as promised. This should not be interpreted as saying that the duality itself relies on a particular ordering of addition and deletion in the definition of action application. Should the other order be adopted instead, however, we would need to require that the actions of a planning task are normalized beforehand so that the intersection of add and delete lists is always empty.

Semantics of search nodes

Since the duality exchanges the roles of progression and regression, one should ask what happens to the semantics of the search nodes, which are known to represent world states in progression and sets of world states (via conjunctive conditions) in regression. The surprising answer the duality gives is that both the views are equally valid for both progression and regression. One just needs to go over to the complement representation to see the other.

Essentially, nodes in progression can be interpreted as a conditions, where a condition t stands for all the states s such that $s \subseteq t$, i.e. states having at most those positive facts as those stated in t, but no others. This is because in STRIPS, we can only make a task of reaching a goal harder by removing a fact from from a state.

Dually, regression can be thought of as performed over single states only, the states corresponding to the search nodes themselves, because we can only make regression harder by adding facts to such states. We invite the reader to check the details for herself by replaying the proof of Theorem 3 from this perspective.

Note that this observation provides us with a new way (arguably less intuitive, but nevertheless a legitimate one) to justify the correctness of the two approaches. While this may sometimes simplify argumentations, the actual implementation "mechanics" remains intact.

Limitations

We close this section by discussing the limitations of the duality concept. A careful analysis of the proof of Theorem 3 reveals that it substantially relies on the particularly simple form of regression in STRIPS planning. Essential is the fact that regressed sub-goals may be represented as conjunctions of atoms. This means the duality does not directly carry over to more expressive formalisms which allow negated goals or preconditions. For similar reasons, extending the duality to Finite Domain Representation (FRD) Helmert (2009) seems problematic. The good news is that the duality applies to the lifted version of STRIPS as realized by the STRIPS subset of the PDDL language McDermott (2000) used in the International Planning Competition (IPC).[2] The IPC benchmark set contains more than a thousand practically relevant problems to which the duality applies.

6 Experiments

The duality mapping we have described in the previous section provides us with a means of transforming one planning task into another while preserving the existence of its solution. It is now natural to ask how difficult are the dual versions of IPC benchmarks for modern planners. We performed a series of experiments in order to answer this question and we report on them in this section.

Note that there are two possible ways of interpreting the results. We may either view the dual versions as new standalone problems, or imagine the duality mapping as part of the algorithm we are currently testing. The second case may be understood as an evaluation of a new, dual algorithm on the original benchmarks. We will prefer the first view for most of this section, but adopt the second where it is more natural.

For our experiments, we collected all the benchmarks from the satisficing tracks of the International Planning

[2] To complement the initial and goal condition, one first obtains the set of all atoms X by grounding the domain predicates.

	FF	LAMA	Mp
ORIG	1009	1192	1114
DUAL	136	175	329

Table 2: First experiment: number of ORIG and DUAL problems solved within 180 seconds by the respective planners.

	FF	FF-U	FF-UI	FF-UIN
DUAL	136	204	682	695

Table 3: Second experiment: number of problems from DUAL solved within 180 seconds by modifications of the planner FF.

Competitions[3] (IPC) of years 1998–2011 that are in the STRIPS subset of the PDDL language.[4] Together we collected 1564 problems. We then used the preprocessing part of the planner FF Hoffmann and Nebel (2001) to produce a grounded version of these. Note that FF's relevance analysis was involved in the process, so all the "rigid" predicates that are only used for modelling purposes and the value of which is not affected by any action were removed. Let us denote the set of these grounded IPC benchmarks ORIG.

The preprocessing tool was then extended further to implement the duality mapping: It first normalizes the actions so that the precondition and delete list never intersect with the add list. To conform with the official IPC semantics, which is "first delete, then add" Fox and Long (2003), this is done by performing for every action a the following two assignments in the prescribed order:

$$del_a := del_a \setminus add_a; \quad add_a := add_a \setminus pre_a.$$

Then the duality mapping is applied. Let the problems obtained this way be denoted as DUAL. All the experiments were performed our servers with 3.16 GHz Xeon CPU, 16 GB RAM, with Debian 6.0.

In the first experiment we ran the following three planners on both ORIG and DUAL benchmark sets:

- the FF planner Hoffmann and Nebel (2001) as a baseline representative of heuristic search Bonet and Geffner (2001) planners,
- the Fast Downward planner Helmert (2006) in the configuration LAMA-2011 Richter and Westphal (2010), another heuristic search planner, the winner of the satisfycing track of the last IPC held in 2011, and
- the planner Mp Rintanen (2010), as a representative of the planning as satisfiability Kautz and Selman (1996) approach.

The time limit was set to 180 seconds per problem.

The results of the first experiment are summarized in Table 2. We see that the problems in DUAL are generally much more difficult to solve than ORIG, and that the SAT-based planner Mp performs better on DUAL than the heuristic search planners.

We conjecture (and later partially verify) the following reasons for the difficulty of DUAL. First, the explicit state forward search planners suffer from not testing for usefulness of actions. This corresponds to omitting the relevance test in the dual, regression-based algorithm and makes the search space unnecessarily large. The second reason is that *invariant* information is no longer recovered from the task description by the planners. Invariant is a property which holds in the initial state and is preserved by all transitions. While logically redundant, invariants are known to be usually critical for efficiency of SAT-based planners Rintanen (2010). Moreover, the existence of simple invariants formed by negative binary clauses is a prerequisite for the reconstruction of a non-trivial Finite Domain Representation (FDR), which LAMA is attempting to build in its preprocessing phase Helmert (2009). As we independently checked, there are almost no binary clause invariants to be recovered from the DUAL benchmarks. This means that Mp has to search for plans without the useful guidance the invariants usually provide and LAMA most of the time discovers only trivial, two-valued domains for its finite domain variables.

Remark. Note that the problems in DUAL still contain the original invariant information, but it has been turned into *backward invariants*, properties of the goal states preserved when traversing the transitions backwards. Obviously, the planners do not check for backward invariants, because typically, e.g., on ORIG, it does not pay off.

In our second experiment, we set out to discover to what extent do the above reasons explain the degraded performance of the planners on DUAL. We focused on the planner FF for its relative simplicity and modified it in several steps in order to make it perform better on DUAL. We prepared the following versions of the planner:

- FF-U, which checks for usefulness of actions and discards the non-useful ones,
- FF-UI, which additionally computes[5] binary clause backward invariant, and discards successor states that violate it,
- FF-UIN, which additionally turns off enforced hill climbing (see Hoffmann and Nebel (2001)) and always directly starts best first search.[6]

We ran all the modifications on DUAL, again with the time limit of 180 seconds per problem.

The numbers of problems solved by the respective modifications are shown in Table 3. For the sake of comparison we also repeat the result for the original FF. It can be seen that each of the modifications represents an improvement over the previous version. Probably the most is gained by incorporating the backward invariant test. Actually, each modification solves a strict superset of the problems solved

[3] http://ipc.icaps-conference.org/

[4] We dropped the action cost feature where present.

[5] We use an efficient implementation of the fixpoint algorithm described in Rintanen (1998).

[6] We observed that enforced hill climbing fails on most of the problems in DUAL, so turning it off up front saves some time.

		FF (unique)	FF-DUAL (unique)
PSR	(50)	39 (2)	45 (8)
Woodworking	(50)	18 (2)	44 (28)
Floortile	(20)	7 (0)	17 (10)

Table 4: Third experiment: Comparing FF and FF-DUAL on three domains where the latter dominates the former. Size of each domain and the number of problems uniquely solved by the respective planner are shown in parenthesis.

by the previous one. An exception is the last step where FF-UI solves 3 problems that FF-UIN cannot solve. However, FF-UIN solves 16 problems that FF-UI cannot solve within the given time limit.

Despite our efforts to improve the performance of FF on DUAL, the planner still solves less problems from DUAL than from ORIG. In our third experiment, we tried to discover whether there are some problems in DUAL that the improved FF-UIN can solve, while the original FF fails on their counterparts in ORIG. This corresponds to the question whether the duality can be made useful in practice by helping to solve difficult IPC benchmarks. To simplify the following discussion, let us call by FF-DUAL a planner composed by the preprocessor, which grounds and dualizes inputs, followed by FF-UIN. We will now compare FF and FF-DUAL on ORIG.

Apart from six problems from the Mystery domain, where FF-DUAL correctly discovers that no plan can exists while FF timeouts, there are three domains where FF-DUAL performs consistently better than FF. Table 4 reports on the number of problems solved, categorized by the domains.

In order to better understand the success of FF-DUAL on the three domains, we more closely analyzed and compared the output of the two versions of the planner. In particular, we focused on the reported heuristic value of the currently expanded state. We noticed the following facts.

- On the domain PSR the heuristic value of the initial state is quite low (between 1 and 10). This holds for both FF and FF-DUAL, but the value for FF-DUAL is typically one higher than that for FF. In other words, the dual version of relaxed plan heuristic is more informative on PSR.
- On Woodworking, the heuristic value of the initial state ranges from 5 up to about 70. FF-DUAL's values are typically not higher, but stay quite close to those of FF.
- Although on Floortile, FF's heuristic is more informed than FF-DUAL's, FF's goal agenda mechanism seems to be making suboptimal decisions in decomposing the goal into sub-goals. On three problems where FF's enforced hill climbing fails within the time limit and the goal agenda is discarded, FF then successfully finds a plan with best first search. At the same time, FF-DUAL directly looks for a plan using best first search and its less informed heuristic.

On all the other domains FF-DUAL's heuristic value of the initial state is typically much lower than the corresponding estimate of FF. This might explain the general lower effectiveness of FF-DUAL on the ORIG benchmarks.

To sum up, in our experiments we have shown that the dual versions of IPC benchmarks are in general much more difficult to solve by modern planners than the originals. This can be partially remedied by adapting a planner to make use of specific features the dual benchmarks possess, but which are usually missing in the standard ones. Although the imagined dualizing planner FF-DUAL does not beat the original FF in the overall number of solved problems, there are certain domains where it indeed pays off to apply the duality mapping before looking for a plan. This represents one possible application of the duality concept in practice.

7 Conclusion

In this paper, we have described a duality mapping on the domain of all STRIPS planning tasks. Its existence shows that computationally, there is no real difference between performing progression and regression as they are each other's dual. Differences between the two that one can measure in practice follow from asymmetries (with respect to the mapping) of the concrete benchmarks and are not inherent to the search paradigms themselves. We believe that understanding these asymmetries and their influence on the efficiency of planning algorithms deserves further study.

Furthermore, we have pointed to several applications of the duality itself. We have shown that new theoretical insights may be obtained by translating known notions via the mapping and analyzing the obtained duals. For instance, there necessarily exists a "precondition relaxation heuristic" a dual of the famous delete relaxation heuristic.

Next we studied the dual versions of the standard IPC benchmarks and discovered they are quite difficult to solve for modern planners. One could argue that there is nothing interesting about difficult benchmarks in themselves if they do not come from practical applications – for instance, random problems form the phase transition region (see Rintanen (2004)) seem to have this status. We, however, do not think the dual IPC benchmarks fall into the same category. After all, they still encode the same transition structures as the originals, albeit in a non-obvious way. Therefore, we believe they should be considered as an auxiliary test set by anyone attempting to develop a really versatile planner.

Finally, we explored the possibility of using the duality to design new algorithms. A simple modification of the planner FF which uses the duality was shown to improve over the original system on several benchmark domains. Note that this obvious schema of first dualizing the input and then running a known algorithm is not the only option of how the duality can be used. More sophisticated algorithms combining progression and regression tied together by the duality can be imagined.

References

Vidal Alcázar and Álvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *ICAPS 2015*, pages 2–6. AAAI Press, 2015.

Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33, 2001.

Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

Maria Fox and Derek Long. Pddl2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

P. Haslum. Additive and reversed relaxed reachability heuristics revisited. In *6th International Planning Competition Booklet (ICAPS-08)*, 2008.

Malte Helmert. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)*, 26:191–246, 2006.

Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6):503–535, 2009.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)*, 14:253–302, 2001.

Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In William J. Clancey and Daniel S. Weld, editors, *AAAI/IAAI, Vol. 2*, pages 1194–1201. AAAI Press / The MIT Press, 1996.

Bart Massey. *Directions In Planning: Understanding The Flow Of Time In Planning*. PhD thesis, University of Oregon, 1999.

Drew V. McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, 2000.

Mats Petter Pettersson. Reversed planning graphs for relevance heuristics in AI planning. In *Planning, Scheduling and Constraint Satisfaction: From Theory to Practice*, volume 117 of *Frontiers in Artificial Intelligence and Applications*, pages 29–38. IOS Press, 2005.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. (JAIR)*, 39:127–177, 2010.

Jussi Rintanen. A planning algorithm not based on directional search. In Anthony G. Cohn, Lenhart K. Schubert, and Stuart C. Shapiro, editors, *KR 2004*, pages 617–625. Morgan Kaufmann, 1998.

Jussi Rintanen. Phase transitions in classical planning: An experimental study. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *KR 2004*, pages 710–719. AAAI Press, 2004.

Jussi Rintanen. Heuristics for planning with SAT. In David Cohen, editor, *CP 2010*, volume 6308 of *LNCS*, pages 414–428. Springer, 2010.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.