Interactions between tree search and task simplification in planning

Anonymous authors
Double blind review

Abstract

Humans can flexibly solve novel, complex problems by planning. This involves forming an internal model of a task, reasoning about possibilities in that model, and selecting effective actions. However, these cognitive operations are costly and humans have limited cognitive resources, necessitating the use of *resource rational* planning strategies. Here, we analyze the interaction of two broad strategies: heuristic tree search and task simplification. We propose a framework for interleaving tree search with task simplification, present a concrete algorithmic instantiation of this approach, and prove it finds the optimal policy in problems composed of discrete objects where the task is to reach a goal configuration in as few steps as possible. When applied to tasks such as Rush Hour and Sokoban, our approach significantly decreases the total amount of computation required to find a solution. These results reveal how the interaction of resource rational strategies can lead to efficiencies above and beyond the individual strategies.

Keywords: planning; resource rationality; tree search; heuristics; abstraction

Introduction

Imagine selling your nightstand and the buyer has just arrived to pick it up. Now you have to come up with a plan to get the nightstand to their car, but there's many other pieces of furniture blocking a direct route, not to mention your apartment does not provide much space for maneuvering heavy objects. When faced with scenarios involving many moving parts and interactions (e.g., Figure. 1), planning is challenging because the space of possible configurations is vast and people have limited cognitive resources such as time, memory, and attention. This necessitates the use of *resource rational* planning strategies (Lewis et al., 2014; Griffiths et al., 2015).

Both classic and recent work in cognitive science explores the cognitive strategies that people use to efficiently plan in novel situations. Two strategies that have attracted particular attention are *heuristic tree search* (Huys et al., 2012; Callaway et al., 2022; Van Opheusden et al., 2023) and *task simplification* (Ho et al., 2022; da Silva Castanheira et al., 2025; Chen et al., 2026). The intuition behind heuristic tree search is to reduce costs by focusing planning on the most promising parts of a task space (Figure 2A). For example, even though you could move your nightstand into your bathroom and put it in the bathtub, there is no benefit to considering that configuration since it is far from your current state (the nightstand being in the bedroom) and far from your goal (getting the nightstand to the front door). Instead, you should search through the tree of possible states and actions, roughly in the direction of your goal, guided by a heuristic (Newell & Simon, 1972; Keramati et al., 2016).

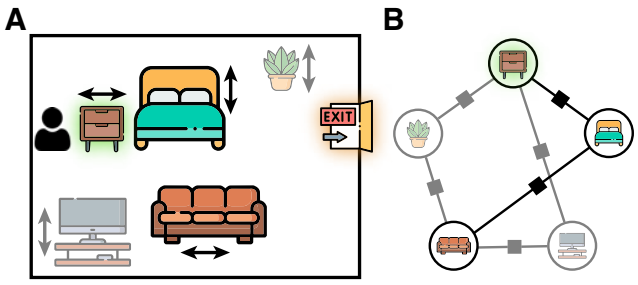

Figure 1: Furniture moving example. **A**: The goal is to move the nightstand to the exit. Furniture may only be moved along the indicated axes, and may not be stacked on top of each other. **B**: Factor graph of interactions between furniture. Grayed out edges indicate interactions which are not important for solving the task.

Task simplification also reduces cognitive costs, but recent work has primarily focused on reducing the complexity of a planning representation, rather than reducing the computations involved in performing planning (Ho et al., 2022). For example, a task representation that includes the nightstand, the bed, the couch, and a TV is more complex than one that includes only the nightstand and bed because it requires maintaining a representation with more entities. Similarly, representing a building as a single, undifferentiated obstacle is easier than representing a building and its individual rooms because the former involves considering strictly fewer details, although the latter representation affords identifying shortcuts through buildings (Ho et al., 2023).

Here, we seek to understand how synergistic interactions between tree search and task simplification can facilitate more efficient planning. Our analysis is motivated by the observation that tree search and task simplification are *nested computational processes*: Selecting a representation of a task constitutes an "outer loop" while planning on a particular representation (e.g., by using heuristic tree search) is an "inner loop". Such outer-inner loop processes are common in metareasoning and resource rational accounts (Lieder & Griffiths, 2020; Bhui et al., 2021), but constitute a fundamental dilemma for the approach because solving meta-problems naïvely is often more computationally intractable than solving a problem directly (Russell & Wefald, 1991; Rich et al., 2020). We propose that at least in the case of task simplification and tree search, this dilemma can sometimes be overcome and may provide more general insight into feasible resource rational strategies.

Specifically, we provide an algorithmic framework for ana-

lyzing how interleaving task simplification/refinement with tree search can significantly reduce the total cost of planning. Inspired by ideas in classical planning (Ghallab et al., 2016) and formal verification (Lee & Seshia, 2016), our approach builds on the idea of *iteratively refining a representation* and *reusing solutions as heuristics*. In our furniture moving example, this approach would suggest that the person first forms a plan to move out the nightstand without any regards to other furniture. Realizing this plan can not be executed, pieces of furniture that are in the way such as the bed would be added to the representation, and a new plan would be formed. Importantly, the information from the previous plan would be used initialize the new plan, allowing the reuse of previous computations.

We instantiate this notion of iterative refinement and reuse in a concrete algorithm, Iterative Construal Refinement (ICR), and provide a mathematical proof that it returns the optimal policy given certain domain structures. We then apply the algorithm two complex planning tasks, Rush Hour and Sokoban, previously used to study human behavior (Jarušek & Pelánek, 2010; Bockholt & Zweig, 2015; Berke et al., 2024; Olieslagers et al., 2025; Chu et al., 2025). We find that when a domain satisfies an admissibility condition, iteratively refining a representation while retaining heuristic information leads to significantly fewer planning operations. Even when the domain does not satisfy this condition, we find that the algorithm can provide computational benefits.

Computational framework

We aim to characterize interactions between tree search and task simplification in a unified formal framework. This section starts by introducing the basic building blocks of our approach, including the standard formulation of sequential decision-making and planning problems in terms of Markov Decision Processes (MDPs) (Sutton et al., 1998), a formalism for task simplification and refinement in terms of compositions of MDPs, and LAO* (Hansen & Zilberstein, 2001), a general-purpose tree search algorithm for MDPs. We then present *Iterative Construal Refinement* (ICR), an algorithm in which solutions to simpler versions of a problem (moving a piece of furniture without taking into account other pieces) are used as heuristics for more complex refinements of the problem (moving a piece of furniture and taking into account all furniture in the house). Finally, we prove that subject to certain constraints on the structure of the MDP and task construals, ICR is guaranteed to converge to the optimal policy.

Simplified Task Representations

We first review some standard definitions (Sutton et al., 1998): An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, R, T \rangle$, where $\mathcal{S}$ is a *state space*, $\mathcal{A}$ is an *action space*, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *reward function*, and $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$[1] is a *transition function*. A *policy*, $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, maps states to distributions over actions, and the *value* of a policy π from a state $s \in \mathcal{S}$ is the expected future cumulative reward, $V^{\pi}(s) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim T(\cdot|s_t,a_t)} \left[\sum_{t=0}^{\infty} R(s_t, a_t) \mid s_0 = s \right]$.

We further assume people plan over a *factored task representation* (Guestrin et al., 2003), in which state spaces, action spaces, transitions, and rewards result from composing simpler *component MDPs*. Formally, a factored task representation is a tuple $\langle I, \mathcal{J}, \{M_i\}_{i \in I}, \{F_j\}_{j \in \mathcal{J}} \rangle$, where:

- I is an index set for component MDPs
- $\mathcal{J}$ is an index set for component interactions
- $\{M_i\}_{i \in I}$ is a set of *component MDPs*. We denote each component MDP M_i and its state space, action space, transition function, and reward functions as $\langle \mathcal{S}_i, \mathcal{A}_i, T_i, R_i \rangle$.
- $\{F_j\}_{j \in \mathcal{J}}$ is a set of *component cause-effect relations*. Each $F_j = \langle I_j, \phi_j \rangle$ is defined by a *scope* $I_j \subseteq I$ representing the component MDPs that interact, and a function $\phi_j : \times_{i \in I_j} \mathcal{S}_i \to [0,1]$ that represents the probability of successfully transitioning to a joint state $s \in \times_{i \in I_j} \mathcal{S}_i$.

A factored representation is useful because it enables us to define a space of simplified *task construals*, ad hoc task representations used for planning (Ho et al., 2022).

Intuitively, a *construal* corresponds to attending to a subset of entities and cause-effect relations in a task. For example, consider the example task of moving out a nightstand (Figure 1), taking into account 4 other objects: $I = \{1,2,3,4\}$ (bed, couch, television, and plant in the example). A construal that includes the agent with the nightstand and 2 other objects (bed and couch, $\mathcal{C} = \{1,2\}$) then induces a *construed task* with its own state space (i.e., states that only include the agent with the nightstand, the bed, and the couch); action space (i.e., actions involving those objects); and causal relationships (i.e., only those that involve interactions between the objects).

Formally, a construal is a subset of component MDP indices, $\mathcal{C} \subseteq I$, and we define a *construed MDP* $M_{\mathcal{C}}$ as consisting of a Cartesian product state space $\mathcal{S}_{\mathcal{C}} = \times_{i \in \mathcal{C}} \mathcal{S}_i$; a union sum action space; $\mathcal{A}_{\mathcal{C}} = \bigcup_{i \in \mathcal{C}} \{(i,a) \mid a \in \mathcal{A}_i\}$; a construed reward function $R_{\mathcal{C}}(s,(i,a)) = R_i(s_i,a)$ where $s \in \mathcal{S}_{\mathcal{C}}$, $(i,a) \in \mathcal{A}_{\mathcal{C}}$, and s_i denotes the component i of s; and a construed transition function:

$$T_{\mathcal{C}}\left(s' \mid s,(i,a)\right) = \left(\phi_{\mathcal{C}}(s') T_i(s'_i \mid s_i, a) + \left(1 - \phi_{\mathcal{C}}(s')\right) \mathbf{1}_{s'_i = s_i}\right) \prod_{k \in \mathcal{C} \setminus \{i\}} \mathbf{1}_{s'_k = s} \tag{1}$$

where $\phi_{\mathcal{C}}(s')$ is the product of cause-effect relations with scopes in the construal $\mathcal{C}$, that is, $\phi_{\mathcal{C}}(s') = \prod_{k \in \{j \in \mathcal{J} \mid I_j \subseteq \mathcal{C}\}} \phi_k(s')$, and $\mathbf{1}_P$ is 1 if P is True and 0 otherwise. In other words, if the product is zero (e.g. an object is overlapped with another), then the transition probability is zero for all s', except $s' = s$. If the product is one, then the transition is applied in the component MDP M_i, leaving the states in the other component MDPs unchanged.

[1] $\Delta(X)$ denotes the set of probability distributions over a set X.

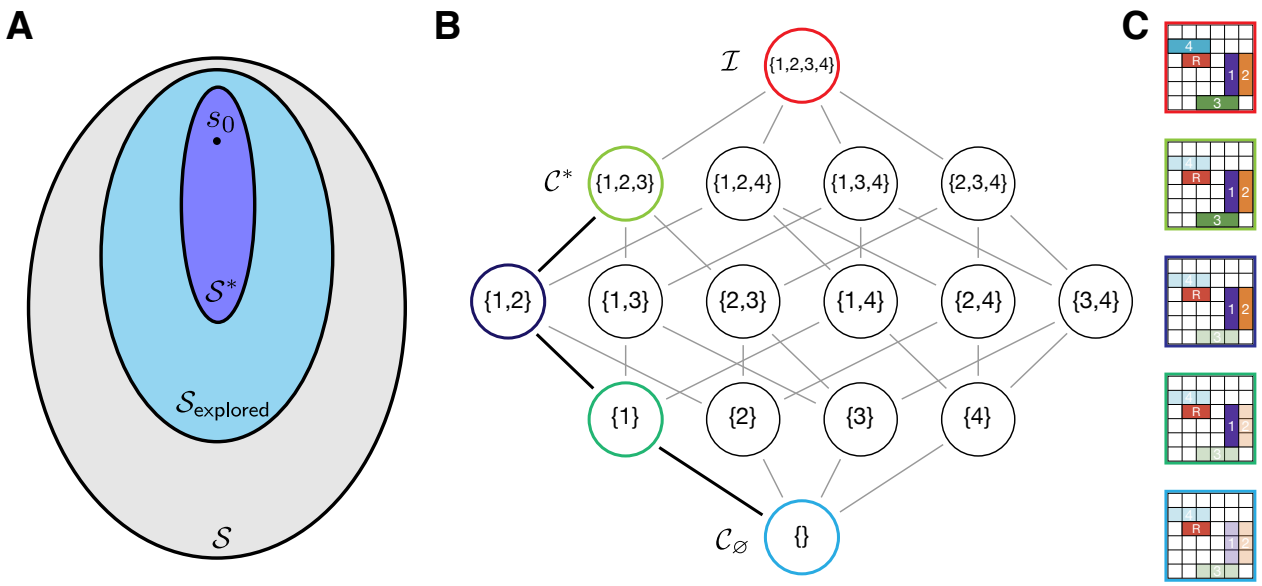

Figure 2: Heuristic tree search (inner loop) and task refinement (outer loop). **A**: Heuristic search state spaces. $\mathcal{S}$ is the set of all states, $\mathcal{S}_{\text{explored}}$ is the set of states explored by the heuristic tree search algorithm, $\mathcal{S}^*$ is the set of states visited by the optimal policy π^* starting at the initial state s_0. A heuristic is useful to the extent that the states explored is smaller than the full state space. **B**: Construal lattice for a problem with 4 construable objects. **C**: Example Rush Hour construals corresponding to vertices in the lattice. $\mathcal{C}^*$ denotes the optimal construal.

Given a construed MDP $M_\mathcal{C}$, its optimal value function at a construed state $s \in \mathcal{S}_\mathcal{C}$ is:

$$V^*_\mathcal{C}(s) = \arg\max_\pi \mathbb{E}_{\pi, T_\mathcal{C}} \left[\sum_{t=0}^{\infty} R_\mathcal{C}(s_t, a_t) \mid s_0 = s \right]$$

and the unique optimal stochastic policy, which chooses uniformly over optimal state-action values, $Q^*_\mathcal{C}(s,a) = R_\mathcal{C}(s,a) + \mathbb{E}_{T_\mathcal{C}}\left[V^*_\mathcal{C}(s')\right]$, is denoted as $\pi^*_\mathcal{C}(a \mid s)$, with $s \in \mathcal{S}_\mathcal{C}$, $a \in \mathcal{A}_\mathcal{C}$. For a given construal $\mathcal{C}$, we refer to $V^*_\mathcal{C}$ and $\pi^*_\mathcal{C}$ as the corresponding *construed value* and *construed policy*, respectively. Additionally, in this work, $V^* = V^*_I$ and $\pi^* = \pi^*_I$, that is, we denote the optimal value function and policy for the full construal I as functions without any subscript.

Partial ordering of construals

Since a construal can be any subset of I, the set of all construals is the powerset of I. This set has a partial ordering, forming a complete, distributive lattice, where nodes represent construals, and edges represent adding or removing an object (Figure 2B). This partial ordering allows us to define the notion of a refined and abstracted construal. A refined construal $\mathcal{C}'$ is a construal with more objects/constraints than an abstracted construal $\mathcal{C}$, where $\mathcal{C} \subseteq \mathcal{C}' \subseteq I$.

A natural question is to ask whether we can solve an abstracted version of the task. An abstracted construal would contain fewer objects and planning might be less computationally expensive. However, if we take this to the extreme, and plan in the empty construal, our plan might not be of any use when we try and execute it because the objects we abstracted might be of importance. Instead, we might want to plan in an abstract construal, and then consider a more refined construal to ensure that our plan can be executed.

The next question that comes up is then, how do construed values and policies relate to the real target of planning: behaving optimally in the actual task (e.g., in the full construal of the problem)? More to the point, how do values in simpler construals relate to values in more complex construals, including those of the fully detailed task?

In general, this relationship can be arbitrary. For example, imagine navigating through a university campus to a specific lecture hall and consider three possible construals of campus of increasing complexity: (A) a construal that represents buildings as impenetrable objects; (B) a construal that also represents the layout of classrooms in buildings; and (C) a construal the also represents which classrooms currently have classes in them. Construals A, B and C are all strictly increasing in complexity in the sense all the details of A are in B and all of those in B are in C. However, the construed values of A, B, and C are not similarly ordered: Under A, one should take a long route around buildings (lower value); under B, one should take a shortcut through the classrooms (higher value); but under C, one should take a long route similar to A (lower value).

This shows that iterative refinements of task construals may produce solutions that lack a systematic relationship to one another. However, in some domains there exists a principled relationship between the *complexity ordering* and *value ordering* over construals, and, moreover, that this ordering can be leveraged to improve the efficiency of heuristic tree search.

We make this intuition more precise by introducing the idea of an *admissible ordering*: a relationship between construals which tells us that the optimal value of a state in a refined construal is less than or equal to that of a more abstract construal. We prove that this is a sufficient condition that guarantees ICR converges to the optimal policy (Supplementary Information).

Iterative Construal Refinement

ICR (Algorithm 1) starts by initializing an empty construal, and plans on this using a heuristic search algorithm with a heuristic initialized at 0 everywhere. This provides the initial value function and policy for the main loop. The main loop consists of four parts: (1) The algorithm adds an object to the construal on every iteration. (2) For every added object, the algorithm plans in the new construal using the lifted value function from the previous iteration as a heuristic. (3) The algorithm tries to add every object one by one until a construal is obtained where the optimal policy is not the same as in the previous abstracted construal. If such a construal is found, the next iteration is started. (4) The algorithm continues adding more objects until the value of the initial state in the current construal is the same as the behavioral utility of the initial state. The behavioral utility $U(\pi_{\mathcal{C}}, s)$ is the value of a state evaluated in the full construal under a given policy. In this case, the policy is the one returned by solving the current construal.

The algorithm requires making two key design choices corresponding to the outer and inner loops: (1) deciding what order to test possible refinements of the current construal by including an additional object, and (2) deciding on a planning algorithm that can use a heuristic to compute the optimal value and policy for a given construal. Here, for simplicity, we use a random refinement order (in the discussion we revisit the possibility of using more intelligent strategies). For the planning algorithm, we needed an algorithm that: (1) guarantees an optimal solution, (2) works on problems with loops and more generally on any MDP, and (3) produces a value function which may be used as a heuristic and can use an existing value function as a heuristic. Three common planning algorithms that satisfy these desiderata are: (1) value/policy iteration (Puterman, 1994), (2) real-time dynamic programming (RTDP) (Barto et al., 1995), and (3) LAO* (Hansen & Zilberstein, 2001). We chose LAO* since value/policy iteration requires enumerating the entire state space while RTDP uses Monte Carlo rollouts to approximate solutions, which makes analysis of costs more difficult. Next, we review the details of this algorithm relevant for ICR.

Algorithm 1: Iterative Construal Refinement

Input: MDP M, index set I, initial state s
Output: Optimal policy π^*

```
/* Initialize                                   */
C ← ∅
V_∅, π_∅ ← HeuristicSearch(M, 0, s_∅)

/* Evaluation outer loop                        */
while V_C(s) ≠ U(π_C, s) do

    /* Improvement inner loop                   */
    for i ∈ I/C do
        C' ← C ∪ {i}
        V_C', π_C' ← HeuristicSearch(M, V_C, s_C')
        if π_C ≠_s π_C' then
            break

    C ← C'

return π_C
```

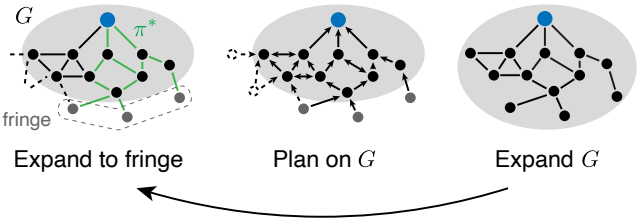

Figure 3: The three core steps in LAO*. Gray area is G, circles are states, blue state is s_0, gray states are fringe states, and dotted states are states outside G not in the fringe.

LAO* (Algorithm 2) works by expanding and solving an explicit graph G until it contains the solution to the problem. This graph consists of the states explored during search ($\mathcal{S}_{\text{explored}}$ in Figure 2A). It starts by adding the initial state s to G and then iterates the following three steps: (1) expand to fringe, (2) plan on G to determine the best actions in G, (3) expand G (Figure 3). In the expand to fringe step, the algorithm does a breadth-first search, only visiting neighboring states that lie on the current optimal policy π^{curr} and lie in G. When a state is reached that does not lie in G, it is added to the fringe F. This steps enumerates all trajectories following π^{curr} from s until states are reached that do not lie in G.

Once the fringe F is obtained, step (2) is to perform dynamic programming (value or policy iteration) on G to obtain a new optimal policy π^{new}. Finally, in step (3), we add all states in the fringe F to G. With this new G and π^{new}, we can go back to step (1) and obtain a new fringe. This continues until the convergence criteria are met. We know the algorithm has converged to the optimal policy when (1) F is empty after expanding to the fringe, and (2) the optimal solution in G has not changed. The latter can be confirmed by checking the value of the starting state $V(s)$.

Since the state spaces of construals are often distinct, we need a way to map a heuristic function over states in one construal $H_{\mathcal{C}}$ to a heuristic function over states in a refinement of that construal $H_{\mathcal{C}'}$. We assign the heuristic value of a state in a refined construal $\mathcal{C}'$ (one with more objects) to be the heuristic value of that state in the abstracted construal $\mathcal{C}$, with the extra objects removed. This is often a many-to-one mapping, since there are often states in $\mathcal{C}'$ where the extra object can

Algorithm 2: HeuristicSearch (LAO*)

Input: MDP $M_{C'}$, heuristic H_C, initial state $s_{C'}$
Output: Optimal construed value function $V_{C'}$ and policy $\pi_{C'}$ from state $s_{C'}$

```
/* Initialize                                          */
G ← {s_C'}
V_C'^curr, π_C'^curr ← Greedy(s_C', H_C)

while True do
    /* (1) Expand to fringe                            */
    F ← ExpandToFringe(M_C', s_C', π_C'^curr, G)

    /* (2) Plan on G                                   */
    V_C'^new, π_C'^new ← ValueIteration(M_C', G, V_C'^curr)

    /* Convergence criteria                            */
    if F = ∅ and V_C'^new(s_C') = V_C'^curr(s_C') then
        break

    /* (3) Expand G                                    */
    G ← G ∪ F
    V_C'^curr, π_C'^curr ← V_C'^new, π_C'^new

return V_C'^new, π_C'^new
```

be in multiple configurations, all mapping to the same state in $\mathcal{C}$ where this object is removed.

With the initial empty construal, the lattice of construals, a way to map heuristics between construal, and an algorithm to plan within a construal, we have all the components to execute the Iterative Construal Refinement algorithm (Algorithm 1). This algorithm solves the resulting construed task using solutions to abstract construals as heuristics for more refined construals. One iteration of the outer loop is visualized in the Supplementary Information.

Correctness of Iterative Construal Refinement

As previously noted, iteratively refined construals do not, in general, produce admissible orderings that produce valid heuristics. However, we can prove a sufficient condition on the structure of a factored MDP that does produce admissible orderings. Our proof relies first on the following definition:

Definition 1 (Lifted task construal)**.** *Given a factored task $\langle I, \mathcal{I}, \{M_i\}_{i\in I}, \{F_j\}_{j\in\mathcal{I}}\rangle$ and construals $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C} \subseteq \mathcal{C}' \subseteq I$, a* construed MDP lifted from $\mathcal{C}$ to $\mathcal{C}'$ *is an MDP $M_{\mathcal{C}\uparrow\mathcal{C}'}$ with state space $\mathcal{S}_{\mathcal{C}\uparrow\mathcal{C}'} = \mathcal{S}_{\mathcal{C}'}$, action space $\mathcal{A}_{\mathcal{C}\uparrow\mathcal{C}'} = \mathcal{A}_{\mathcal{C}'}$, rewards $R_{\mathcal{C}\uparrow\mathcal{C}'} = R_{\mathcal{C}'}$ and transition function :*

$$T_{\mathcal{C}\uparrow\mathcal{C}'}(s' \mid s,(i,a)) \tag{2}$$
$$= \begin{cases} T_{\mathcal{C}}(s'_{\mathcal{C}} \mid s_{\mathcal{C}},(i,a)) \prod_{k\in\mathcal{C}\setminus\mathcal{C}'} \mathbf{1}_{s'_k=s_k} & \text{if } i \in \mathcal{C} \\ T_i(s'_i \mid s_i,a) \prod_{k\in\mathcal{C}'\setminus\{i\}} \mathbf{1}_{s'_k=s_k} & \text{otherwise} \end{cases}$$

Intuitively, lifting a construed MDP corresponds to adding in new entities to a task representation *without including any new interactions or constraints*. This notion of lifting a construed MDP then plays a role in the following definition:

Definition 2 (Preserving optimal values under lifting)**.** *A factored task $\langle I, \mathcal{I}, \{M_i\}_{i\in I}, \{F_j\}_{j\in\mathcal{I}}\rangle$,* preserves optimal values under lifting *if for any construals $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C} \subseteq \mathcal{C}' \subseteq I$, and any $s \in \mathcal{S}_{\mathcal{C}'}$:*

$$V^*_{\mathcal{C}\uparrow\mathcal{C}'}(s) = V^*_{\mathcal{C}}(s_{\mathcal{C}})$$

where $V^*_{\mathcal{C}\uparrow\mathcal{C}'}$ is the optimal value function of a lifted MDP $M_{\mathcal{C}\uparrow\mathcal{C}'}$.

Intuitively, preserving optimal values under lifting means that the optimal value in a construed task is the same as the optimal value in a version of the task that includes additional entities but not their interactions with the original entities. In the supplementary materials, we show that this condition is sufficient for ICR to return the optimal policy, as summarized in the following theorem:

Theorem 1 (Correctness of Iterative Construal Refinement)**.** *If a factored representation $\langle I, \mathcal{I}, \{M_i\}_{i\in I}, \{F_j\}_{j\in\mathcal{I}}\rangle$ preserves optimal values under lifting, then Iterative Construal Refinement outputs the optimal policy from an initial state $s_0 \in \mathcal{S}_I$.*

Simulations

Although ICR is correct given certain task properties, this does not entail that ICR makes heuristic tree search more efficient. To evaluate efficiency, we compared ICR to planning on a full task representation in two domains. The first, Rush Hour, preserves optimal values under lifting, while the second, Sokoban, does not. Our results show that for both these domains, ICR can reduce search costs (as measured by states expanded and Bellman backups). We first describe the tasks before turning to our results.

Tasks

Rush Hour Rush Hour is a sliding block puzzle game, invented in the 1970s by Nob Yoshigahara. The goal is to move a target car (the red car) to the right of the board. The board is a 6×6 grid containing cars that span either two or three tiles. Cars are oriented either horizontally or vertically, and may only be moved forward or backward in the direction they are facing. Cars may not overlap with each other or pass through each other. The player may move any car at any point, given that there is a valid space for it to move to. The difficulty of the game arises from the fact that unblocking a car may result in another car becoming blocked. We chose to study Rush Hour because it is a difficult task where the notion of objects and construals arises naturally. In many Rush Hour puzzles, some cars never need to be considered to solve the puzzle, naturally leading to construed versions of the task. An illustrative example to show how a heuristic from the initial construal is used to plan with LAO* is shown in the Supplementary Information. In Rush Hour, construals always contain the red car. The empty construal contains solely the red target car.

We generated a dataset of 250 Rush Hour puzzles by randomly placing cars on the board, and filtering out puzzles that could not be solved or required fewer than 6 moves to solve. We varied the number of cars on each board from 4 to 8, generating 50 puzzles for each car count. When generating puzzles, we also made sure that no two puzzles were equivalent by investigating their reachable state space.

Sokoban Sokoban is a puzzle game invented by Hiroyuki Imabayashi in 1981 where players must push boxes to specific goal locations. Once all boxes are on a goal location, the puzzle is solved. There is no rule stating that a specific box must be moved onto a specific goal location. The game is played on a grid-world-like board, where the player can only control the movement of the character. The board usually contains walls, which are immovable tiles that cannot be traversed. The difficulty in Sokoban puzzles comes from the fact that boxes may be pushed, but not pulled. A box can not be pushed if the square it is being pushed to is not empty: you cannot push a box that is behind another box or wall. This results in deadlock states: board states where it is impossible for the player to win. Unlike Rush Hour in which an action can always be undone, the push-pull asymmetry results in losing states where the only thing the player can do to make progress is to restart the puzzle. Furthermore, all boxes and goal locations need to be considered to successfully solve a puzzle. Nevertheless, we chose to study Sokoban because we believed heuristics from construed versions of the task may be useful.

We generated a dataset of 150 Sokoban puzzles by randomly placing the character, walls, boxes, and goals on a 5×5 grid, filtering out puzzles that could not be solved or required fewer than 10 moves to solve. We kept the number of walls fixed at 6, and varied the number of boxes and goals from 2 to 4, generating 50 puzzles for each count. We also made sure that every construal in each puzzle is solvable.

Results

We measure computational cost using two metrics. The first counts the number of states expanded during planning to obtain the optimal solution. Expanding a state involves obtaining all the neighboring states. The *number of expanded states* describes how much of the full state space the planning algorithm needed to explore to arrive at the optimal solution. Larger values indicate the algorithm needed to look further before it found a solution, whereas lower values mean the algorithm was able to find the solution without looking at too many possible trajectories. Since we use LAO* as our heuristic search algorithm, the number of expanded states is equal to the size of G (or $\mathcal{S}_{\text{explored}}$ in Figure 2).

The second metric is the *number of Bellman backups*. A Bellman backup is an operation where the value of a state is updated by finding the maximum value of neighboring states, accounting for reward and transition probabilities. It is summarized in the Bellman equation used in many planning algorithms including LAO* (during value iteration). The number of backups represents the amount of updates required to obtain the optimal value function.

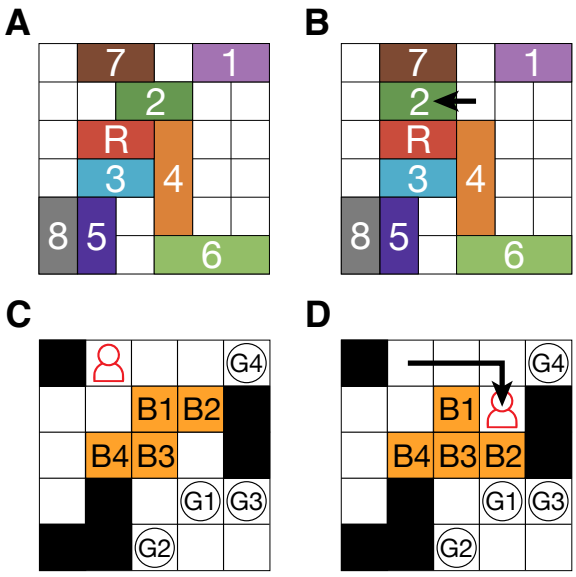

Figure 4: **A**: Rush Hour puzzle and **B**: the optimal move. The goal is to move the red car to the right of the board. One of the shortest solutions is: 2←, 4↑↑, 3→→→, 5↑, 8↑, 6←←←, 4↓↓↓, R→→→. **C**: Sokoban puzzle and **D**: the first 3 optimal moves. The goal is to put all boxes (B1, B2, ...) on a goal (G1, G2, ...). Box B1 does not necessarily have to go on goal G1. The shortest solution involves placing B1 on G3, B2 on G3, B3 on G2, and B4 on G1.

For each puzzle, we ran the iterative refinement algorithm and computed the number of expanded states and Bellman backups for each LAO* call. We summed these quantities across LAO* calls and compared them to running LAO* once on the full construal. We ran the iterative refinement algorithm 10 times for each problem because the algorithm shuffles the order in which construals are considered. This creates a stochastic path through construal space.

We averaged the metrics across these 10 trajectories, and found that the iterative refinement algorithm expands fewer states, both in Rush Hour (Figure 5A, $t(249) = 9.41, p < .001$) and Sokoban (Figure 5C, $t(149) = 5.73, p < .001$). For the number of backups, iterative refinement only provided a benefit when planning in Rush Hour (Figure 5B, $t(249) = 4.75, p < .001$), not in Sokoban (Figure 5D, $t(149) = -3.63, p < .001$).

To find out why this is the case, we looked at the minimum and maximum values of the metrics across the 10 repetitions for each problem. We found that in Sokoban, the path through the space of construals had a bigger effect than in Rush Hour (Figure 5, bottom row). With a more informed construal search algorithm, we expect the Sokoban backups to look more like the Min panels in Figure 5. Another factor that may explain the discrepancy between tasks is that planning in Sokoban requires specifying the problem as an infinite-horizon MDP, with a discount factor $\gamma < 1$. This makes the number of backups

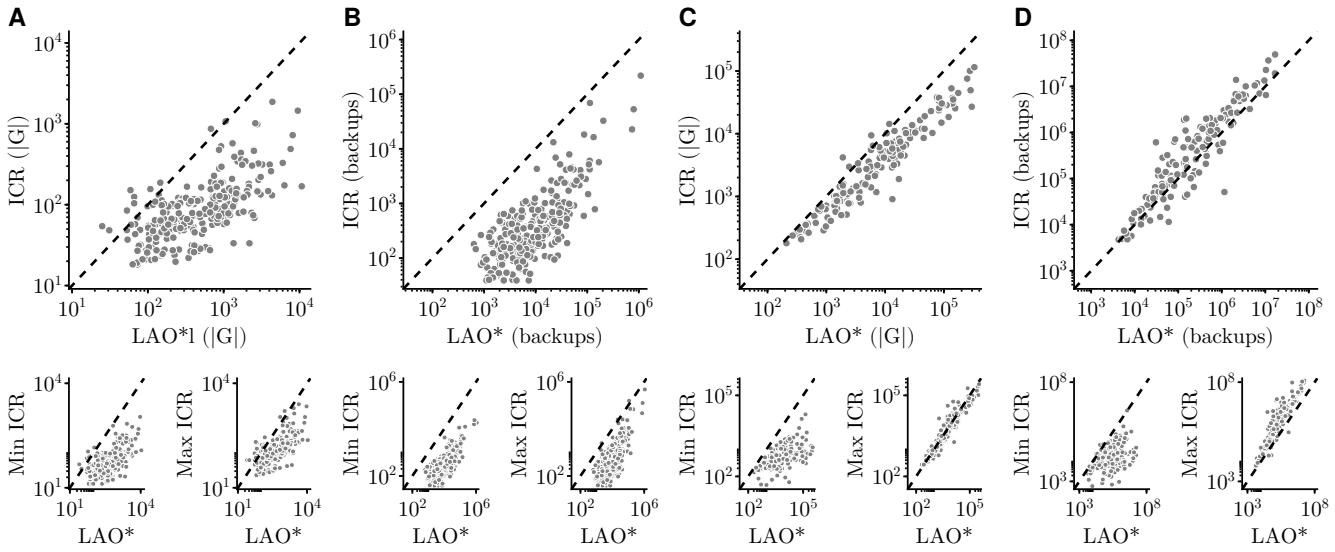

Figure 5: Computational cost of planning in Rush Hour (**A-B**) and Sokoban (**C-D**) with Iterative Construal Refinement (ICR) and LAO* on the full construal using the number of expanded states ($|G|$) and number of backups metrics. Top row are results after averaging 10 trajectories through the construal lattice. Bottom row are results after taking the minimum/maximum of 10 trajectories through the construal lattice.

metric highly dependent on the chosen value of γ as well as the value iteration convergence criterion ε.

Problem features affecting planning cost

We also examined what aspects of a problem affected the efficacy of ICR. For each problem, we calculated four features of complexity: (1) number of objects (cars in Rush Hour, box-goal pairs in Sokoban), (2) difference between number of objects and size of optimal construal, (3) length of optimal solution (in number of moves), and (4) reachable state space size.

Increasing the number of objects in a problem increases the planning cost, and may lead to a small increase in the efficacy of ICR (Figure 6A-B). The issue with number of objects is that it obscures more detailed information about a problem such as how many of these objects actually matter, or how many configurations these objects can be in.

Looking at the number of objects in the optimal construal $\mathcal{C}^*$, the construal with the fewest number of objects that still allows the task to be solved, we found that Iterative Construal Refinement provides larger computational benefits the more objects are not important for solving a problem (Figure 6C-D). This expected result arises from the fact that objects are only added to the construal if they impact the policy. The more objects are unimportant, the shorter the path through the construal lattice, and the less computation is done.

To investigate the configurations of the objects, we computed the size of the reachable state space of each problem. This is the number of states that can be reached when following any trajectory from the start state to a goal state in the full construal. We found that the more configurations are reachable, the greater the computational savings of Iterative Construal Refinement (Figure 6E-F). This speaks to the efficacy of abstract mental representations.

Past research has shown that in Rush Hour, size of the reachable state space is not a good predictor of human solution length, a proxy for how difficult a problem is (Bockholt & Zweig, 2015). Instead, it was found that optimal solution length was the sole significant predictor among a series of graph-based measures. We found that problems with a higher optimal solution length requires more computation to solve, but did not result in greater computational savings from ICR (Figure 6G-H).

Discussion

In this paper, we presented Iterative Construal Refinement (ICR), an algorithm schema for combining task simplification and heuristic tree search to reduce the computational load of planning in tasks with discrete objects. Starting with a factored task representation in which component MDPs can be composed to form simplified task construals, we described how construals can be organized into a partial ordering representing abstraction/refinement relations, and how construal search can be seen as moving through this lattice structure.

In the ICR framework, we start with a simple construal and iteratively refine it, using heuristics from planning in more abstract construals to guide tree search. We show that given a factored task that satisfies certain properties (i.e., that the factored MDP preserves optimal values when lifting to more abstract construals), this iterative process returns the optimal policy in the full task. Intuitively, this properly results in the construal lattice corresponding to an admissible ordering, which can be leveraged by a heuristic tree search inner loop.

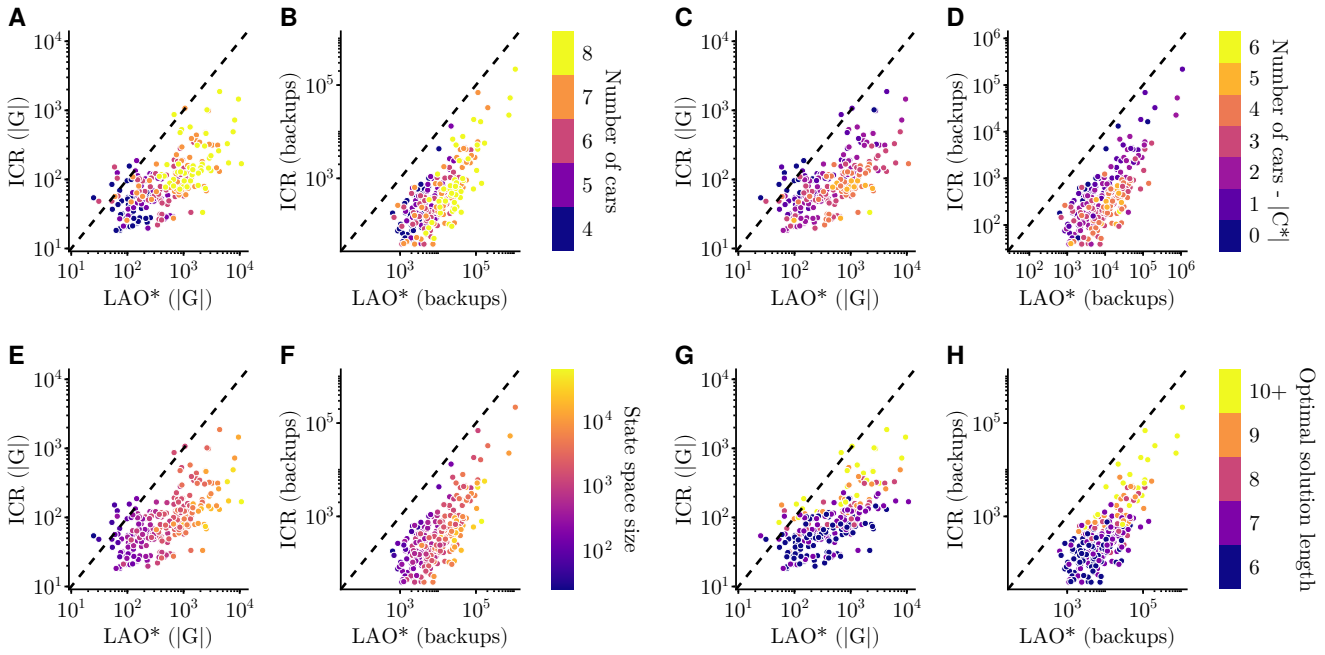

Figure 6: Computational cost of planning in Rush Hour with Iterative Construal Refinement (ICR) and LAO* on the full construal, graded by: **A-B** number of cars in the full construal ($|I|$), **C-D** number of cars not in the optimal construal ($|I| - |\mathcal{C}^*|$), **E-F** reachable state space size, **G-H** optimal solution length. Sokoban results are in the Supplementary Information.

In addition, we tested whether planning over multiple construals while sharing solutions via heuristics reduces the overall costs of planning. To do so, we created a dataset of 250 Rush Hour puzzles and 150 Sokoban puzzles. Running ICR on these problems, and comparing the computational cost to simply planning on the full construal, we found significant computational savings. These were more pronounced in Rush Hour, a task which we prove produces an admissible construal ordering. Even though Sokoban did not satisfy this constraint, we still saw some computational savings, and found that this was highly dependent on the path the algorithm took through the construal lattice.

Taken together, our results suggest that unifying task simplification and heuristic tree search can lead to reduced planning costs, paving the way for models of human planning that more accurately reflect the cognitive computations underlying problem solving. We are excited about several possible future directions opened up by this work.

First, how can the trajectory through the construal lattice be optimized? Here, we do not address this issue since we randomly select next construals, but, as the Sokoban results suggest, the path taken can influence total planning costs. One possibility is to more intelligently refine construals by detecting which additional factors most affect the current (abstract) plan. For example, imagine planning to move the nightstand using an otherwise empty construal but it turns out that the bed is what causes this plan to fail, a more intelligent refinement strategy could detect this possibility. Relatedly, better refinement strategies could add multiple objects simultaneously, potentially further reducing planning costs. Future work could study these traversal strategies to provide a more complete picture, especially in tasks with a larger number of objects, where the construal lattice is large.

Second, do people plan by refining construals? We presented a theoretical framework outlining an algorithmic level explanation (Marr, 1982) of how this may happen in the mind. Simulations show that the algorithm in fact improves the efficiency of tree search, but we do not directly compare to human behavior. In ongoing work, we are exploring ways to test the unique predictions of our account, such as that people will be more likely to use refinement strategies that allow for admissible orderings. More broadly, an important direction for future work is to study how simplification and search interact in complex, naturalistic tasks with qualitatively different compositional properties may support the use of other resource rational planning strategies.

Finally, what is the space of possible psychologically-plausible tree search algorithms given our analyses? We focused on LAO* as it is provably optimal (given an admissible heuristic) and is exact and therefore more amenable to analysis. However, a vast literature in both psychology and neuroscience (Daw et al., 2005; Callaway et al., 2022; Van Opheusden et al., 2023; Kuperwajs et al., 2025) has studied the tree search and simulation strategies that people and other animals use to plan actions. Future work will need to investigate how task simplification may support or constrain these strategies.

References

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial intelligence*, *72*(1-2), 81–138.

Berke, M., Sterling, B., Tenenbaum, A., & Jara-Ettinger, J. (2024). No signatures of first-person simulation in theory of mind judgments about thinking. In *Proceedings of the annual meeting of the cognitive science society* (Vol. 46).

Bhui, R., Lai, L., & Gershman, S. J. (2021). Resource-rational decision making. *Current Opinion in Behavioral Sciences*, *41*, 15–21.

Bockholt, M., & Zweig, K. A. (2015). Why is this so hard? insights from the state space of a simple board game. In *Joint international conference on serious games* (pp. 147–157).

Callaway, F., van Opheusden, B., Gul, S., Das, P., Krueger, P. M., Griffiths, T. L., & Lieder, F. (2022). Rational use of cognitive resources in human planning. *Nature human behaviour*, *6*(8), 1112–1125.

Chen, T., Cheyette, S., Allen, K., Tenenbaum, J., & Smith, K. (2026). "just in time" world modeling supports human planning and reasoning. *arXiv preprint arXiv:2601.14514*.

Chu, J., Zheng, K., & Fan, J. E. (2025). What makes people think a puzzle is fun to solve? In *Proceedings of the annual meeting of the cognitive science society* (Vol. 47).

da Silva Castanheira, J., Shea, N., & Fleming, S. M. (2025). How attention simplifies mental representations for planning. *eLife*, *14*.

Daw, N. D., Niv, Y., & Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, *8*(12), 1704–1711.

Ghallab, M., Nau, D., & Traverso, P. (2016). *Automated planning and acting*. Cambridge University Press.

Griffiths, T. L., Lieder, F., & Goodman, N. D. (2015). Rational Use of Cognitive Resources: Levels of Analysis Between the Computational and the Algorithmic. *Topics in Cognitive Science*, *7*(2), 217–229.

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, *19*, 399–468.

Hansen, E. A., & Zilberstein, S. (2001). Lao$*$: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*(1-2), 35–62.

Ho, M. K., Abel, D., Correa, C. G., Littman, M. L., Cohen, J. D., & Griffiths, T. L. (2022). People construct simplified mental representations to plan. *Nature*, *606*(7912), 129–136.

Ho, M. K., Cohen, J. D., & Griffiths, T. L. (2023). Rational simplification and rigidity in human planning. *Psychological Science*.

Huys, Q. J., Eshel, N., O'Nions, E., Sheridan, L., Dayan, P., & Roiser, J. P. (2012). Bonsai trees in your head: how the pavlovian system sculpts goal-directed choices by pruning decision trees. *PLoS computational biology*, *8*(3), e1002410.

Jarušek, P., & Pelánek, R. (2010). Human problem solving: Sokoban case study. *Technická zpráva, Fakulta informatiky, Masarykova univerzita, Brno*.

Keramati, M., Smittenaar, P., Dolan, R. J., & Dayan, P. (2016). Adaptive integration of habits into depth-limited planning defines a habitual-goal–directed spectrum. *Proceedings of the National Academy of Sciences*, *113*(45), 12868–12873.

Kuperwajs, I., Russek, E. M., Mattar, M. G., Ma, W. J., & Griffiths, T. L. (2025). Looking deeper into the algorithms underlying human planning. *Trends in Cognitive Sciences*.

Lee, E. A., & Seshia, S. A. (2016). *Introduction to embedded systems: A cyber-physical systems approach*. MIT press.

Lewis, R. L., Howes, A., & Singh, S. (2014). Computational rationality: Linking mechanism and behavior through bounded utility maximization. *Topics in Cognitive Science*, *6*(2), 279-311.

Lieder, F., & Griffiths, T. L. (2020). Resource-rational analysis: Understanding human cognition as the optimal use of limited computational resources. *Behavioral and brain sciences*, *43*, e1.

Marr, D. (1982). *Vision: a computational investigation into the human representation and processing of visual information*. W.H. Freeman.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Olieslagers, J., Bnaya, Z., & Ma, W. J. (2025). A computational model of backward reasoning in human problem solving. *PsyArXiv*. Retrieved from `https://osf.io/10.31234/osf.io/h9bsu_v2`

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc.

Rich, P., Blokpoel, M., de Haan, R., & van Rooij, I. (2020). How intractability spans the cognitive and evolutionary levels of explanation. *Topics in cognitive science*, *12*(4), 1382–1402.

Russell, S., Norvig, P., & Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, *25*(27), 79–80.

Russell, S., & Wefald, E. (1991). Principles of metareasoning. *Artificial intelligence*, *49*(1-3), 361–395.

Sutton, R. S., Barto, A. G., et al. (1998). *Reinforcement learning: An introduction* (Vol. 1) (No. 1). MIT press Cambridge.

Van Opheusden, B., Kuperwajs, I., Galbiati, G., Bnaya, Z., Li, Y., & Ma, W. J. (2023). Expertise increases planning depth in human gameplay. *Nature*, *618*(7967), 1000–1005.

Supplementary Information

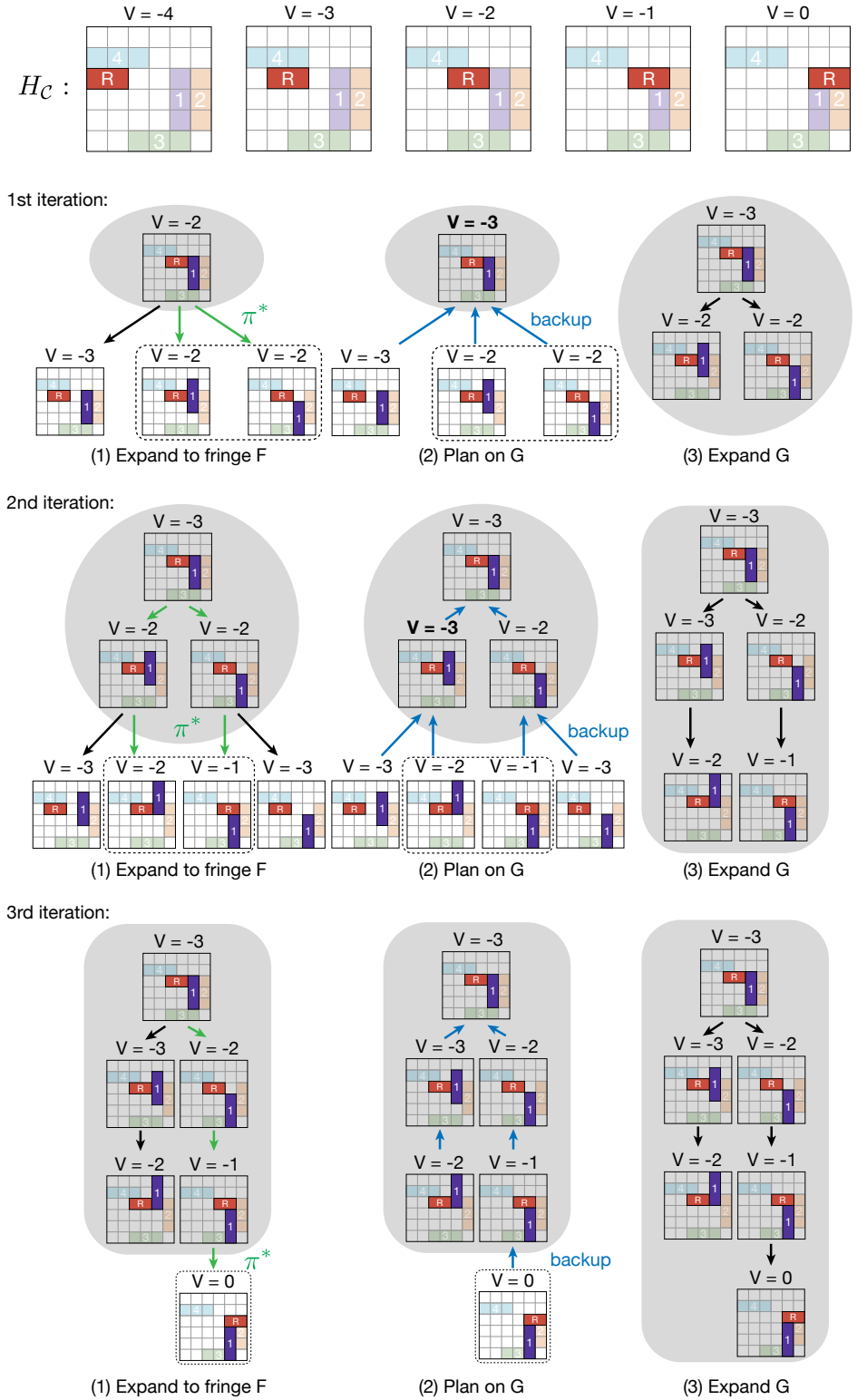

Figure 7: LAO* on an example Rush Hour puzzle, where the previous was the empty construal, only containing the red car, and the current construal contains the purple cars. Gray area is G, dotted box is F, green arrows are the optimal policy $\pi_{\mathcal{C}'}$, blue arrows are Bellman backups done during value iteration. After 3 iterations, expanding to fringe yields an empty F and the value of the initial state remains at -3.

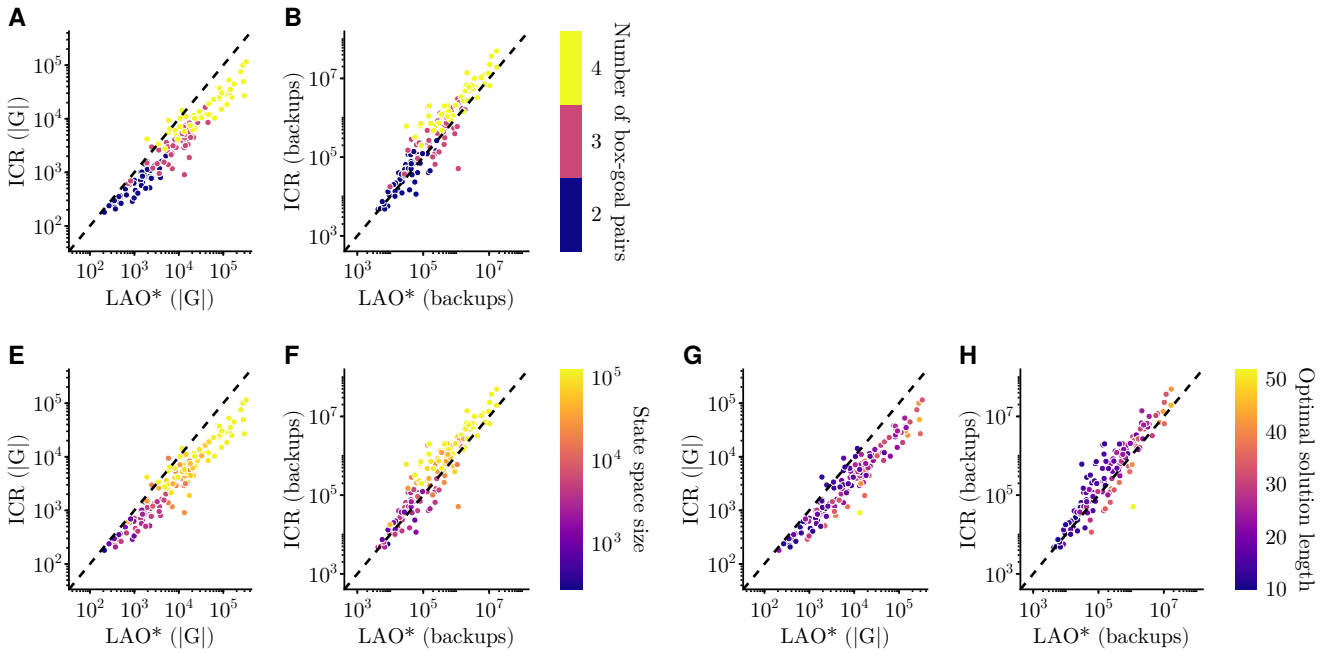

Figure 8: Computational cost of planning in Sokoban with Iterative Construal Refinement (ICR) and LAO* on the full construal, graded by: **A-B** number of box-goal pairs in the full construal ($|I|$), **C-D** the optimal construal is always the full construal, so we exclude the analysis of the difference between the two, **E-F** reachable state space size., **G-H** optimal solution length.

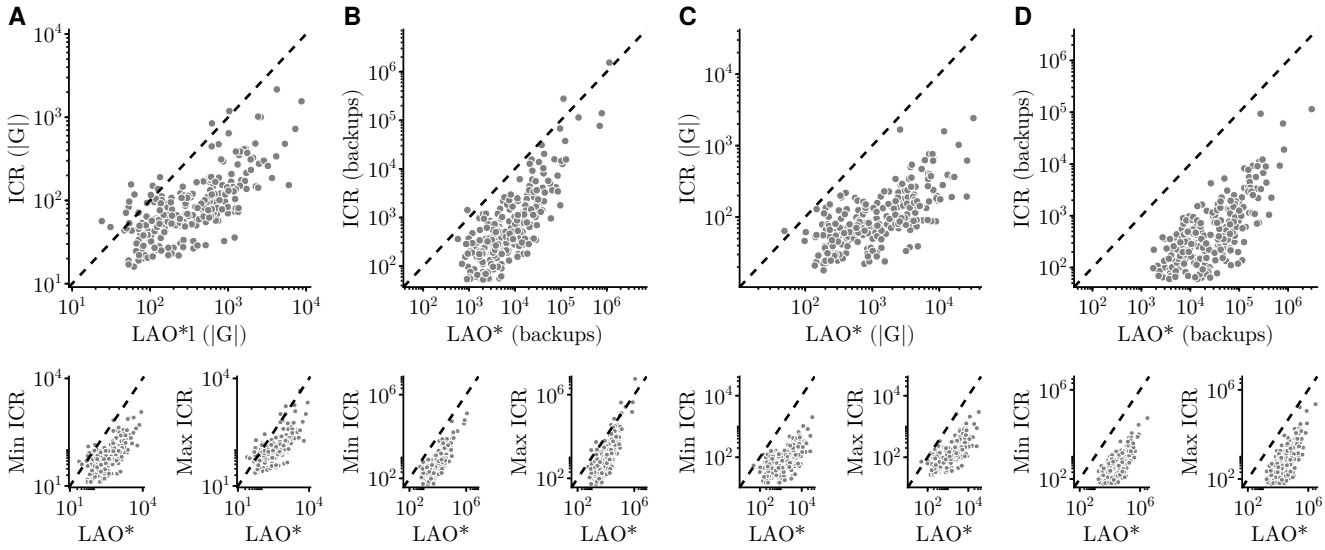

Figure 9: Computational cost of planning in Rush Hour with stochastic transitions (**A-B**, 10% chance that action does not execute) and a larger grid size (**C-D**, 7x7 instead of 6x6) using Iterative Construal Refinement (ICR) and LAO* on the full construal using the number of expanded states ($|G|$) and number of backups metrics. Top row are results after averaging 10 trajectories through the construal lattice. Bottom row are results after taking the minimum/maximum of 10 trajectories through the construal lattice.

Proof of Theorem 1

Admissible orderings

The first step to prove Theorem 1 is to show that the heuristic value functions are admissible heuristics. A heuristic h is admissible if $h(s) \leqslant h^*(s)$ for all s (Russell et al., 1995). In this formulation, h is usually thought of as a cost function. Since value is the

negative of cost, we have to reverse the inequality to get the following theorem:

Theorem 2. (Admissibility of Abstract Optimal Values) *Given a factored task which preserves values under lifting for any construal $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C} \subseteq \mathcal{C}'$,*

$$V^*_{\mathcal{C}}(s_{\mathcal{C}}) \geqslant V^*_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'})$$

for all $s_{\mathcal{C}\uparrow\mathcal{C}'} \in \mathcal{S}_{\mathcal{C}'}$.

Proof. For all $s_{\mathcal{C}\uparrow\mathcal{C}'} \in \mathcal{S}_{\mathcal{C}'}$,

$$\begin{aligned}
V^*_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) &= V^{\pi^*_{\mathcal{C}'}}_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) && \text{[Def. of optimal policy]}\\
&= V^{\pi^*_{\mathcal{C}'}}_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) && \text{[Identical transitions/rewards]}\\
&\leqslant V^*_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) && \text{[Def. of optimality]}\\
&= V^*_{\mathcal{C}}(s_{\mathcal{C}}) && \text{[Definition 2]}
\end{aligned}$$

□

Given that the value functions of abstract construals are admissible heuristic, any construal visited by Iterative Construal Refinement will produce an optimal policy for that construal. This is because LAO* has access to an admissible heuristic, guaranteeing convergence to the optimal policy (Hansen & Zilberstein, 2001).

Stopping criterion

What remains is to proof that the optimal policy for the final visited construal $\pi^*_{\mathcal{C}^{\text{final}}}$ is the same as the optimal policy on the full construal π^*_I. Iterative Construal Refinement last visits the construal which causes the value of the initial state under the optimal policy in the construal to be equal to the behavioral utility of the initial state $V^*_{\mathcal{C}^{\text{final}}}(s_{\mathcal{C}^{\text{final}}}) = U(\pi^*_{\mathcal{C}^{\text{final}}}, s_{\mathcal{C}^{\text{final}}})$. To calculate the behavioral policy, we evaluate the policy $\pi^*_{\mathcal{C}^{\text{final}}}$ in the full construal I, $U(\pi^*_{\mathcal{C}^{\text{final}}}, s_{\mathcal{C}^{\text{final}}}) = V^{\pi^*_{\mathcal{C}^{\text{final}}}}_I(s_{\mathcal{C}^{\text{final}}\uparrow I})$. We complete the proof by showing that the value of the initial state $s_{\mathcal{C}^{\text{final}}\uparrow I}$ under the optimal policy of the final construal, evaluated in the full construal, is the same as the value under the optimal policy of the full construal $V^{\pi^*_{\mathcal{C}^{\text{final}}}}_I(s_{\mathcal{C}^{\text{final}}\uparrow I}) = V^{\pi^*_I}_I(s_{\mathcal{C}^{\text{final}}\uparrow I})$, which would imply that $\pi^*_{\mathcal{C}^{\text{final}}} = \pi^*_I$.

Theorem 3. (Lifted Executability Criterion) *Given a factored task which preserves values under lifting for any construal $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C} \subseteq \mathcal{C}'$, and a state $s_{\mathcal{C}\uparrow\mathcal{C}'} \in \mathcal{S}_{\mathcal{C}'}$, if*

$$V^*_{\mathcal{C}}(s_{\mathcal{C}}) = V^{\pi^*_{\mathcal{C}}}_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}),$$

then

$$V^*_{\mathcal{C}}(s_{\mathcal{C}}) = V^*_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}).$$

Proof. First, note that

$$\begin{aligned}
V^*_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) &\geqslant V^{\pi^*_{\mathcal{C}\uparrow\mathcal{C}'}}_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) && \text{[Def. of optimality]}\\
&= V^{\pi^*_{\mathcal{C}}}_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) && \text{[Def. 2]}\\
&= V^*_{\mathcal{C}}(s_{\mathcal{C}}) && \text{[Assumed in theorem]}
\end{aligned}$$

Since $V^*_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) \geqslant V^*_{\mathcal{C}}(s_{\mathcal{C}})$ and, via Theorem 2, $V^*_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) \leqslant V^*_{\mathcal{C}}(s_{\mathcal{C}})$, it follows that $V^*_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) = V^*_{\mathcal{C}}(s_{\mathcal{C}})$. □

Since $\mathcal{C}^{\text{final}} \subseteq I$, we have that $V^*_{\mathcal{C}^{\text{final}}}(s_{\mathcal{C}^{\text{final}}}) = V^*_I(s_{\mathcal{C}^{\text{final}}\uparrow I})$. Due to Definition 2, this implies $V^{\pi^*_{\mathcal{C}^{\text{final}}}}_{I^{\text{final}}}(s_{\mathcal{C}^{\text{final}}\uparrow I}) = V^*_I(s_{\mathcal{C}^{\text{final}}\uparrow I})$ which in turn implies $\pi^*_{\mathcal{C}^{\text{final}}} = \pi^*_I$, completing the proof of Theorem 1.

Admissible ordering of Rush Hour and Sokoban

For our algorithm to guarantee convergence to the optimal policy, the tasks it is used on must produce construals that have an admissible ordering. As mentioned in Defenition 2, this is achieved if the factored task preserves optimal values under lifting.

In Rush Hour, any rewards collected in a construal $\mathcal{C}$ equal those generated by lifting states into a more refined construal $\mathcal{C}'$. Furthermore, moving any car that is not the red target car results in a negative reward. Lifting a construed MDP from $\mathcal{C}$ to $\mathcal{C}'$ does not consider the cause-effect relationships between the objects not in $\mathcal{C}$ (Definition 1), which means any cars not in $\mathcal{C}$ can not block any cars in $\mathcal{C}$. If there are no cars blocking a policy π, then the value of that policy remains unchanged, hence $V^*_{\mathcal{C}\uparrow\mathcal{C}'}(s) = V^*_{\mathcal{C}}(s_{\mathcal{C}})$ for any $s \in \mathcal{S}_{\mathcal{C}'}$. This proof that Rush Hour has an admissible ordering because of its reward structure is outlined in more detail in Lemma 2. Since the red car is always present (even in the empty construal), this Lemma holds and we have proved that Rush Hour has an admissible ordering.

For Sokoban, things are different. Since the number of boxes and goals is equal, and the puzzle is only solved when *all* boxes are on a goal, any construal that is not the full construal I is not solvable. One way around this is to assume a construal is solved if all construed boxes are on a construed goal location. The issue with this is that the reward structure is no longer equal when a construal is lifted. This means that Lemma 1 and Lemma 2 do not hold, and hence, there is no guarantee of convergence to the optimal policy. As an example, consider a construal consisting of one box and one goal. The optimal policy to this construal may not be admissible for a construal with two boxes and two goals, because the original box may now have to be moved to the newly added goal. The original goal state of box 1 being on goal 1 is hence no longer a valid goal state, since box 1 must be on goal 2, and box 2 on goal 1. The result is that some paths through the construal lattice in Sokoban will not have an admissible ordering. While a subset of the lattice may have an admissible ordering, there is no guarantee that every full lattice is. This likely explains why Sokoban does not benefit from Iterative Construal Refinement when counting the number of backups. Refining a construal may lead the agent astray, assigning goal states which are not representative of the full task.

Lemma 1. *If, for a given task, any rewards collected in a construal $\mathcal{C}$ equal those generated by lifting states into a more refined construal $\mathcal{C}'$, $\mathcal{C} \subseteq \mathcal{C}'$. That is, $R_{\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}, a) = R_{\mathcal{C}}(s_{\mathcal{C}}, a)$, then lifting a policy $\pi_{\mathcal{C}}$ to $\mathcal{C}'$ (denoted by $\pi_{\mathcal{C}}^{\uparrow\mathcal{C}'}$) gives*

$$V^{\pi^*_{\mathcal{C}}}_{\mathcal{C}}(s_{\mathcal{C}}) = V^{\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}}_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}).$$

Proof. We can use Definition 1 to define a lifted transition function $T_{\mathcal{C}\uparrow\mathcal{C}'}$ which only includes constraints $F_j = (I_j, f_j)$ with a scope $I_j \subseteq \mathcal{C}$, meaning no constraint has a component index $i \in \mathcal{C}'/\mathcal{C}$ in it.

If we denote a policy in MDP $M_{\mathcal{C}}$ as $\pi_{\mathcal{C}}$, then we can define a lifted version of that policy as $\pi_{\mathcal{C}}^{\uparrow\mathcal{C}'}$. This is in contrast to $\pi_{\mathcal{C}\uparrow\mathcal{C}'}$ which defines a policy in MDP $M_{\mathcal{C}\uparrow\mathcal{C}'}$ without lifting a previous policy. Since the transitions and rewards in $M_{\mathcal{C}\uparrow\mathcal{C}'}$ under $\pi_{\mathcal{C}}^{\uparrow\mathcal{C}'}$ are identical to those in $M_{\mathcal{C}}$ under $\pi_{\mathcal{C}}$ (see Definition 1), it follows that $V^{\pi_{\mathcal{C}}^{\uparrow\mathcal{C}'}}_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}) = V^{\pi_{\mathcal{C}}}_{\mathcal{C}}(s_{\mathcal{C}})$.

Denote the optimal policy $\pi^*_{\mathcal{C}}$ for MDP $M_{\mathcal{C}}$ lifted to $M_{\mathcal{C}\uparrow\mathcal{C}'}$ as $\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}$. That is, $\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}(a \mid s_{\mathcal{C}\uparrow\mathcal{C}'}) = \pi^*_{\mathcal{C}}(a \mid s_{\mathcal{C}})$. For all states $s_{\mathcal{C}'} \in S_{\mathcal{C}'}$, it is the case that $V^{\pi^*_{\mathcal{C}}}_{\mathcal{C}}(s_{\mathcal{C}}) = V^{\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}}_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'})$.

□

Lemma 2. *If, for a given task, any rewards collected in a construal $\mathcal{C}$ equal those generated by lifting states into a more refined construal $\mathcal{C}'$, $\mathcal{C} \subseteq \mathcal{C}'$, and all rewards from actions associated with the construed object $i = \mathcal{C}'/\mathcal{C}$ are negative, then the optimal value function of the lifted MDP $M_{\mathcal{C}\uparrow\mathcal{C}'}$ is the same as the optimal value function of the abstracted MDP $M_{\mathcal{C}}$*

$$V^*_{\mathcal{C}}(s_{\mathcal{C}}) = V^*_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}).$$

Proof. We start the proof by checking whether $\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}$ is the optimal policy for $M_{\mathcal{C}\uparrow\mathcal{C}'}$, $\pi^*_{\mathcal{C}\uparrow\mathcal{C}'}$. First, the action space of $M_{\mathcal{C}\uparrow\mathcal{C}'}$, namely $\mathcal{A}_{\mathcal{C}'}$, can be partitioned into the actions found in $M_{\mathcal{C}}$, namely $\mathcal{A}_{\mathcal{C}}$, and those unique to $M_{\mathcal{C}\uparrow\mathcal{C}'}$, namely $\mathcal{A}_{\mathcal{C}'/\mathcal{C}}$. For a given state $s_{\mathcal{C}'} \in \mathcal{S}_{\mathcal{C}'}$, it is clear that actions chosen by the lifted policy, $\{a \mid \pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}(a \mid s_{\mathcal{C}'}) > 0\}$, have greater value than any other actions in $\mathcal{A}_{\mathcal{C}}$ since $\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}$ is derived from the optimal policy for $M_{\mathcal{C}}$. Additionally, for any action $a \in \mathcal{A}_{\mathcal{C}'/\mathcal{C}}$, its value will be

$$Q^{\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}}_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}}, s_{\mathcal{C}'/\mathcal{C}}, a) = R_{\mathcal{C}}(s_{\mathcal{C}'}, a) + V^{\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}}_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}}, s'_{\mathcal{C}'/\mathcal{C}}).$$

Since action $a \in \mathcal{A}_{\mathcal{C}'/\mathcal{C}}$ has no effect on $s_{\mathcal{C}}$, and $s'_{\mathcal{C}'/\mathcal{C}}$ is abstracted away in $\mathcal{C}$, we have that $V^{\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}}_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}}, s'_{\mathcal{C}'/\mathcal{C}}) = V^{\pi^*_{\mathcal{C}}}_{\mathcal{C}}(s_{\mathcal{C}})$. Incorporating the result from Lemma 1, the value of the next state is the same as the current state:

$$V^{\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}}_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}}, s'_{\mathcal{C}'/\mathcal{C}}) = V^{\pi^*_{\mathcal{C}}}_{\mathcal{C}}(s_{\mathcal{C}}) = V^{\pi^{*\uparrow\mathcal{C}'}_{\mathcal{C}}}_{\mathcal{C}\uparrow\mathcal{C}'}(s_{\mathcal{C}\uparrow\mathcal{C}'}).$$

Since we assumed rewards associated with $a \in \mathcal{A}_{C'/C}$ are negative, $R_C(s_{C'}, a) < 0$, this means that $Q_{C\uparrow C'}^{\pi_C^{*\uparrow C'}}(s_C, s_{C'/C}, a) < V_{C\uparrow C'}^{\pi_C^{*\uparrow C'}}(s_{C\uparrow C'})$. Thus, every $a \in \mathcal{A}_{C'/C}$ makes the policy worse.

Since for all states $s_{C'} \in \mathcal{S}_{C'}$, every action in $\mathcal{A}_C / \{a \mid \pi^*_{C\uparrow C'}(a \mid s_{C'}) > 0\}$ and $\mathcal{A}_{C'/C}$ makes the policy worse, $\pi_C^{*\uparrow C'}$ satisfies the Bellman optimality equations for $M_{C\uparrow C'}$. That is,

$$V^*_{C\uparrow C'}(s_{C'}) = V_{C\uparrow C'}^{\pi_C^{*\uparrow C'}}(s_{C'}).$$

Since we know from Lemma 1 that

$$V_{C\uparrow C'}^{\pi_C^{*\uparrow C'}}(s_{C\uparrow C'}) = V_C^{\pi^*_C}(s_C)$$

and by definition that

$$V_C^{\pi^*_C}(s_C) = V^*_C(s_C)$$

it follows that

$$V^*_{C\uparrow C'}(s_{C\uparrow C'}) = V^*_C(s_C).$$

□