A Bilevel Planning Algorithm Details

Algorithms 3 and 4 provide pseudocode and additional details necessary for high-level and low-level search respectively in bilevel planning (Algorithm 5).

```
GenAbstractPlan($s_0$, $g$, $\Omega$, $n_{abstract}$)
    $\underline{\Omega} \leftarrow$ Ground($\Omega$)
    // Search over ground operators from s0 to goal (returns top n
       plans).
    $\hat{\pi} \leftarrow$ Search($s_0, g, \underline{\Omega}, n_{\text{abstract}}$)
    return $\hat{\pi}$
```

Algorithm 3: This is GenAbstractPlan which finds a high-level plan by creating operators for all possible groundings then uses search to find n_{abstract} plans. It returns a list of plans $\hat{\pi}$.

```
Refine(($\hat{\pi}$, $x_0$, $\Psi$, $\Sigma$, $n_{samples}$))
    state $\leftarrow$ Abstract($x_0$)
    // While current state is not goal, sample and run current operator
       on current state and check ground atoms.  If is passes continue
       if not backtrack.
    curr$_{\text{idx}} \leftarrow 0$
    while $curr_{idx} < len(\hat{\pi})$ do
        samples[curr$_{\text{idx}}$] $\leftarrow samples[curr_{idx}] + 1$
        state$_{\text{current}}$, $\underline{\Omega} \leftarrow \hat{\pi}$[curr$_{\text{idx}}$]
        $\underline{\Omega}.C.\Theta \sim \underline{\Omega}.\Sigma$
        $\pi$[curr$_{\text{idx}}$] $\leftarrow \underline{\Omega}.C$
        curr$_{\text{idx}} \leftarrow$ curr$_{\text{idx}} + 1$
        if $\underline{\Omega}.C.initiable(state_{current})$ then
            state$_{\text{next}} \leftarrow$ Simulate(state$_{\text{current}}$, $\underline{\Omega}.C$)
            state$_{\text{expected}}$, _ $\leftarrow \hat{\pi}$[curr$_{\text{idx}}$]
            if $state_{next} \subseteq state_{expected}$ then
                can_continue_on $\leftarrow True$
                if $curr_{idx} == len(skeleton)$ then
                    return sucess, $\pi$
            else
                canContinueOn $\leftarrow False$
        else
            canContinueOn $\leftarrow False$
        if not canContinueOn then
            curr$_{\text{idx}} \leftarrow$ curr$_{\text{idx}} - 1$
            if $samples[curr_{idx}] == max\_samples$ then
                return failure, $\pi$
    return success, $\pi$
```

Algorithm 4: This is Refine which turns a task plan $\hat{\pi}$ into a sequence of ground skills. It gets the state and operators from $\hat{\pi}$ and adds the controller with newly sampled continuous parameters to π. After this it checks to see if the added controller is initiable from the current state in the plan and we simulate the skill execution to verify it reached the expected state we predicted next in $\hat{\pi}$. If the controller is not initiable or fails the expected atoms check we backtrack and resample a new continuous parameter for this controller until either we reach the max number of samples or we successfully refine our final controller.

B Detailed Description of Successor Generators

B.1 IMPROVECOVERAGE

The pseudocode for the `improve-coverage` successor generator is shown in Algorithm 6. Given the current candidate operator set Ω, training demonstrations $\mathcal{D}$ and corresponding tasks $\mathcal{T}_{\text{train}}$, we

```
BILEVELPLANNING($\mathcal{O}$, $x_0$, $g$, $\Psi$, $\Omega$, $\Sigma$)
    // Parameters: $n_{\text{abstract}}$, $n_{\text{samples}}$.
    $s_0 \leftarrow$ ABSTRACT($x_0$)
    // Outer Planning Loop
    for $\hat{\pi}$ in GENABSPLAN($s_0$, $g$, $\Omega$, $n_{abstract}$) do
        // Inner Refinement Loop
        if REFINE($\hat{\pi}$, $x_0$, $\Psi$, $\Omega$, $n_{\text{samples}}$) succeeds w/ $\pi$ return $\pi$
```

Algorithm 5: Pseudocode for bilevel planning algorithm, adapted from Silver et al. [1]. The inputs are objects $\mathcal{O}$, initial state x_0, goal g, predicates Ψ, operators Ω, and samplers Σ; the output is a plan π. The outer loop GENABSPLAN generates high-level plans that guide our inner loop, which samples continuous parameters from our samplers Σ to concretize each abstract plan $\hat{\pi}$ into a plan π. If the inner loop succeeds, then the found plan π is returned as the solution; if it fails, then the outer GENABSTRACTPLAN continues.

```
IMPROVECOVERAGE($\Omega$, $\mathcal{D}$)
    $\text{cov}_{\text{init}}, \mathcal{D}_\alpha, \tau_{\text{unc}}, \alpha_{\text{unc}} \leftarrow$ COMPUTECOVERAGE($\Omega, \mathcal{D}$)
    if $cov_{init} = |\mathcal{D}|$ then
        return $\Omega$
    $\text{cov}_{\text{curr}} \leftarrow \text{cov}_{\text{init}}$
    $\Omega' \leftarrow \Omega$
    while $cov_{curr} \geq cov_{init}$ do
        $\omega_{\text{new}} \leftarrow$ INDUCEOPTOCOVER($\tau_{\text{unc}}, \alpha_{\text{unc}}$)
        $\Omega' \leftarrow$ REMOVEPRECANDDELEFFS($\Omega'$) $\cup\ \omega_{\text{new}}$
        $(\mathcal{D}_{\omega_1} \dots \mathcal{D}_{\omega_m}), (\delta_{\tau_1}, \dots, \delta_{\tau_j}) \leftarrow$ PARTITIONDATA($\Omega', \mathcal{D}$)
        $\Omega' \leftarrow$ INDUCEPRECANDDELEFFS($\Omega'$,
            $(\mathcal{D}_{\omega_1} \dots \mathcal{D}_{\omega_m})$,
            $(\delta_{\tau_1}, \dots, \delta_{\tau_j})$)
        $\Omega' \leftarrow \Omega' \cup$ ENSURENECATOMSSAT($\omega_{\text{new}}, \mathcal{D}_\alpha$)
        $(\mathcal{D}_{\omega_1} \dots \mathcal{D}_{\omega_l}), (\delta_{\tau_1}, \dots, \delta_{\tau_j}) \leftarrow$ PARTITIONDATA($\Omega', \mathcal{D}$)
        $\Omega' \leftarrow$ INDUCEPRECANDDELEFFS($\Omega'$,
            $(\mathcal{D}_{\omega_1} \dots \mathcal{D}_{\omega_l})$,
            $(\delta_{\tau_1}, \dots, \delta_{\tau_j})$)
        $\text{cov}_{\text{curr}}, \mathcal{D}_\alpha, \tau_{\text{unc}}, \alpha_{\text{unc}} \leftarrow$ COMPUTECOVERAGE($\Omega', \mathcal{D}$)
    $\Omega' \leftarrow$ PRUNENULLDATAOPERATORS($\Omega'$)
    return $\Omega'$
```

Algorithm 6: Pseudocode for our `improve-coverage` successor generator. The inputs are a set of operators Ω, the set of all training demonstrations $\mathcal{D}$, and the corresponding set of training tasks $\mathcal{T}_{\text{train}}$. The output is a set of operators Ω' such that $\texttt{coverage}(\Omega') \leq \texttt{coverage}(\Omega)$.

first attempt to compute the current coverage of Ω on $\mathcal{D}$. We do this by calling the COMPUTECOVERAGE method. This method simply calls Algorithm 1 on every demonstration $(\overline{x}, \overline{u})$ in $\mathcal{D}$ (the set of objects $\mathcal{O}$ and goal g required by Algorithm 1 are obtained from the training tasks). The COMPUTECOVERAGE method then returns the number of covered transitions[1] (cov_{init}), a dataset of necessary atoms sequences for each demonstration Algorithm 1 is able to cover ($\mathcal{D}_\alpha$), the first uncovered transition encountered ($\tau_{\text{unc}} = (s_k, u_{k+1}, s_{k+1})$), and the corresponding necessary atoms for the transition (α_{unc}). If the number of covered transitions is the same as the size of the training dataset, then all transitions must be covered and the `coverage` term in our objective (Equation 1) must be 0. We thus just return the current operator set Ω with no modifications. Otherwise, we compute a new set of operators Ω' with a lower `coverage` value.

To generate Ω', we first create a new operator with preconditions, add effects and arguments set to cover the transition τ_{unc} and corresponding necessary atoms α_{unc}. The operator's ground controller $\underline{C}(\theta) = u_{k+1}$ is determined directly from the transition's action u_{k+1}. The operator's ground add

---

[1] The total number of transitions in abstract plan suffixes that Algorithm 1 is able to find when run on each demonstration in $\mathcal{D}$.

effects are set to be $\underline{E^+} = (s_{k+1} \setminus s_k) \cap \alpha_{\text{unc}}$. The controller and add effects are lifted by creating a variable v_i for every distinct object that appears in $\underline{C} \cup \underline{E^+}$. The operator's arguments $\overline{v}$ are set to these variables.

Next, we must induce the preconditions and delete effects of this new operator ω_{new}. To this end, we add ω_{new} to our current candidate set, and partition all data in our training set $\mathcal{D}$ into operator specific datasets $\mathcal{D}_\omega$ for each operator ω in our current candidate set. Since operator preconditions and delete effects depend on the partitioning, we first remove these from all operators that are not ω_{new} (REMOVEPRECANDDELEFFS). We perform this partitioning by running the FINDBESTCONSISTENTOP method from Algorithm 8 on this new operator set for every transition in the dataset, though we do not check the condition $s_{i+1}^{pred} \subseteq s_{i+1}$, since the operators do not yet have delete effects specified. While performing this step, we save a mapping δ_{τ_i} from the operator's arguments to the specific objects used to ground it for every transition in the dataset (this will be used for lifting the preconditions and delete effects of each operator below). We assign each transition to the dataset associated with the operator returned by FINDBESTCONSISTENTOP. We return the operator specific datasets $(\mathcal{D}_{\omega_1} \dots \mathcal{D}_{\omega_l})$, as well as the saved object mappings for each transition $(\delta_{\tau_1}, \dots, \delta_{\tau_j})$.

We now induce preconditions and delete effects using $(\mathcal{D}_{\omega_1} \dots \mathcal{D}_{\omega_l})$ and $(\delta_{\tau_1}, \dots, \delta_{\tau_j})$. Before we do this, we delete any operator whose corresponding dataset is empty. Similar to Chitnis et al. [2], we set the *preconditions* to: $P \leftarrow \bigcap_{\tau=(s_i,\cdot,\cdot)\in\mathcal{D}_\omega} \delta_\tau(s_i)$. We also set the *atomic delete effects* to $E^-_\circ \leftarrow \bigcup_{\tau=(s_i,\cdot,s_{i+1})\in\mathcal{D}_\omega} \delta_\tau(s_{i+1}) \setminus \delta_\tau(s_i)$. For every transition (s_i, u_{i+1}, s_{i+1}), let $s_{i+1}^{pred} = (s_i \setminus \underline{E^-_\circ}) \cup \underline{E^+}$. Then, we set $s_{\text{mispred}} = \bigcup_{\tau=(\cdot,\cdot,s_{i+1})\in\mathcal{D}_\omega} s_{i+1}^{pred} \setminus s_{i+1}$. We induce a quantified delete effect for every *predicate* corresponding to atoms in s_{mispred}. We then set each operator's delete effects to be the union of $E^-_\circ$ and the quantified delete effects.

Now that all operators have preconditions and delete effects specified, we must ensure that the newly-added operator (ω_{new}) is able to satisfy the necessary atoms for each of its transitions in $\mathcal{D}_{\omega_{\text{new}}}$. Recall that we set the operator's add effects to be the necessary atoms that changed in the first uncovered transition τ_{unc}. Given the way partitioning is done (specifically the conditions in the FINDBESTCONSISTENTOP method in Algorithm 8), we know that these add effects must satisfy $\alpha_{i+1} \subseteq s_{i+1} \cup \underline{E^+}$ for all transitions $(s_i, u_{i+1}, s_{i+1}) \in \mathcal{D}_{\omega_{\text{new}}}$ with corresponding necessary atoms α_{i+1} for state s_{i+1}. However, the delete effects may cause the necessary atoms to become violated for certain transitions: i.e, $\alpha_{i+1} \nsubseteq (s_{i+1} \setminus \underline{E^-}) \cup \underline{E^+}$. For every such transition, we let $\alpha_{i+1}^{\text{miss}} = \alpha_{i+1} \setminus ((s_{i+1} \setminus \underline{E^-}) \cup \underline{E^+})$. We then create a new operator ω_i^{miss} by copying all components of ω_{new}, and adding lifted atoms from $\alpha_{i+1}^{\text{miss}}$ to both the preconditions and add effects. We modify the operator's arguments to contain new variables accordingly. This now ensures that the necessary atoms are not violated for any transition in $\mathcal{D}_{\omega_{\text{new}}}$. We add these new operators to the current candidate operator set.

After having added new operators to our candidate set in the above step, we must re-partition data and consequently re-induce preconditions and delete effects to match this new partitioning (lines 11-12 of Algorithm 6). We now have a new operator set that is guaranteed to cover the transition τ_{unc} that was initially uncovered. We check whether this new set achieves a lower value for the `coverage` term of our objective, and iterate the above steps until it does.

Finally, after the while loop terminates, we remove all operators from Ω' that have associated datasets that are *empty*. This corresponds exactly to removing operators that are not used in *any* abstract plan suffix computed by COMPUTECOVERAGE and are thus unnecessary for planning.

Proof of termination To see that the main loop of Algorithm 6 is guaranteed to terminate, consider that the operator set Ω' strictly grows larger at every loop iteration (no operators are deleted). Since the predicates are fixed, there is a finite number of possible operators. Thus, at some finite iteration, Ω' will contain every possible operator. At this point, it *must* contain an operator that covers every transition and the loop must terminate.

Anytime Removal of Operators with Null Data In the IMPROVECOVERAGE procedure as illustrated in Algorithm 6, we only prune out operators that do not have any data associated with them after the main while loop has terminated. However, we note here that we can remove such operators from the current operator set (Ω') at any time during the algorithm's loop.

This property arises because *the amount of data associated with a particular operator will only decrease over time*. To see this, note that (1) the number of operators in Ω' only increases over time, and (2) data is assigned to the 'best covering' operator as judged by our heuristic in Equation 3. Given a particular operator ω at some iteration i of the loop, suppose there are d transitions from $\mathcal{D}$ associated with it (i.e, $|\mathcal{D}_\omega| = d$). During future (i.e $> i$) loop iterations, new operators will be added to Ω'. For any of the d transitions in $\mathcal{D}_\omega$, these new operators can either be a worse match (in which case, the transition will remain in $\mathcal{D}_\omega$), or a better match (in which case, the transition will become associated with the new operator). Thus, for any operator ω, once there is no longer any data associated with it, there will *never* be any data associated with it, and it will simply be pruned after the while loop terminates.

As a result, we can prune operators from our current set whenever there is no data associated with them. We do this in our implementation, since it improves our algorithm's wall-clock runtime.

B.2 REDUCECOMPLEXITY

```
REDUCECOMPLEXITY($\Omega$, $\mathcal{D}$, $\mathcal{T}_{train}$)
    $\Omega' \leftarrow$ DELETEOPERATOR($\Omega$)
    $(\mathcal{D}_{\omega_1} \ldots \mathcal{D}_{\omega_m}), (\delta_{\tau_1}, \ldots, \delta_{\tau_j}) \leftarrow$ PARTITIONDATA($\Omega', \mathcal{D}$)
    $\Omega' \leftarrow$ INDUCEPRECANDDELEFFS($\Omega', (\mathcal{D}_{\omega_1} \ldots \mathcal{D}_{\omega_m}),$
        $(\delta_{\tau_1}, \ldots, \delta_{\tau_j})$)
    return $\Omega'$
```

Algorithm 7: Pseudocode for our `reduce-complexity` successor generator. The inputs are a set of operators Ω, the set of all training demonstrations $\mathcal{D}$, and the corresponding set of trainign tasks $\mathcal{T}_{\text{train}}$. The output is a set of operators Ω' such that `complexity`(Ω') $\leq$ `complexity`(Ω).

The pseudocode for our `reduce-complexity` generator is shown in Algorithm 7. As can be seen, the generator is rather simple: we simply delete an operator from the current set (DELETEOPERATOR) and return the remaining operators. Since we've changed the operator set, we must recompute the partitioning and re-induce preconditions and delete effects accordingly.

This generator clearly reduces the `complexity` term from our objective (Equation 1), since $|\Omega'| < |\Omega|$.

C Associating Transitions with Operators

```
FINDBESTCONSISTENTOP(($\Omega$, $s_i$, $s_{i+1}$, $\alpha$, $\mathcal{O}$))
    $\Omega_{\text{con}} \leftarrow \emptyset$
    for $\omega \in \Omega$ do
        for $\underline{\omega} \in$ GETALLGROUNDINGS($\omega, \mathcal{O}$) do
            $s^{\text{pred}}_{i+1} \leftarrow ((s_i \setminus \underline{E^-}) \cup \underline{E^+})$
            if $\underline{P} \subseteq s_i$ AND $\alpha \subseteq s^{pred}_{i+1}$ AND $s^{pred}_{i+1} \subseteq s_{i+1}$ AND $\exists\theta : \underline{C}(\theta) = u_i$ then
                $\Omega_{\text{con}} \leftarrow \Omega_{\text{con}} \cup \underline{\omega}$
    if $\Omega_{con} \neq \emptyset$ then
        return FINDBESTCOVER($\Omega_{con}, (s_i, s_{i+1})$)
```

Algorithm 8: Pseudocode for the FindBestConsistentOp helper method used in Algorithm 1

A key component of our algorithm is the FINDBESTCOVER method from Algorithm 8, which in turn is used in Algorithm 1 and the PARTITIONDATA method of Algorithm 6. The purpose of this method is to associate a transition with a particular operator when multiple operators satisfy the conditions necessary to 'cover' it. Intuitively, we wish to assign a transition to the operator whose prediction best matches the *observed effects* in the transition. We can do this by simply measuring the discrepancy between the operator's add and delete effects, and the observed add and delete effects in the transition. We make two minor changes to this simple measure that are appropriate to our setting. First, we only use the *atomic* delete effects as part of our measure. We exclude the quantified delete effects because these exist in order to enable our operators to decline to predict particular changes in state. Second, we favor operators that correctly predict which atoms *will not* change. Recall that the ENSURENECATOMSSAT method in Algorithm 6 induces such operators by placing the same atoms in the add effects and preconditions.

Given some transition (s_i, u_{i+1}, s_{i+1}), and some ground operator $\underline{\omega}$ with atomic delete effects $\underline{E_\circ^-}$, our heuristic for data partitioning is represented by the score function shown in equation 3.

$$
\begin{aligned}
\underline{\mathcal{K}} &= \underline{E^+} \cap \underline{P} \\
\underline{\mathcal{C}} &= \underline{E^+} \setminus \underline{\mathcal{K}} \\
\text{score} &= |\underline{\mathcal{C}} \setminus (s_{i+1} \setminus s_i)| + \\
&\quad |(s_{i+1} \setminus s_i) \setminus \mathcal{C}| + \\
&\quad |(\underline{E_\circ^-} \setminus (s_i \setminus s_{i+1}))| + \\
&\quad |(s_i \setminus s_{i+1}) \setminus \underline{E_\circ^-}| - \underline{\mathcal{C}}
\end{aligned}
\tag{3}
$$

Once all eligible operators have been scored, we simply pick the lowest-scoring operator to associate with this transition. If multiple operators achieve the same score, we break ties arbitrarily.

D Learning Samplers

In addition to operators, we must also learn samplers to propose continuous parameters for controllers during plan refinement. We directly adapt existing approaches [2, 1] to accomplish this and learn one sampler per operator of the following form: $\sigma(x, o_1, \dots, o_k) = s_\sigma(x[o_1] \oplus \cdots \oplus x[o_k])$, where $x[o]$ denotes the feature vector for o in x, the $\oplus$ denotes concatenation, and s_σ is the model to be learned. Specifically, we treat the problem as one of supervised learning on each of the datasets associated with each operator: $\mathcal{D}_\omega$. Recall that for every transition (x_i, u_{i+1}, x_{i+1}) in $\mathcal{D}_\omega$, we save a mapping $\delta : \overline{v} \to \mathcal{O}_\tau$ from the operator's arguments $\overline{v}$ to objects to ground the operator with. Recall also that every action is a hybrid controller with discrete parameters and continuous parameters θ. To create a datapoint that can be used for supervised learning for the associated sampler, we can reuse this substitution to create an input vector $x[\delta_\tau(v_1)] \oplus \cdots \oplus x[\delta(v_k)]$, where $(v_1, \dots, v_k) = \overline{v}$. The corresponding output for supervised learning is the continuous parameter vector θ in the action u_{i+1}.

Following previous work by Silver et al. [1] and Chitnis et al. [2], we learn two neural networks to parameterize each sampler. The first neural network takes in $x[o_1] \oplus \cdots \oplus x[o_k]$ and regresses to the mean and covariance matrix of a Gaussian distribution over θ. We assume that the desired distribution has nonzero measure, but the covariances can be arbitrarily small in practice. To improve the representational capacity of this network, we learn a second neural network that takes in $x[o_1] \oplus \cdots \oplus x[o_k]$ *and* θ, and returns true or false. This classifier is then used to rejection sample from the first network. To create negative examples, we use all transitions such that the controller used in the transition matches the current controller, but the transition is not in the operator's dataset $\mathcal{D}_\omega$.

E Additional Experiment Details

Here we provide detailed descriptions of each of experiment environments. See (§5) for high-level descriptions and the accompanying code for implementations.

E.1 Screws Environment Details

- **Types:**
  - The `screw` type has features `x`, `y`, `held`.
  - The `receptacle` type has features `x`, `y`.
  - The `gripper` type has features `x`, `y`.
- **Predicates:** `Pickable(?x0:gripper, ?x1:receptacle)`, `AboveReceptacle(?x0:gripper, ?x1:receptacle)`, `HoldingScrew(?x0:gripper, ?x1:screw)`,`ScrewInReceptacle(?x0:screw, ?x1:receptacle)`.
- **Actions:**
  - `MoveToScrew(?x0: gripper, ?x1: screw)`: moves the gripper to be `Near` the screw ?x1.
  - `MoveToReceptacle(?x0: gripper, ?x1: receptacle)`: moves the gripper to be AboveReceptacle(?x0:gripper, ?x1:receptacle)
  - `MagnetizeGripper(?x0: gripper)`: Magnetizes the gripper at the current location, which causes all screws that the gripper is `Near` to be held by the gripper.
  - `DemagnetizeGripper(?x0: gripper)`: Demagnetizes the gripper at the current location, which causes all screws that are being held by the gripper to fall.
- **Goal:** The agent must make `ScrewInReceptacle(?x0:screw, ?x1:receptacle)` true for a particular screw that varies per task.

E.2 Cluttered 1D Environment Details

- **Types:**
  - The `robot` type has features `x`.
  - The `dot` type has features `x`, `grasped`.
- **Predicates:** `NextTo(?x0:robot, ?x1:dot)`, `NextToNothing(?x0:robot)`, `Grasped(?x0:robot, ?x1:dot)`.
- **Actions:**
  - `MoveGrasp(?x0: robot, ?x1: dot, [move_or_grasp, x])`: A single controller that performs both moving and grasping. If `move_or_grasp` < 0.5, then the controller moves the robot to a continuous position `y`. Else, the controller grasps the dot ?x1 if it is within range.
- **Goal:** The agent must make `Grasped(?x0:robot, ?x1:dot)` true for a particular set of dots that varies per task.

E.3 Satellites Environment Details

- **Types:**
  - The `satellite` type has features `x`, `y`, `theta`, `instrument`, `calibration_obj_id`, `is_calibrated`, `read_obj_id`, `shoots_chem_x`, `shoots_chem_y`.
  - The `object` type has features `id`, `x`, `y`, `has_chem_x`, `has_chem_y`.
- **Predicates:** `Sees(?x0:satellite, ?x1:object)`, `CalibrationTarget(?x0:satellite, ?x1:object)`,

`IsCalibrated(?x0:satellite)`, `HasCamera(?x0:satellite)`, `HasInfrared(?x0:satellite)`, `HasGeiger(?x0:satellite)`, `ShootsChemX(?x0:satellite)`, `ShootsChemY(?x0:satellite)`, `HasChemX(?x0:satellite)`, `HasChemY(?x0:satellite)`, `CameraReadingTaken(?x0:satellite, ?x1:object)`, `InfraredReadingTaken(?x0:satellite, ?x1:object)`, `GeigerReadingTaken(?x0:satellite, ?x1:object)`.

- **Actions:**
  - `MoveTo(?x0:satellite, ?x1:object, [x, y])`: Moves the satellite `?x0` to be at `x, y`.
  - `Calibrate(?x0:satellite, ?x1:object)`: Tries to calibrate the satellite `?x0` against object `?x1`. This will only succeed (i.e, make `IsCalibrated(?x0:satellite)` true) if `?x1` is the calibration target of `?x0`.
  - `ShootChemX(?x0:satellite, ?x1: object)`: Tries to shoot a pellet of chemical X from satellite ?x0. This will only succeed if `?x0` both has chemical X and is capable of shooting it.
  - `ShootChemY(?x0:satellite, ?x1:object)`: Tries to shoot a pellet of chemical Y from satellite ?x0. This will only succeed if `?x0` both has chemical Y and is capable of shooting it.
  - `UseInstrument(?x0:satellite, ?x1:object)`: Tries to use the instrument possessed by `?x0` on object `?x1` (note that we assume ?x0 only possesses a single instrument).
- **Goal:** The agent must take particular readings (i.e some combination of `CameraReadingTaken(?x0:satellite, ?x1:object)`, `InfraredReadingTaken(?x0:satellite, ?x1:object)`, `GeigerReadingTaken(?x0:satellite, ?x1:object)`) from a specific set of objects that varies per task.

E.4 Painting Environment Details

- **Types:**
  - The `object` type has features `x`, `y`, `z`, `dirtiness`, `wetness`, `color`, `grasp`, `held`.
  - The `box` type has features `x`, `y`, `color`.
  - The `lid` type has features `open`.
  - The `shelf` type has features `x`, `y`, `color`.
  - The `robot` type has features `x`, `y`, `fingers`.
- **Predicates:** `InBox(?x0:obj)`, `InShelf(?x0:obj)`, `IsBoxColor(?x0:obj, ?x1:box)`, `IsShelfColor(?x0:obj, ?x1:shelf)`, `GripperOpen(?x0:robot)`, `OnTable(?x0:obj)`, `NotOnTable(?x0:obj)`, `HoldingTop(?x0:obj)`, `HoldingSide(?x0:obj)`, `Holding(?x0:obj)`, `IsWet(?x0:obj)`, `IsDry(?x0:obj)`, `IsDirty(?x0:obj)`, `IsClean(?x0:obj)`.
- **Actions:**
  - `Pick(?x0:robot, ?x1:obj, [grasp])`: picks up a particular object, if grasp > 0.5 it performs a top grasp otherwise a side grasp.
  - `Wash(?x0:robot)`: washes the object in hand, which is needed to clean the object.

  - `Dry(?x0:robot)`: drys the object in hand, which is needed after you wash the object.
  - `Paint(?x0:robot, [color])`: paints the object in hand a particular color specified by the continuous parameter.
  - `Place(?x0:robot, [x, y, z])`: places the object in hand at a particular x, y, z location specified by the continuous parameters.
  - `OpenLid(?x0:robot, ?x1:lid)`: opens a specific lid, which is need to place objects inside the box.

- **Goal:** A robot in 3D must pick, wash, dry, paint, and then place various objects in order to get `InBox(?x0:obj)` and `IsBoxColor(?x0:obj, ?x1:box)`, or `InShelf(?x0:obj)` and `IsShelfColor(?x0:obj, ?x1:shelf)` true for particular goal objects.

E.5 Cluttered Painting Environment Details

- **Types:**
  - The `object` type has features `x`, `y`, `z`, `dirtiness`, `wetness`, `color`, `grasp`, `held`.
  - The `box` type has features `x`, `y`, `color`.
  - The `lid` type has features `open`.
  - The `shelf` type has features `x`, `y`, `color`.
  - The `robot` type has features `x`, `y`, `fingers`.

- **Predicates:** `InBox(?x0:obj)`, `InShelf(?x0:obj)`, `IsBoxColor(?x0:obj, ?x1:box)`, `IsShelfColor(?x0:obj, ?x1:shelf)`, `GripperOpen(?x0:robot)`, `OnTable(?x0:obj)`, `NotOnTable(?x0:obj)`, `HoldingTop(?x0:obj)`, `HoldingSide(?x0:obj)`, `Holding(?x0:obj)`, `IsWet(?x0:obj)`, `IsDry(?x0:obj)`, `IsDirty(?x0:obj)`, `IsClean(?x0:obj)`, along with RepeatedNextTo Predicates: `NextTo(?x0:robot, ?x1:obj)`, `NextToBox(?x0:robot, ?x1:box)`, `NextToShelf(?x0:robot, ?x1:shelf)`, `NextToTable(?x0:robot, ?x1:table)`.

- **Actions:**
  - `Pick(?x0:robot, ?x1:obj, [grasp])`: picks up a particular object, if grasp ¿ 0.5 it performs a top grasp otherwise a side grasp.
  - `Wash(?x0:robot)`: washes the object in hand, which is needed to clean the object.
  - `Dry(?x0:robot)`: drys the object in hand, which is needed after you wash the object.
  - `Paint(?x0:robot, [color])`: paints the object in hand a particular color specified by the continuous parameter.
  - `Place(?x0:robot, [x, y, z])`: places the object in hand at a particular x, y, z location specified by the continuous parameters.
  - `OpenLid(?x0:robot, ?x1:lid)`: opens a specific lid, which is need to place objects inside the box.
  - `MoveToObj(?x0:robot, ?x1:obj, [x])`: moves to a particular object with certain displacement x.
  - `MoveToBox(?x0:robot, ?x1:box, [x])`: moves to a particular box with certain displacement x.
  - `MoveToShelf(?x0:robot, ?x1:shelf, [x])`: moves to a particular shelf with certain displacement x.

- **Goal:** A robot in 3D must pick, wash, dry, paint, and then place various objects in order to get `InBox(?x0:obj)` and `IsBoxColor(?x0:obj, ?x1:box)`, or `InShelf(?x0:obj)` and `IsShelfColor(?x0:obj, ?x1:shelf)` true for particular goal objects. In contrast to the previous painting environment, we also need to navigate to the right objects (i.e. all objects are not always reachable from any states). This version of the environment requires operators with ignore effects.

E.6 BEHAVIOR Environment Details

- **Types:**
  - Many `object` types that range from relevant types like `hardbacks` and `notebooks` to many irrelevant types like `toys` and `jars`. All `object` types have features from `location` and `orientation` to `graspable` and `open`. For a complete list of `object` types and features see [5].
- **Predicates:** `Inside(?x0:obj, ?x1:obj)`, `OnTop(?x0:obj, ?x1:obj)`, `Reachable-Nothing()`, `HandEmpty()`, `Holding(?x0:obj)`, `Reachable()`, `Openable(?x0:obj)`, `Not-Openable(?x0:obj)`, `Open(?x0:obj)`, `Closed(?x0:obj)`.
- **Actions:**
  - `NavigateTo(?x0:obj)`: navigates to make a particular object reachable.
  - `Grasp(?x0:obj, [x, y, z])`: picks up a particular object with the hand starting at a particular relative x, y, z location specified by the continuous parameters.
  - `PlaceOnTop(?x0:obj)`: places the object in hand ontop of another object as long as the agent is holding an object and is in range of the object to be placed onto.
  - `PlaceInside(?x0:obj)`: places the object in hand inside another object as long as the agent is holding an object and is in range of the object to be placed into.
  - `Open(?x0:obj))`: opens a specific object (windows, doors, boxes, etc.) if it is ‘openable’.
  - `Close(?x0:obj))`: closes a specific object (windows, doors, boxes, etc.) if it is currently in an ‘open’ state.
- **Goal:** In *Opening Presents*, the robot must `Open(?x0:package)` a number of boxes of type `package` around the room. In *Locking Windows*, the robot must navigate around the house to `Close(?x0:window)` a number of windows. In *Collecting Cans*, the robot must pick up a number of empty soda cans of type `pop` strewn amongst the house and throw them into a trash can of type `bucket`. This will satisfy the goal of getting `Inside(?x0:pop, ?x1:bucket)` for every soda can around the house. In *Sorting Books*, the robot must find books of type `hardback` and `notebook` in a living room and place them each onto a cluttered shelf (i.e. satisfy the goal of `OnTop(?x0:hardback, ?x1:shelf)` and `OnTop(?x0:notebook, ?x1:shelf)` for a number of books).

F Additional Approach Details

Here we provide detailed descriptions of each approach evaluated in experiments. For the approaches that learn operators, we use A^* search with the lmcut heuristic [44] as the high-level planner for bilevel planning in non-BEHAVIOR environments, and use Fast Downward [11] in a configuration with minor differences from `lama-first` as the high-level planner in BEHAVIOR environments, since A^* search was unable to find abstract plans given the large state and action spaces of these tasks. All approaches also iteratively resample until the simulator f verifies that the transition has been achieved, except for GNN Model-Free, which is completely model-free. See (§5) for high-level descriptions and the accompanying code for implementations.

F.1 Ours

- **Operator Learning:** We learn operators via the hill-climbing search described in Section 4.3. For our objective (Equation 1), we set the λ term to be $1/|\mathcal{D}|$, where $|\mathcal{D}|$ represents the number of transitions in the training demonstrations.
- **Sampler Learning:** As described in Section D, each sampler consists of two neural networks: a generator and a discriminator. The generator outputs the mean and diagonal covariance of a Gaussian, using an exponential linear unit (ELU) to assure PSD covariance. The generator is a fully-connected neural network with two hidden layers of size 32, trained with Adam for 50,000 epochs with a learning rate of $1\mathrm{e}{-3}$ using Gaussian negative log likelihood loss. The discriminator is a binary classifier of samples output by the generator. Negative examples for the discriminator are collected from other skill datasets. The classifier is a fully-connected neural network with two hidden layers of size 32, trained with Adam for 10,000 epochs with a learning rate of $1\mathrm{e}{-3}$ using binary cross entropy loss. During planning, the generator is rejection sampled using the discriminator for up to 100 tries, after which the last sample is returned.
- **Planning:** The number of abstract plans for high-level planning was set to $N_{\text{abstract}} = 8$ for our non-BEHAVIOR domains, and $N_{\text{abstract}} = 1$ for our BEHAVIOR domains. The samples per step for refinement was set to $N_{\text{samples}} = 10$ for all environments.

F.2 Cluster and Intersect:

This is the operator learning approach used by Silver et al. [1].

- **Operator Learning:** This approach learns STRIPS operators by attempting to induce a different operator for every set of unique lifted effects (See Silver et al. [1] for more information).
- **Sampler Learning and Planning:** Same as Ours (See (§F.1) for more details).

F.3 LOFT:

This is the operator learning approach used by Silver et al. [3]. We include a version ('LOFT+Replay') that is allowed to mine additional negative data from the environment to match the implementation of the original authors. We also include a version ('LOFT') that is restricted to learning purely from the demonstration data.

- **Operator Learning:** This approach learns operators similar to the Cluster and Intersect baseline, except that it uses search to see if it can modify the operators after performing Cluster and Intersect (See Silver et al. [3] for more information).
- **Sampler Learning and Planning:** Same as Ours (See (§F.1) for more details).

F.4 CI + QE:

A baseline variant of Cluster and Intersect that is capable of learning operators that have quantified delete effects in addition to atomic delete effects.

- **Operator Learning:** This approach first runs Cluster and Intersect, then attempts to induce ignore effects by performing a hill-climbing search over possible choices of ignore effects using prediction error as the metric to be optimized.
- **Sampler Learning and Planning:** Same as Ours (See (§F.1) for more details).

F.5 GNN Shooting:

This approach trains a graph neural network (GNN) [45] policy. This GNN takes in the current state x, abstract state $s = \textsc{Abstract}(x, \Psi_G)$, and goal g. It outputs an action via a one-hot vector over $\mathcal{C}$ corresponding to which controller to execute, one-hot vectors over all objects at each discrete argument position, and a vector of continuous arguments. We train the GNN using behavior cloning on the dataset $\mathcal{D}$. At evaluation time, we sample trajectories by treating the GNN's output continuous arguments as the mean of a Gaussian with fixed variance. We use the known transition model f to check if the goal is achieved, and repeat until the planning timeout is reached.

- **Planning:** Repeat until the goal is reached: query the model on the current state, abstract state, and goal to get a ground skill. Invoke the ground skill's sampler up to 100 times to find a subgoal that leads to the abstract successor state predicted by the skill's operator. If successful, simulate the state forward; otherwise, terminate with failure.
- **Learning:** This approach essentially learns a TAMP planner in the form of a GNN. Following the baselines presented in prior work [2], the GNN is a standard encode-process-decode architecture with 3 message passing steps. Node and edge modules are fully-connected neural networks with two hidden layers of size 16. We follow the method of Chitnis et al. [2] for encoding object-centric states, abstract states, and goals into graph inputs. To get graph outputs, we use node features to identify the object arguments for the skill and a global node with a one-hot vector to identify the skill identity. The models are trained with Adam for 1000 epochs with a learning rate of $1\text{e}{-3}$ and batch size 128 using MSE loss.

F.6 GNN Model-Free:

A baseline that uses the same trained GNN as above, but at evaluation time, directly executes the policy instead of checking execution using f. This has the advantage of being more efficient to evaluate than GNN Shooting, but is less effective.

G Additional Experimental Results and Analyses

Environment	Ours	LOFT	LOFT+replay	CI	CI + QE	GNN
Painting	**69.35 (3.58)**	92.26 (11.41)	135.73 (6.45)	**70.95 (5.07)**	**67.08 (5.86)**	2220.19 (181.29)
Satellites	**19.38 (7.83)**	52.73 (18.35)	438.44 (51.62)	23.29 (5.38)	**15.96 (4.70)**	1625.69 (218.88)
Clutter 1D	**17.98 (1.06)**	68.04 (17.68)	366.89 (146.09)	62.68 (14.89)	28.58 (3.68)	1164.92 (84.74)
Screws	1.31 (0.04)	143.60 (49.10)	5712.80 (736.84)	**0.32 (0.02)**	708.98 (1023.02)	1369.59 (68.44)
Cluttered Satellites	**16.12 (0.55)**	353.67 (52.78)	902.99 (148.22)	107.04 (11.94)	87.24 (10.49)	3043.62 (285.27)
Cluttered Painting	**131.68 (5.05)**	1699.52 (216.71)	7364.03 (532.67)	470.32 (40.38)	2788.74 (1330.38)	4615.70 (334.11)
Opening Presents	**28.91 (11.26)**	106.57 (27.72)	-	100.62 (23.66)	92.63 (17.01)	185.53 (6.63)
Locking Windows	**16.77 (1.55)**	62.55 (10.12)	-	61.71 (8.95)	45.51 (5.74)	319.09 (7.61)
Collecting Cans	3728.73 (9544.75)	1520.93 (354.20)	-	**576.89 (100.57)**	781.38 (350.46)	2121.86 (120.51)
Sorting Books	4981.79 (14460.37)	6423.03 (602.44)	-	**1528.18 (111.18)**	-	5359.99 (170.46)

Table 2: Learning times in seconds on training data for all domains. Note that BEHAVIOR domains (bottom 4) use training set sizes of 10 tasks, while all other domains use training and testing set sizes of 50 tasks. The standard deviation is shown in parentheses.

We have already established that our approach learns operators that lead to more effective *bilevel planning* than baselines. In this section, we are interested in comparing our approach with baselines on three additional metrics: (1) the efficiency of high-level planning using learned operators, (2) the efficiency of the learning algorithm itself, and (3) the simplicity of operator sets we learn. We also run ablations of our method to investigate the importance of optimizing the complexity term, as well as downward-refinability to our method.

Figure 4 shows the nodes created during high-level planning for each of our various environments and operator learning methods. We can see that operators learned by our approach generally lead to comparable or fewer node creations during planning when compared to baselines. In many of the environments where baseline methods are able to achieve a number of points with fewer node cre-

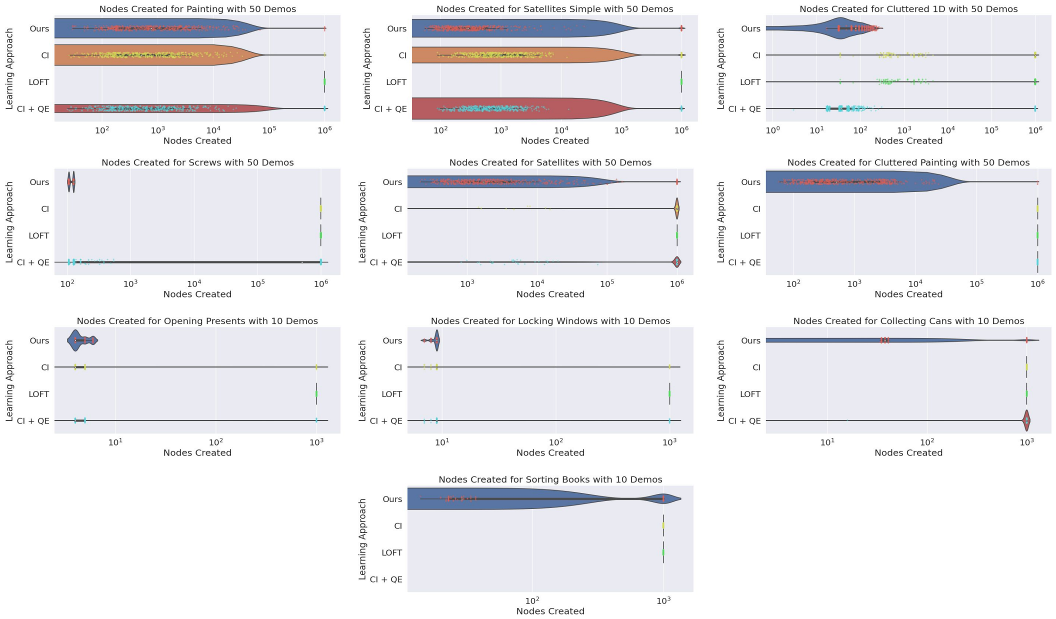

Figure 4: **Nodes Created by Operator Learning Approaches**. We show scatter plots of the nodes created (x-axis) for each operator learning approach (y-axis). We also include a violin graph to visualize the density of points throughout the graph. If bilevel planning failed, we set the nodes created to 10^6 for non-BEHAVIOR domains and 10^3 for BEHAVIOR domains. Our approach achieves a low number of nodes created across when compared to baselines in most domains.

Environment	**Ours**	**LOFT**	**LOFT+replay**	**CI**	**CI + QE**
Painting	**10.00 (0.00)**	13.60 (0.80)	19.20 (0.39)	11.00 (0.00)	10.20 (0.40)
Satellites	**7.40 (0.79)**	10.90 (1.44)	33.80 (3.70)	10.40 (1.20)	9.30 (0.9)
Clutter 1D	**2.00 (0.00)**	7.10 (1.64)	16.10 (2.11)	7.10 (1.64)	3.00 (0.44)
Screws	**4.0 (0.00)**	14.80 (1.98)	91.14 (5.11)	14.80 (1.98)	4.80 (0.97)
Cluttered Satellites	**7.00 (0.00)**	19.80 (2.60)	59.60 (4.45)	16.30 (1.10)	13.9 (0.83)
Cluttered Painting	**13.00 (0.00)**	28.00 (0.00)	157.8 (6.49)	25.20 (2.31)	20.70 (1.61)
Opening Presents	**2.30 (0.90)**	10.80 (2.99)	-	10.80 (2.99)	9.80 (1.83)
Locking Windows	**2.00 (0.00)**	6.10 (0.70)	-	6.10 (0.70)	4.70 (0.64)
Collecting Cans	**6.10 (5.37)**	57.40 (9.43)	-	52.90 (8.41)	13.40 (2.33)
Sorting Books	**14.70 (7.57)**	76.70 (5.62)	-	75.80 (5.79)	-

Table 3: Average number of operators learned for all domains. Note that BEHAVIOR domains (bottom 4) use training set sizes of 10 tasks, while all other domains use training and testing set sizes of 50 tasks. The standard deviation is shown in parentheses.

ations — *Cluttered 1D*, *Opening Presents*, and *Locking Windows* — our method has a significantly higher success rate.

Table 2 shows the learning times for all methods in all domains[2]. Our approach achieves the lowest learning time in 7/10 domains. Upon inspection of our method's performance on the 'Locking Windows' and 'Collecting Cans' domains, we discovered that the high average learning times are because of a few outlier seeds encountering local minima learning, yielding large and complex operator sets (this is the reason for the extremely high standard deviation).

Table 3 shows the number of operators learned for all operator learning methods in all domains. Our approach learns the lowest number of operator sets across all environments and massively out performs other approaches on this metric in *Collecting Cans*, and *Sorting Books*. These results

---

[2] Note that there is no entry for 'CI + QE' for sorting books because learning exceeded the memory limit of our hardware (192 GB)

Environment	Ours	No complexity	Down Eval
Painting	98.80 (1.33)	98.80 (1.33)	26.60 (6.52)
Satellites	93.40 (11.14)	81.20 (19.40)	84.20 (16.88)
Cluttered 1D	100.00 (0.00)	100.0 (0.00)	100.00 (0.00)
Screws	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)
Cluttered Satellites	95.20 (2.40)	94.80 (2.72)	94.00 (3.22)
Cluttered Painting	99.20 (1.33)	99.00 (1.84)	23.80 (7.56)
Opening Presents	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)
Locking Windows	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)
Collecting Cans	77.00 (37.16)	75.00 (38.30)	75.00 (38.80)
Sorting Books	69.00 (36.73)	52.00 (34.00)	67.00 (37.2)

Table 4: Percentage success rate for our original method, as well as ablations where we set the λ parameter from Equation 1 to 0 on test tasks (No complexity), and enforce downward refinability at evaluation time (Down Eval) for all domains. Note that BEHAVIOR domains use training and testing set sizes of 10 tasks, while all other domains use training and testing set sizes of 50 tasks. The percentage standard deviation is shown in brackets.

further highlight our ability to learn operator sets that efficient for high-level planning, but also simpler and therefore, more likely to generalize to new environments.

Table 4 shows the success rate of our method when the λ parameter from Equation 1 is set to 0 (the 'No complexity' column), thereby effectively removing any impact on the optimization from the `complexity` term in the objective. In most environments (Painting, Cluttered 1D, Screws, Cluttered Satellites, Cluttered Painting, Opening Presents, Locking Windows, and Collecting Cans), this change has minimal impact on the method's success rate. However, in two environments (Satellites and Sorting Books), the change causes a significant reduction in success rate. Upon inspection, we found that our approach learned a number of very complex operators in these domains. When λ was set to be greater than 0, our approach would delete a number of these operators, but in this ablation, it was unable to and thus planning performance suffered. We can conclude from these experiments that optimizing the complexity term is a key component of our approach on particular domains.

Table 4 also shows the success rate of our method when bilevel planning is only allowed to produce 1 abstract plan during refinement. This effectively enforces that the learned operators must yield downward-refinable plans. This causes a significant reduction in success rate in many environments, showing that the ability to evaluate multiple abstract plans is important to planning with operators learned by our method. Indeed, all the environments that exhibit significant reductions in success rate do not - to our knowledge - admit a downward-refinable high-level theory over the provided skills and predicates.

H Learned Operator Examples

Finally, we provide operator examples to demonstrate our approaches ability to overcome overfiting to specific situations. Figure 5 shows a comparison of the operators learned with *Open* in *Opening Packages* environment and *NavigateTo* in *Collecting Cans* environment across our approach and 'CI + QE' (the most competitive baseline in these environments). As shown, by optimizing prediction error 'CI + QE' learns a number of operators to describe the same amount of transitions that is covered by the single operator our approach learns. Upon inspection, 'CI + QE' learns overly specific operators when trying to cluster effects that try to predict the entire state to the point where 'Quantified Delete Effects' are not fully utilized. For the full set of operators learned by our algorithm on the 'Sorting Books' task, see Figure 6.

```
Open-package0:
    Arguments: [?x0:package]
    Preconditions: [
        closed-package(?x0:package),
        handempty(),
        openable-package(?x0:package),
        reachable-package(?x0:package)]
    Add Effects: [open-package(?x0:package)]
    Delete Effects: [closed-package(?x0:package)]
    Quantified Delete Effects:
        [ontop-package-room_floor]
    Controller: Open-package(?x0:package),
```

```
Open-package0:
    Arguments: [?x0:package]
    Preconditions: [
        closed-package(?x0:package),
        handempty(),
        openable-package(?x0:package),
        reachable-package(?x0:package)]
    Add Effects: [open-package(?x0:package)]
    Delete Effects: [closed-package(?x0:package)]
    Quantified Delete Effects: []
    Controller: Open-package(?x0:package),

Open-package1:
    Arguments: [?x0:room_floor, ?x1:package]
    Preconditions: [
        closed-package(?x1:package),
        handempty(),
        not-openable-room_floor(?x0:room_floor),
        ontop-package-room_floor(?x1:package, ?x0:room_floor),
        openable-package(?x1:package),
        reachable-package(?x1:package),
        reachable-room_floor(?x0:room_floor)]
    Add Effects: [open-package(?x1:package)]
    Delete Effects: [closed-package(?x1:package),
        ontop-package-room_floor(?x1:package, ?x0:room_floor)]
    Quantified Delete Effects: []
    Controller: Open-package(?x1:package)}
```

```
NavigateTo-pop0:
    Arguments: [?x0:pop]
    Preconditions: [
        handempty(),
        not-openable-pop(?x0:pop)]
    Add Effects: [reachable-pop(?x0:pop)]
    Delete Effects: []
    Quantified Delete Effects: [
        ontop-pop-pop,
        reachable-bed,
        reachable-bucket,
        reachable-pop]
    Controller: NavigateTo-pop(?x0:pop)}
```

```
NavigateTo-pop0:
    Arguments: [?x0:bed, ?x1:pop]
    Preconditions: [
        handempty(),
        not-openable-bed(?x0:bed),
        not-openable-pop(?x1:pop),
        ontop-pop-bed(?x1:pop, ?x0:bed),
        reachable-bed(?x0:bed)]
    Add Effects: [reachable-pop(?x1:pop)]
    Delete Effects: [reachable-bed(?x0:bed)]
    Quantified Delete Effects: []
    Controller: NavigateTo-pop(?x1:pop),

NavigateTo-pop1:
    Arguments: [?x0:pop]
    Preconditions: [
        handempty(),
        not-openable-pop(?x0:pop),
        reachable-pop(?x0:pop)]
    Add Effects: []
    Delete Effects: []
    Quantified Delete Effects: []
    Controller: NavigateTo-pop(?x0:pop),

NavigateTo-pop2:
    Arguments: [?x0:pop]
    Preconditions: [
        handempty(),
        not-openable-pop(?x0:pop)]
    Add Effects: [reachable-pop(?x0:pop)]
    Delete Effects: []
    Quantified Delete Effects: []
    Controller: NavigateTo-pop(?x0:pop),
```

Figure 5: **Operator Comparison**. Operators learned after our approach (left) and ‘CI+QE‘ (right), for *Open* in *Opening Packages* environment (top) and *NavigateTo* in *Collecting Cans* Cans environment. Our approach learns fewer operators that are generally simpler, and thus more conducive to effective bilevel planning and generalization.

```
Grasp-notebook0:
    Arguments: [?x0:notebook]
    Preconditions: [handempty(), not-openable-notebook(?x0:notebook), reachable-notebook(?x0:notebook)]
    Add Effects: [holding-notebook(?x0:notebook)]
    Delete Effects: [handempty(), reachable-notebook(?x0:notebook)]
    Quantified Delete Effects: [ontop-notebook-coffee_table, ontop-notebook-room_floor]
    Controller: Grasp-notebook(?x0:notebook)

NavigateTo-notebook0:
    Arguments: [?x0:notebook]
    Preconditions: [handempty(), not-openable-notebook(?x0:notebook)]
    Add Effects: [reachable-notebook(?x0:notebook)]
    Delete Effects: []
    Quantified Delete Effects: [reachable-board_game, reachable-coffee_table, reachable-hardback, reachable-notebook, reachable-shelf, reachable-video_game]
    Controller: NavigateTo-notebook(?x0:notebook)

PlaceOnTop-shelf0:
    Arguments: [?x0:shelf, ?x1:hardback]
    Preconditions: [holding-hardback(?x1:hardback), not-openable-hardback(?x1:hardback), not-openable-shelf(?x0:shelf), reachable-shelf(?x0:shelf)]
    Add Effects: [handempty(), ontop-hardback-shelf(?x1:hardback, ?x0:shelf)]
    Delete Effects: [holding-hardback(?x1:hardback)]
    Quantified Delete Effects: []
    Controller: PlaceOnTop-shelf(?x0:shelf)

PlaceOnTop-shelf1:
    Arguments: [?x0:shelf, ?x1:notebook]
    Preconditions: [holding-notebook(?x1:notebook), not-openable-notebook(?x1:notebook), not-openable-shelf(?x0:shelf), reachable-shelf(?x0:shelf)]
    Add Effects: [handempty(), ontop-notebook-shelf(?x1:notebook, ?x0:shelf)]
    Delete Effects: [holding-notebook(?x1:notebook)]
    Quantified Delete Effects: []
    Controller: PlaceOnTop-shelf(?x0:shelf)

Grasp-hardback0:
    Arguments: [?x0:hardback]
    Preconditions: [handempty(), not-openable-hardback(?x0:hardback), reachable-hardback(?x0:hardback)]
    Add Effects: [holding-hardback(?x0:hardback)]
    Delete Effects: [handempty(), reachable-hardback(?x0:hardback)]
    Quantified Delete Effects: [ontop-hardback-coffee_table, ontop-hardback-room_floor]
    Controller: Grasp-hardback(?x0:hardback)

NavigateTo-shelf0:
    Arguments: [?x0:shelf]
    Preconditions: [not-openable-shelf(?x0:shelf)]
    Add Effects: [reachable-shelf(?x0:shelf)]
    Delete Effects: []
    Quantified Delete Effects: [ontop-hardback-coffee_table, reachable-board_game, reachable-coffee_table, reachable-hardback, reachable-notebook,
reachable-video_game]
    Controller: NavigateTo-shelf(?x0:shelf)

NavigateTo-hardback0:
    Arguments: [?x0:hardback]
    Preconditions: [handempty(), not-openable-hardback(?x0:hardback)]
    Add Effects: [reachable-hardback(?x0:hardback)]
    Delete Effects: []
    Quantified Delete Effects: [reachable-board_game, reachable-coffee_table, reachable-hardback, reachable-notebook, reachable-shelf, reachable-video_game]
    Controller: NavigateTo-hardback(?x0:hardback)
```

Figure 6: Sorting Books learned operators.