Name: Barathkrishna Satheeshkumar
Andrew ID: bsathees

Approach

My goal was to implement a domain-independent symbolic task planner capable of solving a wide range of environments. The planner parses the input environment to extract symbols, initial and goal conditions, and action templates. I then generate all grounded actions based on these templates and available symbols. Invalid actions (e.g., ones that reference non-existent conditions) are filtered out dynamically by checking their preconditions at runtime.

The planner uses A*-like search and maintains:

- **Open set**: a priority queue ordered by f = g + h.
- **Closed set**: a hash set to prevent revisiting states.
- **Graph**: a record of all visited states to support backtracking.

Once the goal state is reached, the planner reconstructs the action sequence by tracing back through a parent map. Each symbolic action in the plan is then translated into its grounded counterpart using the action map.

Heuristic Design: Empty-Delete-List Heuristic

I implemented a relaxed planning heuristic based on the **empty-delete-list assumption**—assuming that actions only have positive effects. The heuristic is calculated by counting how many goal conditions are still unsatisfied in the current state. I simulate forward progression using only the add effects of actions and ignore deletes.

✅ **Admissible**: This heuristic never overestimates the cost because it assumes no achieved condition will be undone, which makes the estimated path to the goal cheaper or equal to reality.

✅ **Consistent**: Since applying any valid action under this assumption does not undo any goal progress, the heuristic difference between states is always ≤ the cost of the action (which is 1), ensuring consistency.

This heuristic balances simplicity with effectiveness, providing significant performance improvements without sacrificing optimality.

Planner Guarantees

- ✔️ **Completeness**: Guaranteed by the A* search process with cycle avoidance.
- ✔️ **Domain-Independence**: The planner does not rely on any environment-specific logic.
- ✔️ **Optimality**: Ensured by the use of an admissible and consistent heuristic.

Implementation Details

Planner interface:

```
C/C++
symbolic_planner sp(symbols, initial_conditions, goal_conditions, actions);
```

Key functions:

- `generateActions(...)`: Grounds all valid symbolic actions.
- `isGoal(...)`: Checks if a state satisfies all goal conditions.
- `heuristic(...)`: Counts unmatched goal conditions, ignoring deletes.
- `get_applicable_actions(...)`: Filters actions whose preconditions are met.

State Representation:

```
C/C++
struct state {
	std::unordered_set<Condition, ConditionHasher, ConditionComparator>
	condition;
	int g = std::numeric_limits<int>::max();
	int h = std::numeric_limits<int>::max();
	bool closed = false;
	state(std::unordered_set<Condition, ConditionHasher, ConditionComparator>
	condition_) : condition(condition_) {}
		bool operator==(const state& rhs) const {
		return this->condition == rhs.condition;
	}
};
```

I defined a custom hasher (`stateHasher`) and equality comparator (`stateComparator`) to store `state` objects in hash-based data structures such as `unordered_set` and `unordered_map`.

Compilation Instructions

```
Unset
g++ planner.cpp -o planner.exe
./planner.exe environment.txt
```

Results

Environment	Heuristic	Plan Length	States Expanded	Total Time (ms)
Blocks	✅ Yes	3	17	2
	❌ No	3	82	6
BlocksTriangle	✅ Yes	6	559	189
	❌ No	6	6824	2152
FireExtinguisher	✅ Yes	21	735	380
	❌ No	21	770	443
TowersOfHanoi (described below)	✅ Yes	7	62	40
	❌ No	7	104	68

Explanation of Reported Statistics

The table above summarizes planner performance across four environments, comparing runs with and without heuristics.

- **Plan Length**: This represents the number of actions in the final plan. Since the heuristic used is admissible and consistent, plan length remains the same in all cases, ensuring optimality is preserved.
- **States Expanded**: This reflects the size of the search graph—how many unique states were explored before reaching the goal. The heuristic significantly reduces this number, especially in larger environments (e.g., BlocksTriangle: from 6824 to 559), demonstrating improved efficiency.
- **Total Time**: Includes planner initialization, action map generation, and search. While small environments like *Blocks* show only slight improvements, environments with large action spaces like *BlocksTriangle* and *FireExtinguisher* benefit substantially from heuristic guidance, reducing both state expansion and overall time.
- **TowersOfHanoi**: This custom environment was designed to test the planner's generalization capabilities. It confirms that the heuristic consistently reduces planning effort even in domains not provided by default.

These results show that the planner maintains optimal plans while substantially improving performance when heuristics are applied.

Extra Credit - TowersOfHanoi Environment

Symbols: `Disk1, Disk2, Disk3, PegA, PegB, PegC`

Initial Conditions: `Disk(Disk1), Disk(Disk2), Disk(Disk3), Peg(PegA), Peg(PegB), Peg(PegC), On(Disk1, Disk2), On(Disk2, Disk3), On(Disk3, PegA), Clear(Disk1), Clear(PegB), Clear(PegC), Smaller(Disk1, Disk2), Smaller(Disk2, Disk3), Smaller(Disk1, Disk3)`

Goal Conditions: On(Disk1, Disk2), On(Disk2, Disk3), On(Disk3, PegC)

Actions:

```
MoveToPeg(d, x, p)
Preconditions: Disk(d), Peg(p), Clear(d), Clear(p), On(d, x)
Effects: On(d, p), Clear(x), !On(d, x), !Clear(p)

MoveToDisk(d, x, target)
Preconditions: Disk(d), Disk(target), Clear(d), Clear(target), Smaller(d, target), On(d, x)
Effects: On(d, target), Clear(x), !On(d, x), !Clear(target)
```

This environment was added to the planner as a configuration file and tested using both heuristic and non-heuristic planners. The planner successfully solves the environment optimally and efficiently, with the heuristic again reducing search space without affecting correctness.

Discussion: Heuristic Pros and Cons

✅ Pros

- **Improved Efficiency**: Heuristic dramatically reduces the number of explored states.
- **Faster Planning**: Significantly lower planning time, especially in larger environments.
- **No Loss of Optimality**: Plans remain optimal due to admissibility and consistency.
- **Generic and Lightweight**: Works across domains with no extra domain-specific logic.

❌ Cons

- **Simplistic in Some Domains**: The heuristic doesn't account for delete effects, which may matter in tightly coupled tasks.
- **Possibly Misleading**: Can misguide the planner in domains with negative side effects or dependencies.
- **Heuristic Overhead**: Requires separate relaxed state updates for evaluation.