Optimal Planning for Delete-free Tasks with Incremental LM-cut

Florian Pommerening and Malte Helmert

Universität Basel
Departement Informatik

27. 06. 2012

Content

Delete-free Planning

- Binary cost delete-free STRIPS task $\Pi = \langle V, I, G, O \rangle$
    - V set of variables
    - $I, G \subseteq V$ initial/goal state
    - O set of operators $o = \langle \text{pre}(o) \rightarrow \text{add}(o) \rangle_{\text{cost}(o)}$
    - $\text{cost}(o) \in \{0, 1\}$
- Optimal planning
    - Search for cheapest operator sequence $o_1, \ldots o_n$
    - $G \subseteq s[o_1] \cdots [o_n]$
    - NP-equivalent instead of PSPACE-equivalent
- Why?
    - Cost of optimal plan: delete-relaxation heuristic h^+
        - h^+ is well-informed
        - Other heuristics are based on h^+
    - Interesting delete-free domains

$h^{\text{LM-cut}}$

- Based on *disjunctive action landmarks* (LMs)
  - Set of operators $I = \{o_1, \ldots, o_n\}$
  - Every plan contains at least one o_i
  - Cost of a landmark: $\min_{o_i \in I}\{\text{cost}(o_i)\}$

1. Calculate $h^{\max}$
   - Only achieve most expensive subgoal/precondition
   - $h^{\max}(s) = \infty$ task unsolvable
   - $h^{\max}(s) = 0$ stop searching for LMs
2. Use $h^{\max}$ values to discover new LM
3. Reduce operator costs by landmark's cost for operators in LM
   - Sum of landmark costs is admissible heuristic
4. Repeat

Search Strategies

Branch-and-Bound (BnB) Search

- Memory friendly depth-first search
- Recursively search for solution in cost interval
  - Decrease upper bound for every discovered solution
  - Continue search for cheaper solution
  - Prune nodes with lower bound outside of interval

Iterative-deepening A* (IDA*) Search

- Search for solution with increasing cost $h^{\text{LM-cut}}(I), \ldots, h^{+}(I)$
  - IDA* layer i: BnB search with closed interval $[i, i]$

Theorem

BnB and IDA* are *complete* and *optimal* if used with a finite search space and an admissible heuristic.

Content

Search Space

Theorem

Applying an operator cannot make an applicable operator inapplicable in delete-free tasks.

Theorem

No operator has to occur twice in an optimal relaxed solution.

- Order can mostly be ignored
  - Search in serializable subsets of O
- Branch over applicable operator
  - Apply it *now or never*
- Finite branching factor (2) and search tree depth ($|O|$)

Incremental Computation

- Successor generated by applying/removing operator
- Binary cost tasks
  - Each operator o has *containing LM* L_o
  - $L_o = \{o\}$ **or** $|L_o| > 1$ **or** L_o undefined
- Apply operator o
  - L_o discharged
  - All other LMs are LMs in successor
- Remove operator o
  - o no longer possible choice
  - Remove o from L_o
  - $L_o \setminus \{o\}$ is LM in successor
  - Task unsolvable if $L_o = \{o\}$
  - All other LMs are LMs in successor

Re-calculation of $h^{\text{LM-cut}}$

- Removing a LM
  - Return landmark's costs to *remaining cost*
  - Binary cost tasks: Set operator cost back to 1
- Can change $h^{\max}$ value

Theorem

The LM-cut algorithm discovers a new landmark if the $h^{\max}$ cost of the successor increases.

- Only possible if
  - $L_o = \{o, o_1, \ldots, o_n\}$
  - 0-cost operator forbidden with L_o undefined

Variable Ordering

- *Minimum remaining values* heuristic
  - CSP technique
  - Choosing variables to branch over
- One operator from each LM is needed
  - Smaller LM $\Rightarrow$ fewer choices
  - Smallest LM $\sim$ variable with minimum remaining values
- l_{min}: size of smallest LM containing applicable operators
- Collect applicable operators in LMs of size l_{min}
- Randomly select one for branching

Automatic Application of Operators

- Automatically apply operators with $L_o = \{o\}$
  - Branching strategy already contains effect
  - Useful with different heuristic
- Automatically apply 0-cost operators
  - Very useful in domains with such operators
  - No 0-cost operators in tested domains

Content

Methodology

- Evaluation
  - 876 tasks in 22 domains
  - Time limit: 300 s
  - Memory limit: 2 GB (only reached for huge tasks)
- Coverage scores
  - Solve probability for randomly selected domain and task
  - Averages of 5 runs with different seeds

Basic Results

- FastDownward with A* and $h^{\text{LM-cut}}$
- Incremental LM-cut with BnB/IDA*

Resuts

	Coverage (%)
FastDownward	49.249
BnB	59.032
IDA*	60.120

- Improvement over Fast Downward
- IDA* better than BnB
  - But still room for improvement for BnB

Plan Improvement

- Better upper bound $\Rightarrow$ more pruned nodes
- Initial upper bound
  - Use cost of relaxed solution (here: with h^{lst})
  - No search if $h^{\text{lst}}(I) = h^{\text{LM-cut}}(I)$
- Improve intermediate solutions
  - Local Steiner tree improvement (based on h^{lst})
  - Continue search with improved solution and new bound

Results

	Coverage (%)
BnB	59.032
IDA*	60.120
BnB (initial upper bound)	59.981
BnB (improved all solutions)	60.519

Content

Future Work

- Optimization for binary cost tasks
  - Performance of implementation
  - Different operator orders
  - Smaller search space (e.g. task decomposition)
- Generalization to arbitrary costs
  - Branching decisions no longer mutually exclusive
  - Different data structures needed
- Generalization to general planning
  - Classical search space
  - Depth of search space not limited by $|O|$
  - Use $A^*/IDA^*/\ldots$ instead of branch-and-bound search

Main Contributions

- New h^+ values
  - 576 of 876 tasks solved
  - Evaluation of other heuristics (h^{lst}, $h^{\text{LM-cut}}$, $h^{\max}$, $h^{\text{FF/add}}$, ...)
- New ways to calculate h^+
  - BnB/IDA* search with custom search space
  - Incremental version of $h^{\text{LM-cut}}$
  - Exceeds performance of Fast Downward ($A^*/h^{\text{LM-cut}}$)
  - BnB and IDA* incomparable
    - BnB as any-time search

Thank you for your attention!
Any questions?

Planning

- Development of domain independent problem solvers
- Common formalism needed
- STRIPS planning task $\Pi = \langle V, I, G, O \rangle$

Formal definition

- V set of variables
- $I \subseteq V$ initial state
- $G \subseteq V$ goals
- O set of operators with
  - $\mathrm{pre}(o) \subseteq V$ Preconditions
  - $\mathrm{add}(o) \subseteq V$ Add effects
  - $\mathrm{del}(o) \subseteq V$ Delete effects
  - $\mathrm{cost}(o) \in \mathbb{R}_0^+$ Cost

Example (LOGISTICS)

Planning

- Development of domain independent problem solvers
- Common formalism needed
- STRIPS planning task $\Pi = \langle V, I, G, O \rangle$

Formal definition

- V set of variables
- $I \subseteq V$ initial state
- $G \subseteq V$ goals
- O set of operators with
  - $\mathrm{pre}(o) \subseteq V$ Preconditions
  - $\mathrm{add}(o) \subseteq V$ Add effects
  - $\mathrm{del}(o) \subseteq V$ Delete effects
  - $\mathrm{cost}(o) \in \mathbb{R}_0^+$ Cost

Example (LOGISTICS)

- at(package, location)
- at(vehicle, location)
- in(package, vehicle)

Planning

- Development of domain independent problem solvers
- Common formalism needed
- STRIPS planning task $\Pi = \langle V, I, G, O \rangle$

Formal definition

- V set of variables
- $I \subseteq V$ initial state
- $G \subseteq V$ goals
- O set of operators with
  - $\text{pre}(o) \subseteq V$ Preconditions
  - $\text{add}(o) \subseteq V$ Add effects
  - $\text{del}(o) \subseteq V$ Delete effects
  - $\text{cost}(o) \in \mathbb{R}_0^+$ Cost

Example (LOGISTICS)

- {at(p-1, loc-B-1),
  at(p-2, loc-A-2),
  at(truck-1, loc-A-1),
  at(truck-2, loc-B-2)}

Planning

- Development of domain independent problem solvers
- Common formalism needed
- STRIPS planning task $\Pi = \langle V, I, G, O \rangle$

Formal definition

- V set of variables
- $I \subseteq V$ initial state
- $G \subseteq V$ goals
- O set of operators with
  - $\text{pre}(o) \subseteq V$ Preconditions
  - $\text{add}(o) \subseteq V$ Add effects
  - $\text{del}(o) \subseteq V$ Delete effects
  - $\text{cost}(o) \in \mathbb{R}_0^+$ Cost

Example (LOGISTICS)

- {at(p-1, loc-B-1), at(p-2, loc-A-3)}

Planning

- Development of domain independent problem solvers
- Common formalism needed
- STRIPS planning task $\Pi = \langle V, I, G, O \rangle$

Formal definition

- V set of variables
- $I \subseteq V$ initial state
- $G \subseteq V$ goals
- O set of operators with
  - $\text{pre}(o) \subseteq V$ Preconditions
  - $\text{add}(o) \subseteq V$ Add effects
  - $\text{del}(o) \subseteq V$ Delete effects
  - $\text{cost}(o) \in \mathbb{R}_0^+$ Cost

Example (LOGISTICS)

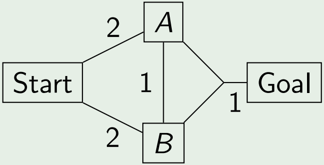
- $o =$ load-truck(?t, ?p, ?l)
- $\text{pre}(o) = \{$at(?t, ?l), at(?p, ?l)$\}$
- $\text{add}(o) = \{$in(?p, ?t)$\}$
- $\text{del}(o) = \{$at(?p, ?l)$\}$
- $\text{cost}(o) = 1$

$h^{\mathrm{FF/add}}$

- Cheapest way to reach a variable: *achiever*
  - Achieve *all* preconditions/subgoals (h^{add})
- Recursively collect necessary achievers in set

Path finding example

- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal



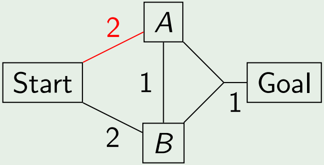
- Overestimation due to greedy search
- In general not admissible ($h^{+} \leq h^{\mathrm{FF/add}}$)

$h^{\text{FF/add}}$

- Cheapest way to reach a variable: *achiever*
  - Achieve *all* preconditions/subgoals (h^{add})
- Recursively collect necessary achievers in set

Path finding example

- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal



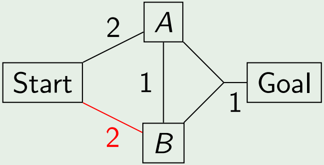
- Overestimation due to greedy search
- In general not admissible ($h^+ \leq h^{\text{FF/add}}$)

$h^{\mathrm{FF/add}}$

- Cheapest way to reach a variable: *achiever*
  - Achieve *all* preconditions/subgoals (h^{add})
- Recursively collect necessary achievers in set

Path finding example

- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal



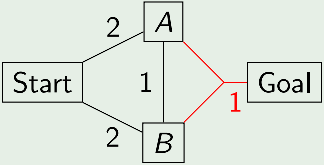
- Overestimation due to greedy search
- In general not admissible ($h^{+} \leq h^{\mathrm{FF/add}}$)

$h^{\mathrm{FF/add}}$

- Cheapest way to reach a variable: *achiever*
  - Achieve *all* preconditions/subgoals (h^{add})
- Recursively collect necessary achievers in set

Path finding example

- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal



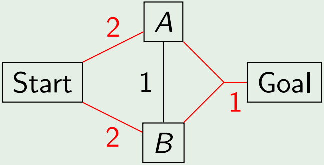
- Overestimation due to greedy search
- In general not admissible ($h^+ \leq h^{\mathrm{FF/add}}$)

$h^{\mathrm{FF/add}}$

- Cheapest way to reach a variable: *achiever*
  - Achieve *all* preconditions/subgoals (h^{add})
- Recursively collect necessary achievers in set

Path finding example

- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal

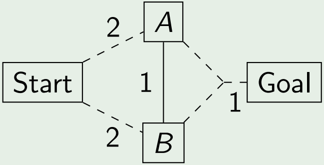


- Overestimation due to greedy search
- In general not admissible ($h^+ \leq h^{\mathrm{FF/add}}$)

Local Steiner Tree Plan Improvement Procedure

Path finding example

- Pick a variable: B
- Partition plan
  - Part dependent on B
  - Part only used to add B
  - Rest
- Find cheaper alternative to reach B

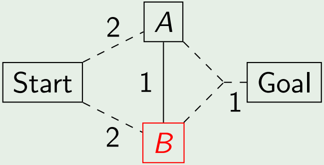


- h^{lst}: Optimization of $h^{\mathrm{FF/add}}$
- Achiever mapping for arbitrary plan π
  - Achiever of v: first operator adding v in π
  - Extract solution with $h^{\mathrm{FF/add}}$
  - Remove unnecessary achiever settings

Local Steiner Tree Plan Improvement Procedure

Path finding example

- Pick a variable: B
- Partition plan
  - Part dependent on B
  - Part only used to add B
  - Rest
- Find cheaper alternative to reach B

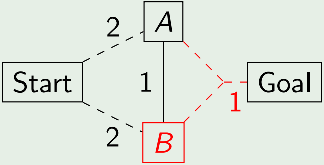


- h^{lst}: Optimization of $h^{\text{FF/add}}$
- Achiever mapping for arbitrary plan π
  - Achiever of v: first operator adding v in π
  - Extract solution with $h^{\text{FF/add}}$
  - Remove unnecessary achiever settings

Local Steiner Tree Plan Improvement Procedure

Path finding example

- Pick a variable: B
- Partition plan
  - Part dependent on B
  - Part only used to add B
  - Rest
- Find cheaper alternative to reach B

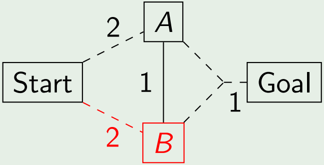


- h^{lst}: Optimization of $h^{\text{FF/add}}$
- Achiever mapping for arbitrary plan π
  - Achiever of v: first operator adding v in π
  - Extract solution with $h^{\text{FF/add}}$
  - Remove unnecessary achiever settings

Local Steiner Tree Plan Improvement Procedure

Path finding example

- Pick a variable: B
- Partition plan
  - Part dependent on B
  - Part only used to add B
  - Rest
- Find cheaper alternative to reach B

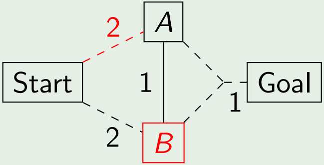


- h^{lst}: Optimization of $h^{\mathrm{FF/add}}$
- Achiever mapping for arbitrary plan π
  - Achiever of v: first operator adding v in π
  - Extract solution with $h^{\mathrm{FF/add}}$
  - Remove unnecessary achiever settings

Local Steiner Tree Plan Improvement Procedure

Path finding example

- Pick a variable: B
- Partition plan
  - Part dependent on B
  - Part only used to add B
  - Rest
- Find cheaper alternative to reach B

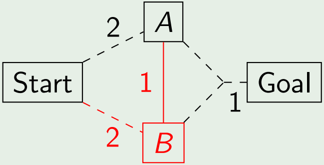


- h^{lst}: Optimization of $h^{\mathrm{FF/add}}$
- Achiever mapping for arbitrary plan π
  - Achiever of v: first operator adding v in π
  - Extract solution with $h^{\mathrm{FF/add}}$
  - Remove unnecessary achiever settings

Local Steiner Tree Plan Improvement Procedure

Path finding example

- Pick a variable: B
- Partition plan
  - Part dependent on B
  - Part only used to add B
  - Rest
- Find cheaper alternative to reach B

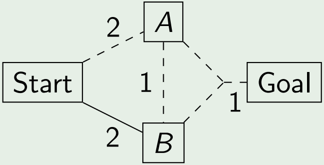


- h^{lst}: Optimization of $h^{\text{FF/add}}$
- Achiever mapping for arbitrary plan π
  - Achiever of v: first operator adding v in π
  - Extract solution with $h^{\text{FF/add}}$
  - Remove unnecessary achiever settings

Local Steiner Tree Plan Improvement Procedure

Path finding example

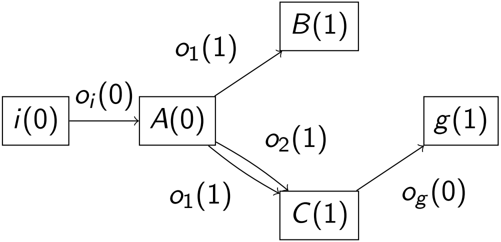
- Pick a variable: B
- Partition plan
  - Part dependent on B
  - Part only used to add B
  - Rest
- Find cheaper alternative to reach B



- h^{lst}: Optimization of $h^{\mathrm{FF/add}}$
- Achiever mapping for arbitrary plan π
  - Achiever of v: first operator adding v in π
  - Extract solution with $h^{\mathrm{FF/add}}$
  - Remove unnecessary achiever settings

Justification Graph

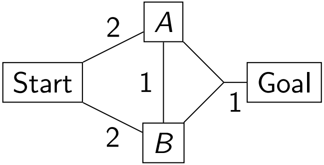
- Precondition choice function (pcf)
  - Maps operators to most expensive precondition
  - Not unique
- LMs discovered with *justification graph*
  - One node per variable
  - One edge per add effect $a \in \text{add}(o)$
    - $\text{pcf}(o) \xrightarrow{o} a$



- $V = \{A, B, C, i, g\}$
- $I = \{i\}, G = \{g\}$
- $O = \{o_i, o_1, o_2, o_g\}$
  - $o_i = \langle i \rightarrow A \rangle_0$
  - $o_1 = \langle A \rightarrow B, C \rangle_1$
  - $o_2 = \langle A \rightarrow C \rangle_1$
  - $o_g = \langle B, C \rightarrow g \rangle_0$

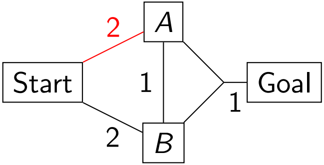
h^{lst}

- $h^{\mathrm{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
  - Pick a variable: B
  - Partition plan
    - Part dependent on B (P_B^+)
    - Part only used to add B (P_B^-)
    - Rest (P_B^0)
  - Find cheaper alternative for P_B^- given P_B^0



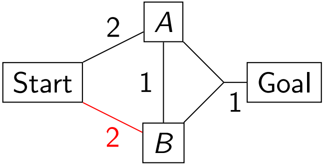
h^{lst}

- $h^{\mathrm{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
  - Pick a variable: B
  - Partition plan
    - Part dependent on B (P_B^+)
    - Part only used to add B (P_B^-)
    - Rest (P_B^0)
  - Find cheaper alternative for P_B^- given P_B^0



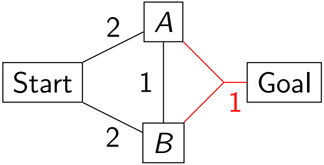
h^{lst}

- $h^{\text{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
    - Choose cheapest way to A
    - Choose cheapest way to B
    - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
    - Pick a variable: B
    - Partition plan
        - Part dependent on B (P_B^+)
        - Part only used to add B (P_B^-)
        - Rest (P_B^0)
    - Find cheaper alternative for P_B^- given P_B^0



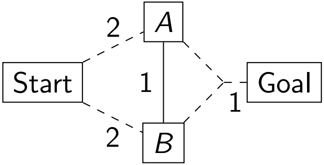
h^{lst}

- $h^{\text{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
    - Choose cheapest way to A
    - Choose cheapest way to B
    - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
    - Pick a variable: B
    - Partition plan
        - Part dependent on B (P_B^+)
        - Part only used to add B (P_B^-)
        - Rest (P_B^0)
    - Find cheaper alternative for P_B^- given P_B^0



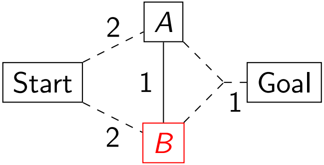
h^{lst}

- $h^{\text{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
  - Pick a variable: B
  - Partition plan
    - Part dependent on B (P_B^+)
    - Part only used to add B (P_B^-)
    - Rest (P_B^0)
  - Find cheaper alternative for P_B^- given P_B^0



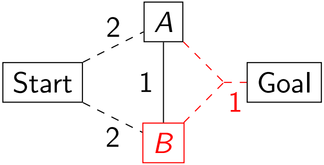
h^{lst}

- $h^{\mathrm{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
  - Pick a variable: B
  - Partition plan
    - Part dependent on B (P_B^+)
    - Part only used to add B (P_B^-)
    - Rest (P_B^0)
  - Find cheaper alternative for P_B^- given P_B^0



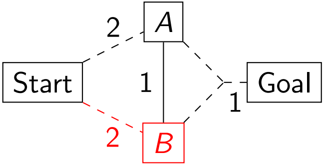
h^{lst}

- $h^{\text{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
  - Pick a variable: B
  - Partition plan
    - Part dependent on B (P_B^+)
    - Part only used to add B (P_B^-)
    - Rest (P_B^0)
  - Find cheaper alternative for P_B^- given P_B^0



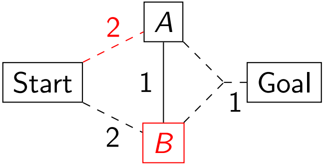
h^{lst}

- $h^{\text{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
  - Pick a variable: B
  - Partition plan
    - Part dependent on B (P_B^+)
    - Part only used to add B (P_B^-)
    - Rest (P_B^0)
  - Find cheaper alternative for P_B^- given P_B^0



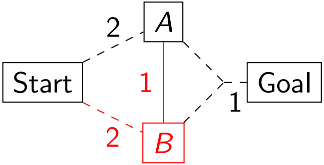
h^{lst}

- $h^{\text{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
  - Pick a variable: B
  - Partition plan
    - Part dependent on B (P_B^+)
    - Part only used to add B (P_B^-)
    - Rest (P_B^0)
  - Find cheaper alternative for P_B^- given P_B^0



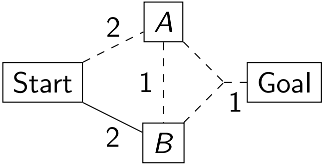
h^{lst}

- $h^{\text{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
  - Pick a variable: B
  - Partition plan
    - Part dependent on B (P_B^+)
    - Part only used to add B (P_B^-)
    - Rest (P_B^0)
  - Find cheaper alternative for P_B^- given P_B^0



h^{lst}

- $h^{\text{FF/add}}$ is greedy
- Path planning example
- First reach A and B, then go to goal
  - Choose cheapest way to A
  - Choose cheapest way to B
  - Go to Goal
- h^{lst} optimizes achiever choices
- Based on Steiner tree problem
  - Pick a variable: B
  - Partition plan
    - Part dependent on B (P_B^+)
    - Part only used to add B (P_B^-)
    - Rest (P_B^0)
  - Find cheaper alternative for P_B^- given P_B^0



Branch-and-Bound Search (Pseudo Code)

```
def BranchAndBound(problem):
    global variable interval = [0, ∞)
    global variable bestSolution = None
    initialNode = SearchNode(parent     = None
                             subproblem = problem)
    BranchAndBoundRecursive(initialNode)
    return bestSolution

def BranchAndBoundRecursive(node):
    if [node.calculateLowerBound(), ∞) ∩ interval == ∅:
        return
    if node.subproblem is solution:
        bestSolution = extractSolution(node)
        interval = interval ∩ [0, bestSolution.cost)
        return
    for sucessor in node.subproblem.successors:
      successorNode = SearchNode(parent     = node
                                 subproblem = sucessor)
      BranchAndBoundRecursive(successorNode)
```

Avoid unnecessary re-calculations

	$h^{\text{LM-cut}}$ computed
o was applied	
L_o undefined	Never
$L_o = \{o\}$	Never
$L_o = \{o, o_1, \dots, o_n\}$	Always
o was forbidden	
L_o undefined	If and only if $\text{cost}(o) = 0$
$L_o = \{o\}$	unsolvable
$L_o = \{o, o_1, \dots, o_n\}$	Always

Automatic Application of Operators - Unit Propagation

- Model checking technique
  - Set variable to last remaining value
- Analogy: LMs with only one element $l = \{o\}$
  - Every plan must contain o
  - Apply o without branching
  - Repeat until fixed point is reached
- Here: not necessary
  - Operator from smallest LM is selected
  - No re-calculation of $h^{\text{LM-cut}}$
  - Unsolvable task is detected immediately
- Could be useful with different heuristic
  - Isolated effect shows significant increase in coverage

Automatic Application of Operators - 0-Cost Operators

- Pure symbol heuristic
  - Literal only occurs positive $\Rightarrow$ set variable to true
- Analogy: Operators with base cost 0
  - Does not change solution cost
  - Cannot make applicable operators inapplicable
  - Automatic application

Results

	Coverage (%)
BnB	87.778
BnB (0-cost)	100.000
IDA*	87.778
IDA* (0-cost)	100.000

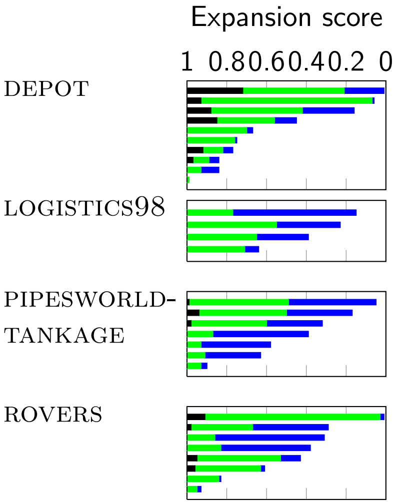
Evaluated on different domains

IDA*-Layer Analysis

- Three phases in IDA* node expansions:
  - Solution discovery (last layer)
  - Proof of optimality (second to last layer)
  - Avoidable part of proof (all other layers)
- Few expansions in *avoidable layers* (4.19% on average)
- Better search strategy with same operator order
  - Small expected improvement
- Different operator order
  - Can decrease expansions in all layers

IDA*-Layer Analysis (cont.)

- Bar length: Expansion score
  - Longer bar ∼ more expansions
- Coloring: relative size of IDA* layers
  - Blue ∼ Last layer
  - Green ∼ Second to last layer
  - Black ∼ All other layers



Restarts

- Operator order depends on random seed
- *Heavy-tailed* distribution for some tasks
  - Could benefit from random restarts
- Experiments
  - Different constant restart times
  - Geometrically increasing time
  - Universal restart strategy (Luby et al.) $[1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots]$
- No positive effect
  - Not enough tasks benefit from restarts

LOGISTICS98
PROB13