

Another Searching Problem!

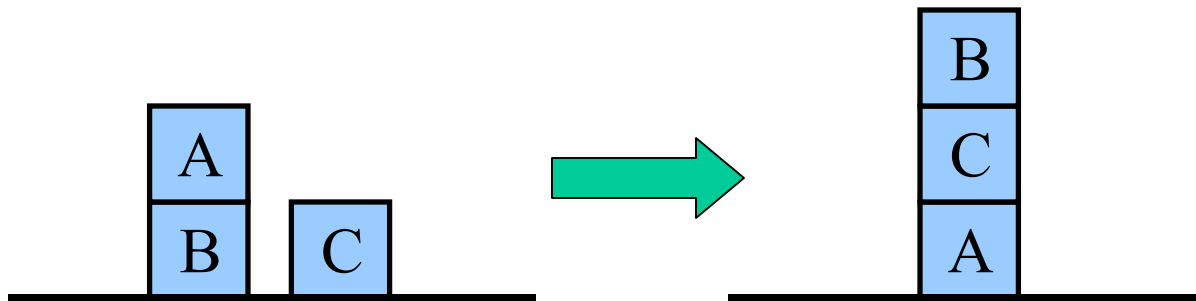
- Given an initial situation, how does one find the set of actions that will change it to achieve certain goals?
 - different from reasoning or plain search, in that a set of goal conditions is specified, not a particular goal state or states.
- This kind of specification of goal conditions yields a different representation of goal states and intermediate actions between states
- Typical condition for these search domains:
 - too large to explore thoroughly
 - despite huge search space, only certain actions really need to be considered at any stage in the solution
 - need intuition of helpfulness and causality of actions to determine steps towards best solution

Planning

- Solution to these kinds of problems is called a **plan**
- Requires representation of world in question into a minimal set of relevant characteristics
 - **states**: represented as a conjunction of positive literals, all **grounded** (no variables) and **function-free** (literal components). States operate under **closed-world assumption**, meaning that only true characteristics are mentioned (all others are false)
 - **goals**: represented as a **partially-specified state**, as a conjunction of literals that corresponds to any number of possible goal states
 - **actions**: specified as a pairing of **preconditions** (characteristics that must be true before the action can happen) and **postconditions** (the effect on various characteristics after the action takes place). Also called an **action schema**.
- This representation is also called STRIPS (**ST**anford **R**esearch **I**nstitute **P**roblem **S**olver) → generally accepted planning format

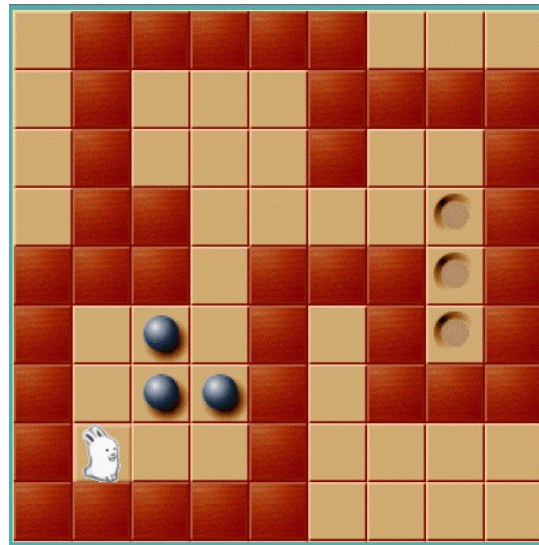
Planning Domains

- Blocks World
 - Given initial layout of blocks on a table, figure out steps to move from initial layout to finish layout, assuming that:
 1. only one block may be moved at a time
 2. only one block may rest on top of another
 3. the table may hold any number of blocks



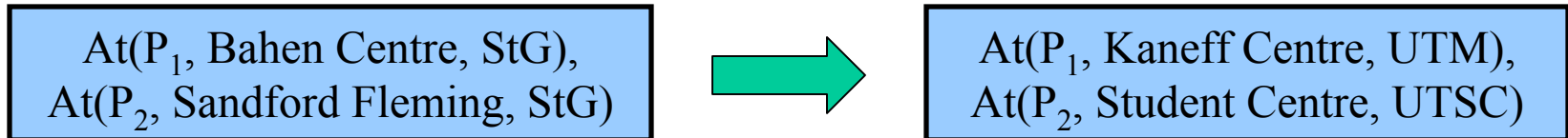
Planning Domains (cont'd)

- Sokoban
 - given initial layout of crates and destination squares, move all the crates to destination squares, given that:
 - all crates must be pushed from behind by avatar
 - crates can only move to a square if that square is empty



Planning Domains (cont'd)

- Logistics
 - concerned with transport of goods from a building in one city to another building in another city. Larger-scale logistics problems involve multiple shipments using the minimal number of trips possible.



- very practical and marketable application of planning
- finding minimal set of actions is NP-hard

Forward Searching

- Finding solution using brute-force search involves successor functions that blindly match actions to the current state to produce the next possible states
- **Forward search** (also called **progression planning**) starts at the initial state and explores all possible states from there until goal state is reached
 - similar to forward chaining in reasoning
- Problem with this approach:
 - can end up exploring actions and paths that are irrelevant in achieving goal
 - bogs down quickly without good heuristic (branching factors are usually high using brute force)

Backwards Search

- Searching backwards (**regression planning**) guarantees that goal will be reached by starting from goal state
- If the planning algorithm only considers **relevant actions** in exploring the state space, then the branching factor for the exploration is usually several times smaller going backwards than going forwards
- Searching approaches (iterative deepening, A^* , bidirectional search) can then be applied to this problem to uncover the steps in the plan

STRIPS Representation

- STRIPS is typically written in terms of a conjunction of literals, like the following for Blocks World:

Start state:

$$\text{On}(A,B) \wedge \text{On}(B,\text{Table}) \\ \wedge \text{On}(C, \text{Table})$$

Goal state:

$$\text{On}(B,C) \wedge \text{On}(C,A) \wedge \\ \text{On}(A, \text{Table})$$

Move (A, Table) :

$$\text{Precondition: } \neg\text{On}(B,A) \wedge \neg\text{On}(C,A) \\ \text{Effect: } \text{On}(A,\text{Table}) \wedge \neg\text{On}(A,B) \wedge \neg\text{On}(A,C)$$

Result of Move (A, Table) :

$$\text{On}(A,\text{Table}) \wedge \text{On}(B,\text{Table}) \wedge \\ \text{On}(C, \text{Table})$$

STRIPS Representation (cont'd)

- STRIPS can also be represented as a vector of boolean values, representing the characteristics of the system
- Vector values can either be 0 (required false), 1 (required true) or * (unknown/irrelevant/don't care)

Start state:

A	A	A	B	B	B	C	C	C
B	C	T	A	C	T	A	B	T
1	0	0	0	0	1	0	0	1

Goal state:

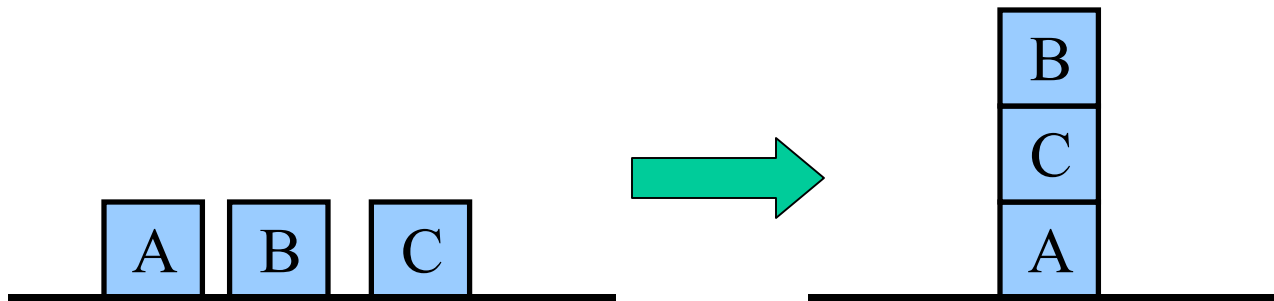
A	A	A	B	B	B	C	C	C
B	C	T	A	C	T	A	B	T
*	*	1	*	1	*	1	*	*

Move (A, Table) :

	A	A	A	B	B	B	C	C	C
	B	C	T	A	C	T	A	B	T
Pre:	*	*	*	0	*	*	0	*	*
Post:	0	0	1	*	*	*	*	*	*

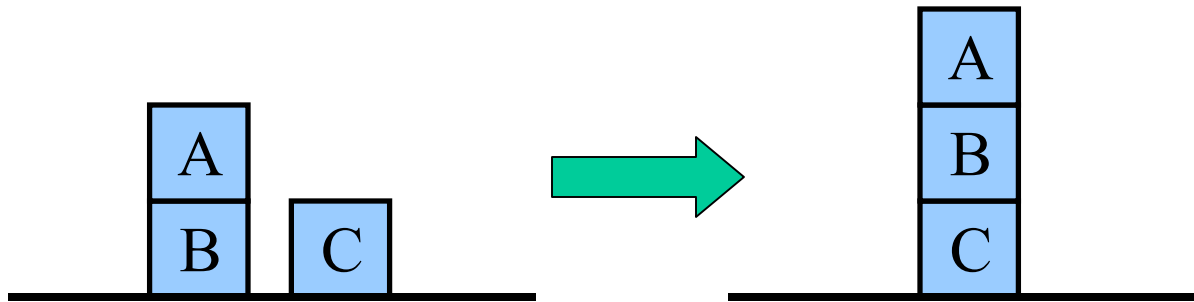
Heuristic Search

- Typical heuristic for planning domains is **Hamming distance** (number of characteristics that are mismatched between current state and goal)
- Leads to certain problems with finding goal:
 - interfering subgoals: B on C interferes with C on A

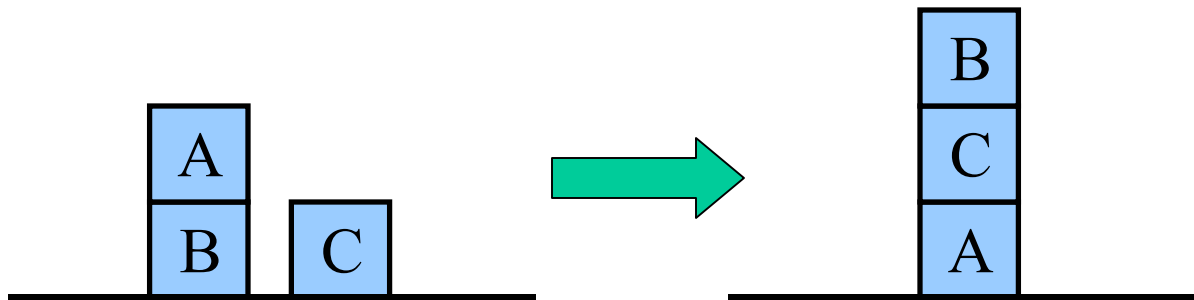


Heuristic Search Problems

- Other problems:
 - Undoing satisfied goals: $A \rightarrow$ Table reduces heuristic gain

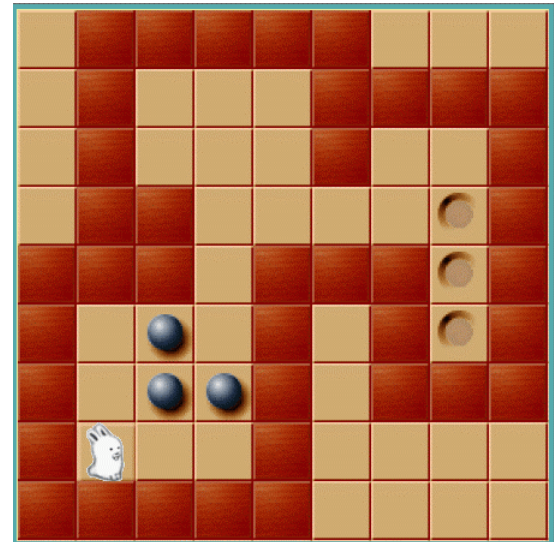


- Avoid satisfying subgoals (“Sussman Anomaly”): $C \rightarrow A$ is a bad move that actually improves the heuristic



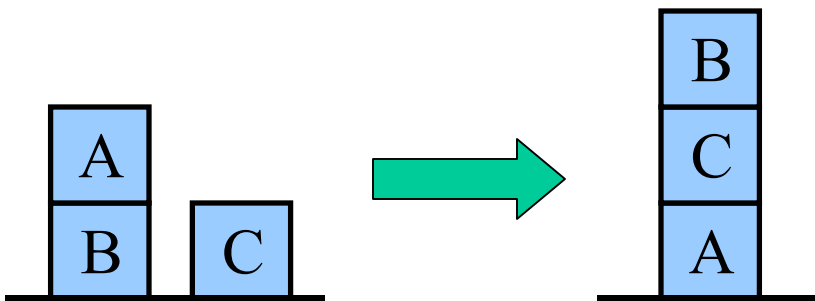
Partial-Order Planners

- In the case of Sokoban, achieving the solution involves achieving smaller, relatively distinct goals
 - right-most crate should be moved first, followed by either of the two crates on the left
- Problem with this: Backtracking
 - could solve for the right-most crate multiple times, in the case where that subgoal is not explored first
 - assuming we know how to identify and achieve these subgoals, how do we know how to order them?



Partial-Order Planning

- Instead of backtracking, add actions that help satisfy the preconditions of the goals, and shift their order to resolve any conflicts
 - if conflicts can't be resolved, then backtracking is necessary. However, this approach significantly reduces the amount of backtracking used, usually altogether.
- Example: Sussman Anomaly



$$\text{Start} = \frac{}{\text{On}(A,B) \wedge \text{On}(B,\text{Table}) \wedge \text{On}(C,\text{Table})}$$

$$\text{End} = \frac{}{\text{On}(B,C) \wedge \text{On}(C,A) \wedge \text{On}(A,\text{Table})}$$

$$\text{Move}(A,\text{Table}) = \frac{\neg\text{On}(C,A) \wedge \neg\text{On}(B,A)}{\text{On}(A,\text{Table}) \wedge \neg\text{On}(A,B) \wedge \neg\text{On}(A,C)}$$

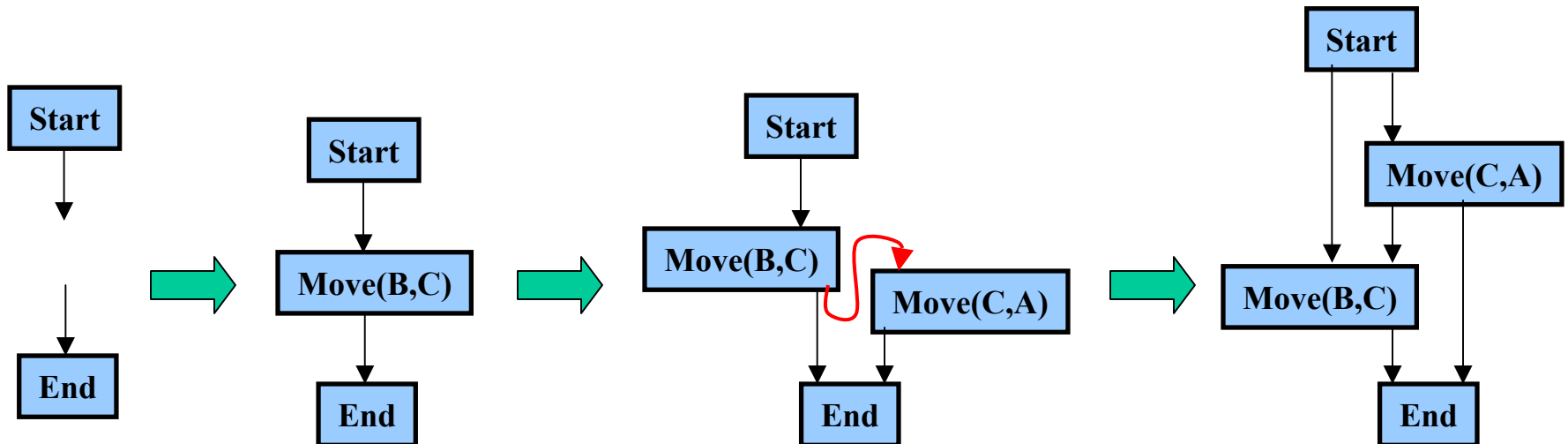
$$\text{Move}(C,A) = \frac{\neg\text{On}(A,C) \wedge \neg\text{On}(B,C) \wedge \neg\text{On}(B,A)}{\text{On}(C,A) \wedge \neg\text{On}(C,B) \wedge \neg\text{On}(C,\text{Table})}$$

$$\text{Move}(B,C) = \frac{\neg\text{On}(A,B) \wedge \neg\text{On}(C,B) \wedge \neg\text{On}(A,C)}{\text{On}(B,C) \wedge \neg\text{On}(B,A) \wedge \neg\text{On}(B,\text{Table})}$$

Partial-Order Planning (cont'd)

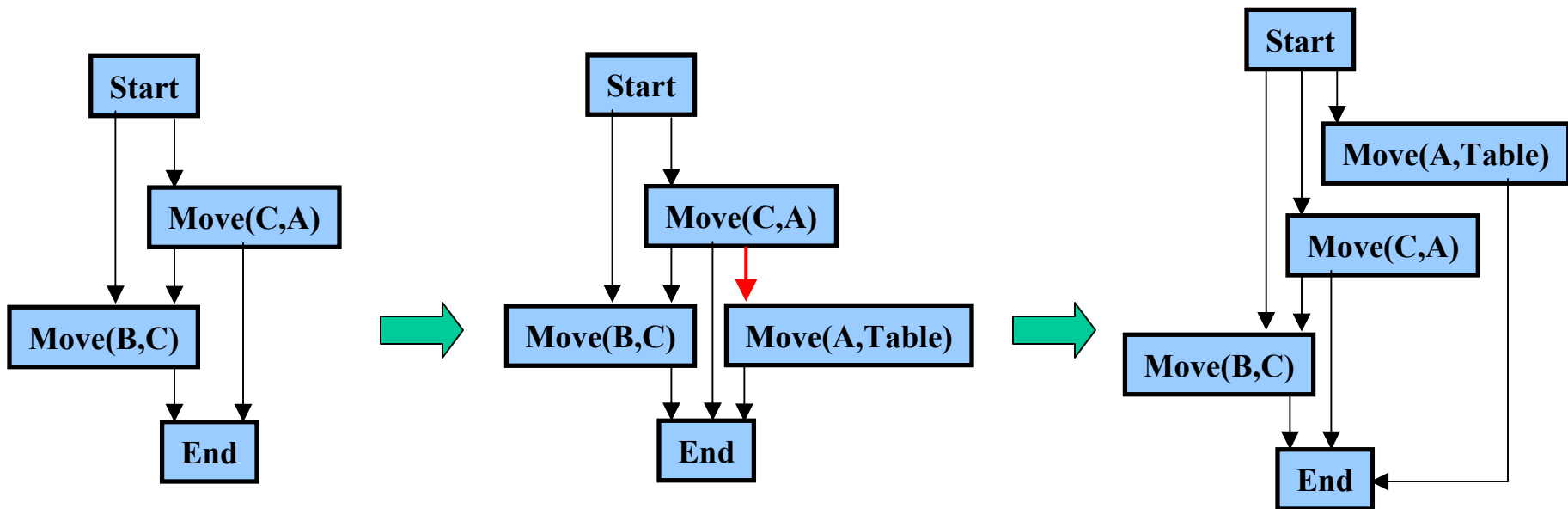
– Backwards search:

- To satisfy End's first condition, use Move(B,C)
 - Move(B,C) is now another subgoal to consider
 - Start satisfies two preconditions of Move(B,C)
- To satisfy End's second condition, use Move(C,A)
 - There is a potential threat between a postcondition of Move(B,C) and a precondition of Move(C,A)
 - Therefore, Move(C,A) must come first (no conflict).



Partial-Order Planning (cont'd)

- To satisfy End's third condition, use Move(A,Table)
 - There is a potential threat between a postcondition of Move(C,A) and a precondition of Move(A,Table)
 - Therefore, Move(C,A) must come before Move(A,Table).
 - This also satisfies the outstanding precondition of Move(B,C), otherwise we'd look at that condition next
- Conditions satisfied, plan achieved, no backtracking!



Variations on POP

- **Sensorless planning**
 - similar to the sensorless searching problem
 - planning is for all contingencies (i.e. no start conditions assumed). Navigate through belief states to satisfy goal condition.
- **Conditional planning**
 - outcome of certain events is not certain
 - plan can split into two directions at some point. Split is assumed to be relatively predictable, so that plans can be made for both outcomes.
- **Execution monitoring**
 - on-the-fly state monitoring and plan modification
- **Continuous planning**
 - no simple start-to-end planner; continuously generates new goals and new plans to achieve them

Time Constraints

- Having a plan settled is a good step, but for many real-world applications it is also useful to know how long a plan will take to execute.
- In the case where all the steps are in series with each other, the times for each stage are simply added. However, for branches in parallel, there are additional constraints
 - total duration is measured along the **critical path** (the parallel path that takes longest to run)
 - certain parallel segments may have to wait on a common resource (think parallel OS processes). This not only affects the total time, but also the order in which the parallel processes occur (to minimize extra overhead)

Time Constraints (cont'd)

- With parallel plans, one is typically slower, giving the other path a **slack time**, indicating the difference between the slow path and the current one
- The **minimum slack algorithm** is a heuristic which chooses to execute the action that has the least slack, whenever a resource constraint exists

Modern Planning Algorithms

- POP (1991)
- Graph Plan (1995)
- SAT Plan (1996)
- Forward search with heuristics (2000)
 - most-constrained variable heuristic