OPPA European Social Fund

Prague & EU: We invest in your future.

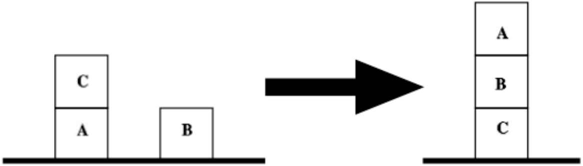
Automated (AI) Planning

Planning as Plan-Space Search

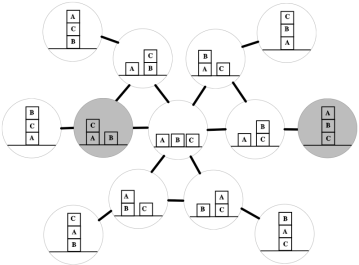
Carmel Domshlak

State Space Search

- So far we have considered planning as search in state space
  - forward - build a plan in the same order that it is executed
  - backward - build a plan in the reverse order of its execution
  - temporal undirected - unordered commitments on executing actions in time



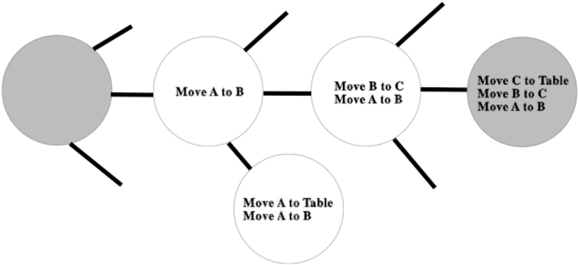
State Space Search



- Potential problem:
  Spending lots of time on trying the same set of actions in different orderings before realizing that there is no solution (with this set)
  - Easier to see in FS/BS, and a bit harder to see in TUS.
- Key observation: When we choose what to do, we also choose when to do

Searching in the Space of Plans

- In 1974, Earl Sacerdoti built a planner, called *NOAH*, that considered planning as search through plan space
  - Search states (nodes) = partially specified plans
  - Transitions (edges) = plan refinement operations
  - Initial state = null plan
  - Goal states = valid plans for the problems



State Space vs. Plan Space

- Search through plan space ... hmm ... *what is plan*?
- Answer I: Totally ordered sequence of either actions or meta-actions
  - But then search through state space is isomorphic to search through plan space!
  - Hmmm . . . the nature of the space being searched is in the eye of the beholder ...
  - So what is the point of introducing "search through plan space"??
- Answer II: Partially ordered sequence of actions

State Space vs. Plan Space

- Search through plan space ... hmm ... *what is plan*?
- Answer I: Totally ordered sequence of either actions or meta-actions
  - But then search through state space is isomorphic to search through plan space!
  - Hmmm . . . the nature of the space being searched is in the eye of the beholder ...
  - So what is the point of introducing "search through plan space"??
- Answer II: Partially ordered sequence of actions

Least Commitment Planning

Think how *you* might solve a planning problem of ...
going for a vacation to Italy

1. Need to purchase plane tickets
2. Need to buy a "Lonely Planet" guide to Italy

BUT there is no need to decide (*yet*) which purchase should be done first

Least Commitment Planning

Think how *you* might solve a planning problem of ...
going for a vacation to Italy

1. Need to purchase plane tickets
2. Need to buy a “Lonely Planet” guide to Italy

BUT there is no need to decide (*yet*) which purchase should be done first

Least Commitment Planning

- Represent plans in a flexible way that enables deferring decisions
- At the planning phase, only the essential ordering decisions are recorded

Partial-Order Plans

- Given a Strips task $\Pi = (P, A, I, G)$ we search through a space of *hypothetical* partial-order plans

- A plan (= search node) is a triplet: $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$ in which
  - $\mathcal{A}$ is a set of actions from A, possibly with (labeled) repetitions
  - $\mathcal{O}$ is a set of ordering constraints over $\mathcal{A}$
  - $\mathcal{L}$ is a set of causal links (a bit later)
- Example: $\mathcal{A} = \{a_1, a_2, a_3\}$, $\mathcal{O} = \{a_1 < a_3, a_2 < a_3\}$

- Observe: Planner (eventually) must do constraint satisfaction to ensure the consistency of $\mathcal{O}$.

Causal Links

A key aspect of least commitment planning is to keep track of past decisions and the *reasons* for those decisions

- If you purchase plane tickets, then make sure bring them to the airport
- If another goal causes you to drop the tickets (e.g., having you hands free to open the taxi door), then you should be sure to pick them up again.

- A good way to reason about (and act for) non-interference between different actions introduced to the plan is to record dependencies between actions *explicitly*
- Causal links $a_p \xrightarrow{q} a_c$ records our decision to use a_p to produce the precondition q of a_c

Causal Links

A key aspect of least commitment planning is to keep track of past decisions and the *reasons* for those decisions

- If you purchase plane tickets, then make sure bring them to the airport
- If another goal causes you to drop the tickets (e.g., having you hands free to open the taxi door), then you should be sure to pick them up again.

- A good way to reason about (and act for) non-interference between different actions introduced to the plan is to record dependencies between actions *explicitly*
- Causal links $a_p \xrightarrow{q} a_c$ records our decision to use a_p to produce the precondition q of a_c

Threats

- Causal links are used to detect when a newly introduced action interferes with past decisions.
- Such an action is called a threat

- Suppose that
  - $a_p \xrightarrow{q} a_c$ is a causal link in $\mathcal{L}$ (of some plan $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$), and
  - a_t is yet another action in $\mathcal{A}$
- We say that a_t threatens $a_p \xrightarrow{q} a_c$ if
  - $\mathcal{O} \cup \{a_p < a_t < a_c\}$ is consistent, and
  - $q \in \mathsf{del}(a_t)$

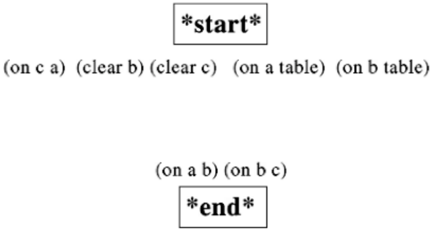
Eliminating Threats

- When a plan contains a threat, then it is *possible* that the plan would not work as anticipated.
  - *Which means what?*
- Solution: identify threats and take evasive countermeasures
  - promotion by $\mathcal{O} \cup = \{a_t > a_c\}$
  - demotion by $\mathcal{O} \cup = \{a_t < a_p\}$
  - ...

Planning Problems as Null Plans

Uniformity is the key for simplicity

- Can use the same structure to represent both the planning problem and complete plans
- Planning problem as a **null plan** $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$ where
  - $\mathcal{A} = \{a_0, a_\infty\}$, $\mathcal{O} = \{a_0 < a_\infty\}$, $\mathcal{L} = \{\}$
  - pre$(a_0) = \{\}$, del$(a_0) = \{\}$, add$(a_0) = I$
  - pre$(a_\infty) = G$, del$(a_0) = \{\}$, add$(a_0) = \{\}$



The POP Algorithm
Schematic description

Regressive algorithm that searches plan-space

- Starts with the null plan
- Makes *non-deterministic* plan refinement choices until
  - all preconditions of all actions in the plan have been supported by causal links, and
  - all threatened causal links have been protected from possible interference

The POP Algorithm
Input and Output

Recursive calls to POP with $POP(\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle, agenda, A)$ where

- $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$ is a plan structure
- $agenda$ is a list of "open goals" that need to be supported by causal links
- A is the action set of our Strips problem

Initial call is with

- null plan $\langle \{a_0, a_\infty\}, \{a_0 < a_\infty\}, \{\} \rangle$, and
- $agenda = \{(g, a_\infty) \mid g \in \mathsf{pre}(a_\infty) \equiv G\}$

If $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$ is outputted by POP, then *any* total ordering of actions $\mathcal{A}$ consistent with $\mathcal{O}$ is a valid plan for our problem.

The POP Algorithm

$POP(\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle, agenda, A)$

- Termination: **if** $agenda = \emptyset$ **then return** $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$
- Goal selection: **choose** $(q, a_{need}) \in agenda$

The POP Algorithm

$POP(\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle, agenda, A)$

- Termination: **if** $agenda = \emptyset$ **then return** $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$
- Goal selection: **choose** $(q, a_{need}) \in agenda$
- Action selection:
  - **choose** action a_{add} (either from $\mathcal{A}$, or from A) such that
    - $q \in$ add(a_{add}), and
    - $\mathcal{O} \cup \{a_{add} < a_{need}\}$ is consistent
  - **if** no such action **then return** FALSE
  - **otherwise**
    - $\mathcal{L} \cup= \{a_{add} \xrightarrow{q} a_{need}\}$ and $\mathcal{O} \cup= \{a_{add} < a_{need}\}$
    - **if** a_{add} is a new action instance **then** $\mathcal{A} \cup= \{a_{add}\}$, and $\mathcal{O} \cup= \{a_0 < a_{add} < a_\infty\}$

The POP Algorithm

$POP(\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle, agenda, A)$

- Termination: **if** $agenda = \emptyset$ **then return** $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$
- Goal selection: **choose** $(q, a_{need}) \in agenda$
- Action selection:
  - **choose** action a_{add} (either from $\mathcal{A}$, or from A) such that
    - $q \in$ add(a_{add}), and
    - $\mathcal{O} \cup \{a_{add} < a_{need}\}$ is consistent
  - **if** no such action **then return** FALSE
  - **otherwise**
    - $\mathcal{L} \cup= \{a_{add} \xrightarrow{q} a_{need}\}$ and $\mathcal{O} \cup= \{a_{add} < a_{need}\}$
    - **if** a_{add} is a new action instance **then** $\mathcal{A} \cup= \{a_{add}\}$, and $\mathcal{O} \cup= \{a_0 < a_{add} < a_\infty\}$
- Update goal set:
  - $agenda \setminus= \{(q, a_{need})\}$
  - **if** a_{add} was a new action instance **then** $agenda \cup= \{(r, a_{add}) \mid r \in$ pre$(a_{add})\}$

The POP Algorithm

$POP(\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle, agenda, A)$

- Termination: **if** $agenda = \emptyset$ **then return** $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$
- Goal selection: **choose** $(q, a_{need}) \in agenda$
- Action selection: **choose** and **process** a_{add} ...
- Update goal set: add preconditions of a_{add} to the agenda

  ...

- Causal link protection: **foreach** causal link $\{a_p \xrightarrow{r} a_c\} \in \mathcal{L}$, and a_t that is threatening it
  - **choose** either $\mathcal{O} \cup = \{a_t > a_c\}$, or $\mathcal{O} \cup = \{a_t < a_p\}$
  - **if** neither constraint is consistent **then return** FALSE

The POP Algorithm

$POP(\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle, agenda, A)$

- Termination: **if** $agenda = \emptyset$ **then return** $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$
- Goal selection: **choose** $(q, a_{need}) \in agenda$
- Action selection: **choose** and **process** a_{add} ...
- Update goal set: add preconditions of a_{add} to the agenda

  ...

- Causal link protection: **foreach** causal link $\{a_p \xrightarrow{r} a_c\} \in \mathcal{L}$, and a_t that is threatening it
  - **choose** either $\mathcal{O} \cup = \{a_t > a_c\}$, or $\mathcal{O} \cup = \{a_t < a_p\}$
  - **if** neither constraint is consistent **then return** FALSE
- Recursive invocation: $POP(\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle, agenda, A)$

Choice Points

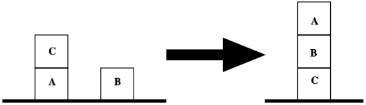
Three choice points

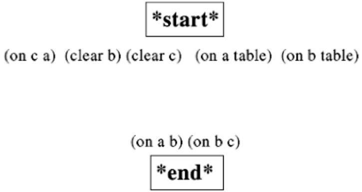
- Goal selection
- Action selection
- Causal link protection

How crucial these choices are?

- Affect soundness?
- Affect completeness?
- Affect efficiency?

Example - Step 1





Initial call to POP with

- Null Plan (see the right figure)
- $agenda = \{(\texttt{onAB}, a_\infty), (\texttt{onBC}, a_\infty)\}$

First choice is *goal selection*

- Affects efficiency, but *not* completeness!

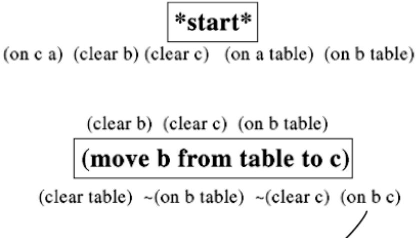
Example - Step 2

Suppose $(\texttt{onBC}, a_\infty)$ is selected (i.e., $a_{need} = a_\infty$)

- Need to choose an action a_{add} that will provide onBC
  - This is a real non-deterministic choice!

Suppose that an oracle suggests making a_{add} be a new instance of the action $move\text{-}B\text{-}from\text{-}Table\text{-}to\text{-}C$

- a causal link $a_{add} \xrightarrow{\texttt{onBC}} a_\infty$ is added to $\mathcal{L}$
- *agenda* is properly updated *(how exactly?)*
- no threats to resolve ... recursive call



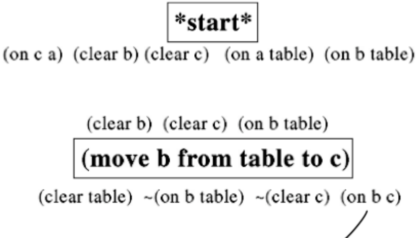
Example - Step 2

Suppose $(\texttt{onBC}, a_\infty)$ is selected (i.e., $a_{need} = a_\infty$)

- Need to choose an action a_{add} that will provide onBC
  - This is a real non-deterministic choice!

Suppose that an oracle suggests making a_{add} be a new instance of the action $move\text{-}B\text{-}from\text{-}Table\text{-}to\text{-}C$

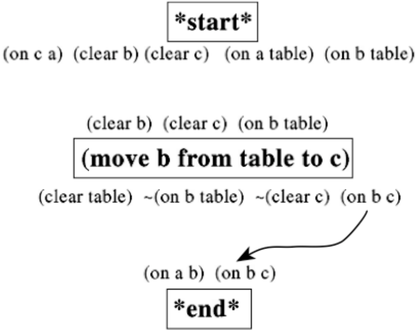
- a causal link $a_{add} \xrightarrow{\texttt{onBC}} a_\infty$ is added to $\mathcal{L}$
- $agenda$ is properly updated *(how exactly?)*
- no threats to resolve . . . recursive call



Example - Step 2

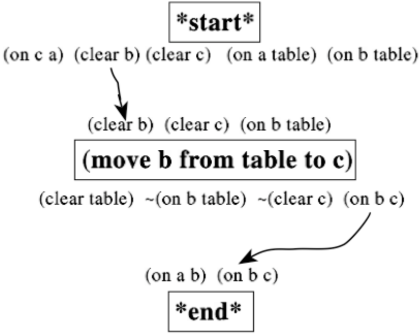
Suppose that an oracle suggests making a_{add} be a new instance of the action $move\text{-}B\text{-}from\text{-}Table\text{-}to\text{-}C$

- a causal link $a_{add} \xrightarrow{\texttt{onBC}} a_\infty$ is added to $\mathcal{L}$
- $agenda$ is properly updated *(how exactly?)*
- no threats to resolve . . . recursive call



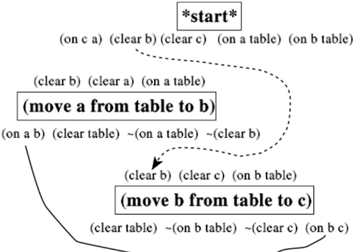
Example - Step 3

- Suppose $(\texttt{clearB}, move\text{-}B\text{-}from\text{-}Table\text{-}to\text{-}C)$ is selected
- Oracle suggests to reuse an existing action instance a_0
  - add a causal link $a_0 \xrightarrow{\texttt{clearB}} move\text{-}B\text{-}from\text{-}Table\text{-}to\text{-}C$
  - $agenda$ is properly updated *(how exactly?)*
  - no threats to resolve . . . recursive call



Example - Step 4a

- Suppose $(\texttt{onAB}, a_\infty)$ is selected
- Oracle suggests making a_{add} be a new instance of the action $move\text{-}A\text{-}from\text{-}Table\text{-}to\text{-}B$, and we do that ...
- ... BUT this time we have a threat!
  - $move\text{-}A\text{-}from\text{-}Table\text{-}to\text{-}B$ and $move\text{-}B\text{-}from\text{-}Table\text{-}to\text{-}C$ have no constraints on their relative ordering
  - $move\text{-}A\text{-}from\text{-}Table\text{-}to\text{-}B$ deletes `clearB` that is required by $move\text{-}B\text{-}from\text{-}Table\text{-}to\text{-}C$

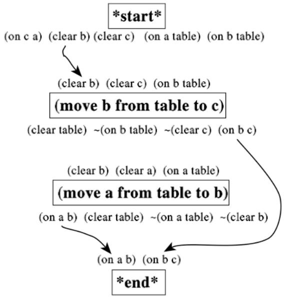


Example - Step 4b

Try to protect the causal link

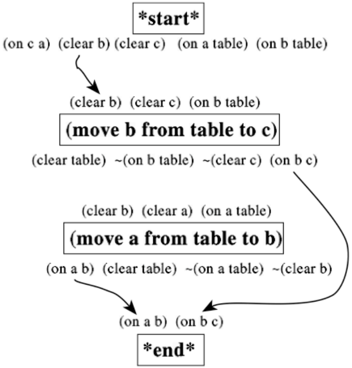
$a_0 \xrightarrow{\texttt{clearB}} move\text{-}B\text{-}from\text{-}Table\text{-}to\text{-}C$

- In general, there are two options — *promotion* and *demotion* — and this is a true non-deterministic choice!
- In our example, demotion is inconsistent *(why?)*, but promotion is OK



Example - Next steps

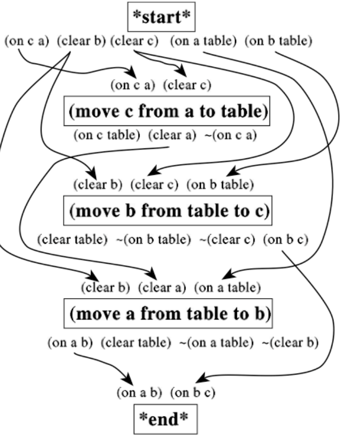
- What is now on the *agenda*? ...in $\mathcal{A}$? ...in $\mathcal{L}$? ...in $\mathcal{O}$?



- Next steps follow the same lines of reasoning

Example - Next steps

- Eventually POP returns



- Blackboard: *Is it a correct partial order plan?*

Advantages

- Natural extension to planning with partially instantiated actions
  - ... add action instance $move\text{-}A\text{-}from\text{-}x?\text{-}to\text{-}B$
  - ... postpone unifying $?x$ with a concrete object until necessary
- Natural extensions to more complex action formalisms
  - ... action durations
  - ... delayed effects
  - ...
- Least commitment may lead to shorter search times
  - Mainly due to smaller branching factor

Disadvantages

- Significantly more complex algorithm
  - ... higher *per-node* cost
- Hard to determine what is true in a state
  - ... harder to devise informed heuristics (for all three types of choices)
  - ... how to prune infinitely long paths??

Framework: Temporal POCL planning

Temporal planning problem

- $\Pi = \langle P, A, I, G \rangle$ where
  - P is a set of atoms,
  - $I \subseteq P$ is the initial state,
  - $G \subseteq P$ is the goal,
  - A is the set of actions,
    each with $pre(a)$, $add(a)$, and $del(a)$, and duration $dur(a)$.
- Two 'dummy' actions: $Start$ produces I, End requires G.
- Two actions a and b interfere when
  - $[pre(a) \cup add(a)] \cap del(b) \neq \emptyset$ or
  - $[pre(b) \cup add(b)] \cap del(a) \neq \emptyset$

Framework: Temporal POCL planning

- Applicability of an action: $pre(a) \subseteq s$
- Action application: $[s - del(a)] \cup add(a)$
- Goal state: $G \subseteq s$
- Solution plan: set ρ of couples $\langle a_i, t_i \rangle, i = 1, \ldots, n$ st:
  - $a_i \in A$ and t_i starting time of the application of a
  - $\forall \langle a_i, t_i \rangle \in \rho$, $pre(a_i)$ true at time t_i
  - $\forall g \in G$, g true at time $\max_{\langle a_i, t_i \rangle \in \rho}[t_i + dur(a_i)]$
  - $\forall \langle a_i, t_i \rangle, \langle a_j, t_j \rangle \in \rho$, if a_i and a_j interfere, then they do not overlap.

Framework: Temporal POCL planning

- Applicability of an action: $pre(a) \subseteq s$
- Action application: $[s - del(a)] \cup add(a)$
- Goal state: $G \subseteq s$
- Solution plan: set ρ of couples $\langle a_i, t_i \rangle, i = 1, \ldots, n$ st:
  - $a_i \in A$ and t_i starting time of the application of a
  - $\forall \langle a_i, t_i \rangle \in \rho$, $pre(a_i)$ true at time t_i
  - $\forall g \in G$, g true at time $\max_{\langle a_i, t_i \rangle \in \rho}[t_i + dur(a_i)]$
  - $\forall \langle a_i, t_i \rangle, \langle a_j, t_j \rangle \in \rho$, if a_i and a_j interfere, then they do not overlap.

Goal

Develop an optimal temporal planner with good performance

Optimal planning is branching and pruning

Branching is used for expanding partial solutions

Pruning is used for discarding them

Optimal state-based planners:

- Branch by performing state progression or regression
- Prune by comparing the estimated cost of the partial plan with a given bound

Optimal SAT and CSP planners:

- Branch by picking a variable and trying each of its values
- Prune by backtracking over inconsistencies due to encoded bounds

Pruning: A key feature of modern planners

- In heuristic search planners achieved by use of lower bounds or admissible heuristics
- In SAT and CSP approaches achieved by adding the goal at a fixed bound and performing constraint propagation
- Both ideas combined in SAT/CSP formulations obtained from planning graph (that contains lower bounds)

POCL: Smart but blind branching scheme

POCL branching main loop:

- Find and repair a "flaw" till not possible (and then backtrack) or done
- Flaws: open conditions, threats, ...

Benefit: easy to extend to temporal planning

Problem: weak pruning mechanism; detects very late that partial plan is not good

Benefit POCL branching: Easy to add time

- Partial ordering $a < a'$ over actions replaced by temporal precedence constraint $T(a) + dur(a) <= T(a')$

- Consistency over resulting Simple Temporal Problem easy to enforce by bounds consistency:

  Iterate over

$$T_{max}(a) := \min[T_{max}(a), T_{max}(a') - dur(a)]$$

$$T_{min}(a') := \max[T_{min}(a'), T_{min}(a) + dur(a)]$$

  until fixed-point or some empty variable domain

- Expressive planners based on this formulation are IxTeT, RAX...

Problem POCL planning: Weak pruning

Backtrack when partial plan or STP inconsistent.

$\rightsquigarrow$ Need to detect "bad partial plans" earlier

Example: TOWER-N problems

- Initial state: N blocks b_i lie on the table
- Goal: $\forall i \in [1, \ldots, N-1]$, $on(b_i, b_{i+1})$

One partial plan:

$$\langle stack(b_{i+1}, b_{i+2}), t\rangle, \langle stack(b_i, b_{i+1}), t+2\rangle$$

Open condition for $\langle stack(b_i, b_{i+1}), t+2\rangle$:

$$\langle holding(b_i), t+2\rangle$$

Problem POCL planning: Weak pruning

Possible repairs:

- $\langle pickup(b_i), t+1 \rangle$

  $\Longrightarrow$ 1 good choice: can lead to a solution

- $\langle pick(b_i, b_j), t+1 \rangle$, for all $j \in [1..N], i \neq j$

  $\Longrightarrow$ *N*-1 bad choices: backtrack (later) because do not lead to optimal solutions

Recent attempts (RePop, VHPOP) for guiding search but no optimality guarantees

Proposed approach

Solves TOWER-N problems optimally and backtrack free

CPT: Temporal POCL with strong pruning

- POCL branching over time: STP + bounds consistency
- Strong pruning: representing and reasoning about all possible actions in the domain, not only those already commited in the plan
- Canonicity restriction: no action executed more than once in the plan (this restriction can be eliminated)

Constraint Programming formulation

1. Variables
2. Domain preprocessing
3. Constraints
4. Branching scheme and heuristic

Variables

For all action $a \in A$ and all $p \in pre(a)$:

- $T(a) :: [0, \infty]$ = starting time of a
- $S(p, a) :: \{a' \in A | p \in add(a')\}$ = support of p for a
- $T(p, a) :: [0, \infty]$ = starting time of support $S(p, a)$
- $InPlan(a) :: [0, 1]$ = presence of a in the plan

Preprocessing: Lower bounds by *some* h_T

- Use some backward heuristic h_T that is admissible for makespan
- Simple example: $h_T^{\max}$

$$h^{\max}(s) = \begin{cases} 0, & s \subseteq I \\ \min_{a \in A, p \in add(a)} 1 + h^{\max}(pre(a)), & |s| = \{p\} \\ \max_{p \in s} h^{\max}(\{p\}), & |s| > 1 \end{cases}$$

$$h_T^{\max}(s) = \begin{cases} 0, & s \subseteq I \\ \min_{a \in A, p \in add(a)} dur(a) + h_T^{\max}(pre(a)), & |s| = \{p\} \\ \max_{p \in s} h_T^{\max}(\{p\}), & |s| > 1 \end{cases}$$

Preprocessing (I)

- Initial lower bounds: $T_{min}(a) = h_T(a)$
- Structural mutexes: pairs of atoms p, q for which $h_T(\{p, q\}) = \infty$
- e-deleters: extended deletes computed from structural mutexes
  action a e-deletes p if
  - a deletes p, or
  - $q \in add(a) \land h_T(\{p, q\}) = \infty$, or
  - $q \in pre(a) \land h_T(\{p, q\}) = \infty \land p \notin add(a)$

Preprocessing (II)

Distances:

- $dist(a, a') = h_T(a')$ with $I = P \setminus edel(a)$
- $dist(Start, a) = h_T(a)$
- $dist(a, End)$:
  shortest-path algorithm on a 'relevance graph'

nodes	actions A
edges	$\{a \rightarrow a' \mid add(a') \cap pre(a) \neq \emptyset\}$
edge cost	of $a \rightarrow a'$ is $\delta(a', a) = dur(a') + dist(a', a)$
source node	End

$$dist(a, End) := spath(End, a) - dur(a)$$

Constraints

- Bounds:

$$T(Start) + dist(Start, a) \leq T(a)$$

$$T(a) + dist(a, End) \leq T(End)$$

- Preconditions:
  supporter a' of precondition p of a must precede a:

$$T(a) \;\geq \min_{a' \in D[S(p,a)]} [T(a') + \delta(a', a)]$$

$$T(a') + \delta(a', a) > T(a) \rightarrow S(p, a) \neq a'$$

Constraints

- Causal Link Constraints: for all $a \in A$, $p \in pre(a)$ and a' that e-deletes p, a' precedes $S(p, a)$ or follows a:

$$T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leq T(p, a)$$

$$\vee \quad T(a) + \delta(a, a') \leq T(a')$$

- Mutex Constraints: for effect-interfering a and a'

$$T(a) + \delta(a, a') \leq T(a') \;\vee\; T(a') + \delta(a', a) \leq T(a)$$

Constraints

- Support Constraints: $T(p,a)$ and $S(p,a)$ related by

$$S(p,a) = a' \rightarrow T(p,a) = T(a')$$

$$\min_{a' \in D[S(p,a)]} T(a') \ \leq \ T(p,a) \ \leq \ \max_{a' \in D[S(p,a)]} T(a')$$

$$T(p,a) \neq T(a') \rightarrow S(p,a) \neq a'$$

Branching

- A Support Threat $\langle a', S(p,a)\rangle$ generates the split

$$[T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leq T(p,a);$$

$$T(a) + \delta(a, a') \leq T(a')]$$

- An Open Condition $S(p,a)$ generates the split

$$[S(p,a) = a'; S(p,a) \neq a']$$

- A Mutex Threat $\langle a, a'\rangle$ generates the split

$$[T(a) + \delta(a, a') \leq T(a'); T(a') + \delta(a', a) \leq T(a)]$$

Heuristics

- Support Threats $\langle a', S(p,a)\rangle$ with minimum slack $\max[slack(a' \prec S(p,a)), slack(a \prec a')]$ selected first, where

$$slack(a \prec a') = T_{max}(a') - (T_{min}(a) + \delta(a,a'))$$
$$slack(a' \prec S(p,a)) = T_{max}(p,a) - (T_{min}(a') + \min_{a' \in D[S(p,a)]} \delta(a',a))$$

- Open conditions $S(p,a)$ selected latest first; i.e. maximizing the expression $\min_{a' \in D[S(p,a)]} T_{min}(a')$, splitting on the 'arg min' action a'.
- Mutex Threats $\langle a, a'\rangle$ selected as they are encountered

Implementation

Constraint programming tools offer:

- Predefined global constraints,
- Efficient procedures for maintaining consistency,
- Extensibility for designing new constraints, new heuristics, and controlling the search,
- Built-in search algorithms such as branch-and-bound.

Additional scheduling technique: Mutex sets

Based on scheduling technique edge-finding:

- A mutex set is a set M of actions *in the plan*, such that any two actions in M are interfering.
- The time window associated with the set of actions M, $\max_{a \in M}(T_{max}(a) + dur(a)) - \min_{a \in M} T_{min}(a)$, must provide enough 'room' for scheduling all actions in $a \in M$ in sequence.
- Lower bound $\Delta(M)$ for the time needed for scheduling all actions in M is given by

$$\sum_{a \in M} [dur(a) + \min_{a' \in M | a' \neq a} dist(a, a')] - \max_{\{a,a'\} \subseteq M} dist(a, a')$$

TOWER-N domain

Problem	CPU time (sec.)				Makespan
	CPT	BBOX	IPP	TP4	
tower-8	0.33	2.95	0.05	17.68	14
tower-9	0.64	7.28	0.11	887.7	16
tower-10	1.01	13.6	0.38	-	18
tower-11	1.69	28.2	2.26	-	20
tower-12	3.61	-	15.35	-	22
tower-13	5.83	-	123.78	-	24
tower-14	9.70	-	-	-	26
tower-15	13.65	-	-	-	28

Temporal domains

Temporal problems	CPU time (sec.) (# states) CPT	TP4	Makespan
zeno1	0.06 (2)	0.05 (4)	173
zeno2	0.95 (892)	1.23 (17124)	592
zeno3	0.50 (4)	0.05 (618)	280
zeno4	4.59 (2233)	-	522
zeno5	3.83 (124)	34.78 (595988)	400
zeno6	1.78 (54)	6.03 (116715)	323
zeno7	77.58 (45187)	-	665
zeno8	265.93 (78044)	-	522
zeno9	1522.24 (432210)	-	522
zeno10	82.62 (12692)	-	453
zeno11	116.15 (874)	-	423

Temporal domains

Temporal problems	CPU time (sec.) (# states) CPT	TP4	Makespan
driver1	0.06 (6)	0.05 (49)	91
driver2	734.98 (724327)	458.19 (17444608)	92
driver3	0.12 (11)	0.05 (621)	40
driver4	91.32 (54350)	-	52
driver5	0.40 (152)	-	51
driver6	111.10 (59702)	-	52
driver7	0.59 (103)	20.79 (323963)	40
driver8	-	-	-
driver9	493.91 (137716)	-	92
driver10	8.75 (1517)	-	38

Temporal domains

Temporal problems	CPU time (sec.) (# states) CPT	TP4	Makespan
satellite1	0.05 (5)	0.05 (80)	46
satellite2	0.95 (1435)	8.45 (712294)	70
satellite3	0.20 (26)	0.05 (21143)	34
satellite4	4.36 (5257)	-	58
satellite5	2.32 (1191)	-	36
satellite6	0.82 (47)	-	46
satellite7	2.36 (325)	-	34
satellite8	3324.92 (827408)	-	46
satellite9	8.84 (516)	-	34
satellite10	2160.24 (261474)	-	43

Temporal domains

Problems	CPU time (sec.)		Makespan	
	LPGP	CPT	LPGP	CPT
zeno4	65.32	4.59	740	522
zeno5	43.83	3.83	583	400
zeno6	57.61	1.78	350	323
driver1	0.33	0.06	91	91
rover1	0.30	0.12	55	53
rover2	0.24	0.07	44	43
rover3	0.44	0.11	58	53
rover4	0.40	0.09	47	45
satellite1	0.17	0.05	46	41
satellite2	24.15	0.95	70	65
satellite3	62.22	0.20	34	29

Parallel domains

Problems	CPU time (sec.)				Makespan
	CPT	BBOX	IPP	TP4	
bw.12step	0.21	0.26	0.03	0.08	12
bw.large.a	0.44	1.13	0.07	0.08	12
bw.large.b	1.75	17.94	2.33	-	18
bw.large.c	231.22	-	-	-	28
rocket.a	0.28	0.38	7.97	44.20	7
rocket.b	0.24	0.45	11.95	31.83	7
log.a	0.70	0.47	781.13	-	11
log.b	0.90	0.91	2099.89	-	13
log.c	1.43	1.46	-	-	13
log.d	29.03	3.73	-	-	14

Parallel domains

Problems	CPU time (sec.)				Makespan
	CPT	BBOX	IPP	TP4	
zeno7	0.84	0.67	0.05	1.76	6
zeno8	5.39	1.59	0.22	166.22	5
zeno9	6.41	2.54	0.68	-	6
zeno10	6.84	4.01	221.32	-	6
zeno11	14.90	5.60	31.06	-	6
zeno12	16.39	11.10	-	-	6
zeno13	45.97	11.42	-	-	7
driver7	0.24	0.24	0.15	22.98	6
driver8	0.30	0.40	3.53	33.59	7
driver9	1.46	1.55	11.26	2979.66	10
driver10	1.02	1.00	17.06	1823.16	7
driver11	4.33	2.67	2.26	1259.06	9

Parallel domains

Problems	CPU time (sec.)				Makespan
	CPT	BBOX	IPP	TP4	
satellite3	0.12	0.26	0.03	0.08	6
satellite4	0.40	1.39	7.28	755.08	10
satellite5	0.99	1.50	145.67	-	7
satellite6	0.56	1.34	90.46	-	8
satellite7	1.55	1.80	1039.23	-	6
satellite8	101.18	235.13	-	-	8
satellite9	8.52	4.68	-	-	6
satellite10	185.90	42.35	-	-	8
satellite11	22.51	-	-	-	8

Non-canonical planning with CPT

The current version of CPT finds non-canonical plans.

Key ideas:

- Distinguish action types from action tokens
- Tokens are generated dynamically from action types

Implementation:

- Emulates domain that contains an infinite supply of tokens
- Variables associated with such tokens are identical until a token becomes part of the plan

Summary

- Optimal temporal planner with performance that approaches best parallel planners over domains with uniform durations
- Combines POCL temporal branching scheme with strong pruning mechanisms based on the use of a variety of constraints and existing lower bounds

Optimal, Suboptimal and Easy planning

- Optimal planning: minimizes plan makespan.

  Examples: GRAPHPLAN, IPP, SATPLAN, GP-CSP, TP4, CPT...

- Suboptimal planning: no guarantee on plan quality, tries to minimize the number of actions in the plan.

  Examples: HSP, FF, LPG, SAPA...

- Easy planning: same as suboptimal planning, with the objective of privilegiate inferences over search.

  Example: eCPT.

Goal

Temporal planner for easy planning that solves as much problems as possible without search.

Without search means:

- Avoid backtracks,
- Privilegiate inferences over search,
- Add only polynomial operations,
- Analyse the results from the point of view of general behavior (backtracks, ...) instead of running time.

Difficulty of easy planning

- Very few planners perform inferences, some examples are SATPLAN, GP-CSP and CPT.
- To render them "easy": increase the lower bound on the makespan (the horizon).

Two problems appear:

1. The size of the encodings based on one variable per time unit increases too much,
2. Constraints that require the validity of the goals at the horizon loose their pruning power.

Extensions of CPT for “easy planning”

A combination of simple ideas, obtained from the observation and analysis of the behavior (backtracks, ...) in various problems.

- Impossible supports
- Unique supports
- Distance boosting
- Qualitatives precedences
- Actions landmarks
- Branching and heuristics

Impossible supports

Supports elimination by preprocessing.

Example:

- $unstack(A, B)$ has $handempty$ as precondition
- $putdown(A)$ adds $handempty$ so

$$putdown(A) \in D[S(handempty, unstack(A, B))]$$

$\Longrightarrow$ but: $putdown(A)$ e-deletes $on(A, B)$, precondition of $unstack(A, B)$

$\Longrightarrow$ furthermore: $on(A, B)$ cannot be re-established without deleting $handempty$

Impossible supports

For each variable $S(p, a)$ and each value $a' \in D[S(p, a)]$:

- let $I' = P \setminus edel(a')$
- let $A' = A \setminus \{a \in A | p \in add(a) \cup del(a)\}$
- reachability analysis with I' and A'

$\Longrightarrow$ if a precondition of a is not reachable: $S(p, a) \neq a'$

Unique supports

Pruning rule used during constraint propagation.

Example:

- $unstack(A, B)$ and $pickup(C)$ have $handempty$ as precondition and delete,
- they cannot be applied in parallel,
- after the application of one of them, $handempty$ is deleted.

$\Longrightarrow$ the action that supports $handempty$ for the first cannot support $handempty$ for the second.

Unique supports

- An action a consumes an atom p when

$$p \in pre(a) \cap del(a)$$

- For two actions a and a' that consumes the same atom p, the following constraint is added:

$$S(p, a) \neq S(p, a')$$

Distance boosting

Increases distances and prunes supports by preprocessing.

Example:

- The distance between $putdown(A)$ and $pickup(A)$ is equal to 0.

  $\Longrightarrow$ However, applying $putdown(A)$ then $pickup(A)$ is useful only if an action inserted between them uses an effect of $putdown(A)$, for example if A is on a block B that we want to move.
- Similarly, the distance between $pickup(A)$ and $putdown(A)$ is equal to 0.

  $\Longrightarrow$ But: no action can be inserted between them that uses an effect of $pickup(A)$.

Distance boosting

An action a cancels an action a' when

- All atoms added by a' are e-deleted by a,
- All atoms added by a are preconditions of a'.

For an action a that cancels an action $a' \in D[S(p,a)]$:

- If all actions that use an add effect of a' e-delete p: $S(p,a) \neq a'$.
- Else: $dist(a',a)$ becomes $\min_b[dist(a',b) + dist(b,a)]$ with $b \neq a$ and $b \neq a'$, such that
  - either b uses an add effect of a' but does not e-delete p,
  - or b adds p.

Qualitative precedences

CPT reasons with temporal precedences of the form $T(a) + \delta(a, a') \leq T(a')$ instead of qualitative precedences.

$\Longrightarrow$ Problem: they does not capture transitivity.

For exemple: from $a < b$ and $b < c$, CPT does not infer $a < c$.

- the initial domain of variables a, b, and c is $[1, \ldots, 100]$,
- by bounds consistency:

$$a :: [1, \ldots, 98], b :: [2, \ldots, 99], c :: [3, \ldots, 100]$$

$\Longrightarrow$ does not make $a < c$ true for every combination of values

Qualitative precedences

When a temporal precedence is made true: a qualitative precedence is recorded. For $a \prec a'$, the transitive closure is computed:

- if $InPlan(a) = 1$: $\forall a''$ st $a'' \prec a$, $a'' \prec a'$ is inferred
- if $InPlan(a') = 1$: $\forall a''$ st $a' \prec a''$, $a \prec a''$ is inferred

Inference rules using these qualitative precedences:

- for an action $a' \in D[S(p, a)]$:

  if $InPlan(a') = 1$ and $a \prec a'$ then $S(p, a) \neq a'$

- for an action $a' \in D[S(p, a)]$ and an action b that e-deletes p:

  if $InPlan(b) = 1$, $a' \prec b$ and $b \prec a$, then $S(p, a) \neq a'$

Landmark actions

By preprocessing: we can find some actions that must belong to any solution plan.

For example:

- a block A must be moved,
- A is under B, itself under C.

$\Longrightarrow unstack(C, B)$ and $unstack(B, A)$ must be used in any solution plan, and $unstack(C, B) \prec unstack(B, A)$.

- An action a is a landmark if a goal of the problem is not reachable when a is exluded from the domain.
- An action landmark a precedes an action landmark b, if b is not reachable when the action a is excluded.

Branching and heuristics

Support threats $\langle a', S(p,a)\rangle$:

1. to minimize $T_{min}(a)$
2. to minimize $T_{max}(p,a)$
3. to minimize $\max[slack(a' \prec S(p,a)), slack(a \prec a')]$
   where:
   $slack(a,a') = T_{max}(a') - [T_{min}(a) + \delta(a,a')]$
   $slack(a', S(p,a)) =$
   $T_{max}(p,a) - [T_{min}(a') + \min_{a' \in D[S(p,a)]} \delta(a',a)]$

Open conditions $S(p,a)$:

1. to minimize $T_{max}(p,a)$
2. to minimize $slack(a',a) = T_{max}(a) - (T_{min}(a') + \delta(a',a))$ where a' produces p for a ($a' \in D[S(p,a)]$), minimizing $T_{min}(a')$.

Results on various domains

		eCPT			FF	
	#pbs	solved	no bkt. (max bkt.)	max nds	solved	max nds
blocks	50	50	50 (0)	275	42	146624
depots	20	18	16 (4)	285	19	166141
driver	20	17	16 (5)	176	15	4657
ferry	50	50	50 (0)	1176	50	201
gripper	50	50	50 (0)	201	50	200
logistics	50	50	50 (0)	273	50	2088
miconic	50	50	50 (0)	131	50	76
rovers	20	20	20 (0)	207	20	3072
satellite	20	20	20 (0)	249	20	5889
zeno	20	14	14 (0)	70	20	933

Discussion

- Unexpected results: a few simple inference rules are sufficient to avoid backtracks in many benchmarks.
- Interest of the CP+POCL formulation: it has permitted the fine-grained analysis of backtracks and finding new rules.
- Inferences have a cost: actually, methods that privilegiate search are more efficient.
- Robustness improvement: in the domains studied, we almost sure get a solution in reasonable time.

OPPA European Social Fund
Prague & EU: We invest in your future.

OPPA
PRAHA PRAGUE PRAGA PRAG
EVROPSKÁ UNIE