Exploiting Landmarks for Hybrid Planning

Mohamed Elkawkagy Pascal Bercher

Bernd Schattenberg Susanne Biundo

Institute of Artificial Intelligence

ulm university universität ulm

Motivation

- Landmarks in classical state based planning are facts that have to hold in some intermediate state of every plan that solves the given planning problem .
- Hierarchical Planning
  - Accomplish some set of tasks, rather than to achieve a goal
  - Based on the concepts of tasks and methods
  - High-level tasks are recursively decomposed down to sub-tasks
- Landmarks in hierarchical planning are tasks that occur in any sequence of decompositions leading from the initial plan to a solution plan.

Formal Framework (I)

- **Task**

  - Primitive task $\rightarrow$ action in state based planning
  - Abstract task $\rightarrow$ complex task (implemented by primitive tasks)

$$t(\tau) = \langle prec(t(\tau)), add(t(\tau)), del(t(\tau)) \rangle$$

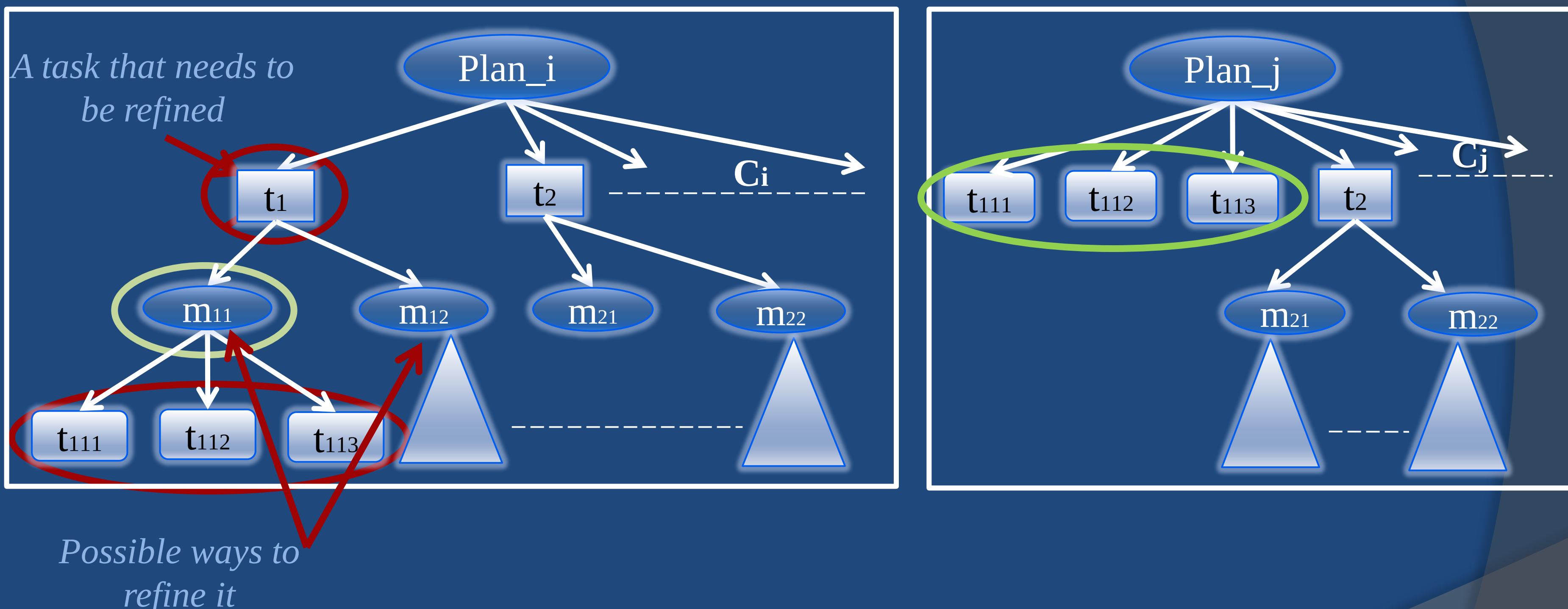
  - Difference: Primitive tasks are executed directly while abstract tasks require a sequence of primitive tasks to be performed

- **Plan** : $P = \langle S, C \rangle$
- **Method** : $m = \langle t, P \rangle$
- **Declarative domain model** : $D = (T, M)$

Formal Framework (Π)

Planning problem specification **$\Pi = < D, S_{init}, P_{init} >$**

Plan refinement $\rightarrow$ transforming the current plan into a more specific plan



Solution Plan of a planning problem **Π** is obtained by refining the initial plan **P_{init}** stepwise into a plan **$P = < S, C >$** that has only primitive plan steps and the set of constraints is consistent.

Planning Strategy compares the available plan refinements to choose the most suitable one to refine the current plan.

Our Approach

1) Analyzing task decomposition structure

- Building Task Decomposition Tree (TDT).

2) Extracting Landmark

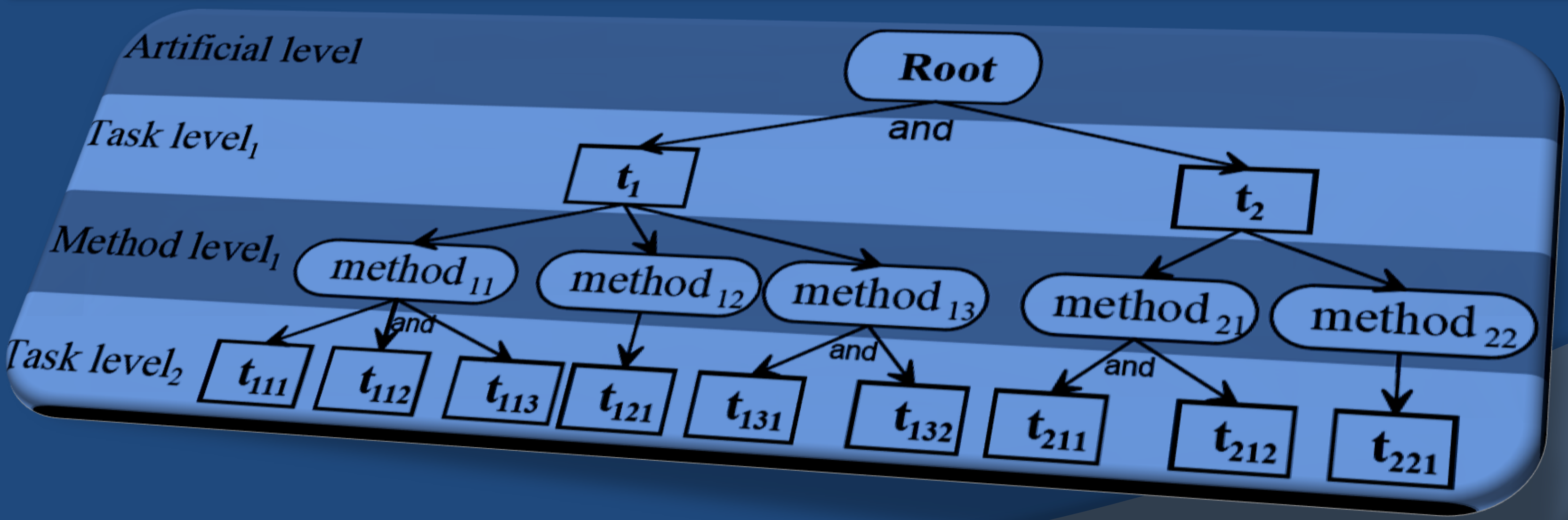
- Identify the essential tasks.

3) Exploiting Landmark information during the planning process

- Operating on reduced domain model by ignoring unsuccessful decomposition methods.
- Providing search strategy with focal information

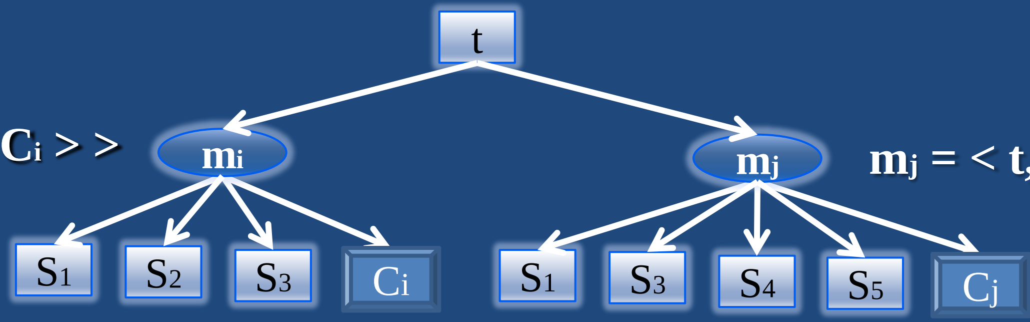
Task Decomposition Tree

Task Decomposition Tree (TDT): AND/OR tree that represents all possible ways to decompose the abstract tasks of P_{init} by methods in D until a primitive level is reached or a task is encountered that is already included in an upper level of the TDT



Operators

$m_i = \langle t, \langle S_i, C_i \rangle \rangle$



$m_j = \langle t, \langle S_j, C_j \rangle \rangle$

Common Task Set Operator: In the TDT, for two methods m_i, and m_j of a task t, the Common Task Set Operator is defined via:

$$m_i \widehat{\cap} m_j = Tasks(S_i) \cap Tasks(S_j)$$

Remaining Task Set Operator: In the TDT, for two methods m_i, and m_j of a task t, the Remaining Task Set Operator of m_i and m_j is defined via:

$$m_i \widehat{\cup} m_j = \{Tasks(S_i) \setminus (m_i \widehat{\cap} m_j), Tasks(S_j) \setminus (m_i \widehat{\cap} m_j)\}$$

Identifying Landmarks

Landmark Table: represents a mapping between an abstract task and its subtasks in the decomposition methods that refine this abstract task.

Landmark	Intersection(I)	Options(O)
$Task_1$	$\{t_{11}, t_{12}, \cdots\}$	$\{\{t_{h1}, t_{h2}, \cdots\}_{m\_h}, \{t_{l1}, t_{l2}, \cdots\}_{m\_l}, \cdots\}$
$\cdots$	$\cdots$	$\cdots$
$Task_n$	$\{t_{n1}, t_{n2}, \cdots\}$	$\{\{t_{k1}, t_{k2}, \cdots\}_{m\_k}, \{t_{o1}, t_{o2}, \cdots\}_{m\_o}, \cdots\}$

- The intersection **I(t)** contains those subtasks which occur on every possible path of decompositions that transform **t** into a primitive plan.

Identifying Landmarks

Landmark Table: represents a mapping between an abstract task and its subtasks in the decomposition methods that refine this abstract task.

Landmark	Intersection(I)	Options(O)
$Task_1$	$\{t_{11}, t_{12}, \cdots\}$	$\{\{t_{h1}, t_{h2}, \cdots\}_{m\_h}, \{t_{l1}, t_{l2}, \cdots\}_{m\_l}, \cdots\}$
$\cdots$	$\cdots$	$\cdots$
$Task_n$	$\{t_{n1}, t_{n2}, \cdots\}$	$\{\{t_{k1}, t_{k2}, \cdots\}_{m\_k}, \{t_{o1}, t_{o2}, \cdots\}_{m\_o}, \cdots\}$

- The remaining task sets **R(t)** (aka options) represent sets of those subtasks that optionally occur when decomposing an abstract task towards a solution plan.
- Every set is indexed by the name of the method which contains these subtasks.

Landmark Extraction

Input : A task decomposition tree TDT.
Output: The filled landmark table LT.

```
LT ← ∅, infeasible ← ∅
for i ← 1 to TDT.maxDepth() do
    foreach abstract task t in level i of TDT do
        if LT contains an entry for t then continue
        repeat
            Let M be the methods of t in the TDT.
            I(t) ← ⋂̂_{m∈M} m
            R(t) ← ( ⋃̂_{m∈M} m ) \ {∅}
            foreach primitive task t' ∈ I(t) do
                if t' can be proven infeasible then
                    remove all m ∈ M from the TDT, including all sub-nodes.
                    break
                end
            end
            foreach remaining task set r ∈ R(t) do
                foreach primitive task t' ∈ r do
                    if t' can be proven infeasible then
                        remove the method m = ⟨t, P⟩, with
                        Tasks(P) = I(t) ∪ r from the TDT, including all
                        sub-nodes.
                        continue
                    end
                end
            end
        until no method was removed from TDT
        LT ← LT ∪ {(t, I(t), R(t))}
        if I(t) = R(t) = ∅ then
            infeasible ← infeasible ∪ {t}
        end
    end
end
return propagate(LT,TDT,infeasible)
```

It runs iteratively through all levels of the TDT until the maximum level has been reached.

Landmark Extraction

Input : A task decomposition tree TDT.
Output: The filled landmark table LT.

```
LT ← ∅, infeasible ← ∅
for i ← 1 to TDT.maxDepth() do
    foreach abstract task t in level i of TDT do
        if LT contains an entry for t then continue
        repeat
            Let M be the methods of t in the TDT.
            I(t) ← ⋂̂_{m∈M} m
            R(t) ← (⋃̂_{m∈M} m) \ {∅}
            foreach primitive task t' ∈ I(t) do
                if t' can be proven infeasible then
                    remove all m ∈ M from the TDT, including all sub-nodes.
                    break
                end
            end
            foreach remaining task set r ∈ R(t) do
                foreach primitive task t' ∈ r do
                    if t' can be proven infeasible then
                        remove the method m = ⟨t, P⟩, with
                        Tasks(P) = I(t) ∪ r from the TDT, including all
                        sub-nodes.
                        continue
                    end
                end
            end
        until no method was removed from TDT
        LT ← LT ∪ {(t, I(t), R(t))}
        if I(t) = R(t) = ∅ then
            infeasible ← infeasible ∪ {t}
        end
    end
end
return propagate(LT,TDT,infeasible)
```

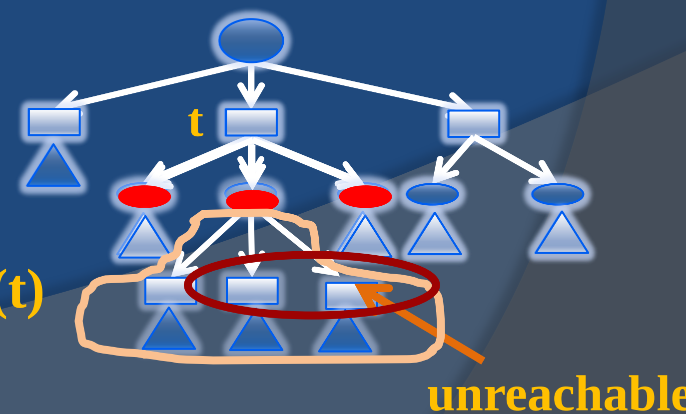
- The Methods M={m_1, m_2, …, m_n} that decompose a current task t are collected.
- The intersection I(t) and remaining tasks sets R(t) are computed.

Landmark Extraction

Input : A task decomposition tree TDT.
Output: The filled landmark table LT.
LT $\leftarrow \emptyset$, `infeasible` $\leftarrow \emptyset$
for $i \leftarrow 1$ **to** *TDT.maxDepth()* **do**
  foreach *abstract task t in level i of TDT* **do**
    if *LT contains an entry for t* **then continue**
    repeat
      Let M be the methods of t in the TDT.
      $I(t) \leftarrow \hat{\bigcap}_{m \in M} m$
      $R(t) \leftarrow (\hat{\bigcup}_{m \in M} m) \setminus \{\emptyset\}$
      foreach *primitive task* $t' \in I(t)$ **do**
        if t' *can be proven infeasible* **then**
          remove all $m \in M$ from the TDT, including all sub-nodes.
          break
        end
      end
      foreach *remaining task set* $r \in R(t)$ **do**
        foreach *primitive task* $t' \in r$ **do**
          if t' *can be proven infeasible* **then**
            remove the method $m = \langle t, P \rangle$, with
            $Tasks(P) = I(t) \cup r$ from the TDT, including all
            sub-nodes.
            continue
          end
        end
      end
    until *no method was removed from TDT*
    LT $\leftarrow$ LT $\cup \{(t, I(t), R(t))\}$
    if $I(t) = R(t) = \emptyset$ **then**
      `infeasible` $\leftarrow$ `infeasible` $\cup \{t\}$
    end
  end
end
return *propagate(LT,TDT,infeasible)*

- The reachability of each primitive task t' in I(t) is investigated by estimating the achievability of the preconditions of a task.

  If test succeeds: TDT is updated by pruning *all* methods of t (this triggers further updates).



Landmark Extraction

```
Input   : A task decomposition tree TDT.
Output: The filled landmark table LT.
LT ← ∅, infeasible ← ∅
for i ← 1 to TDT.maxDepth() do
  foreach abstract task t in level i of TDT do
    if LT contains an entry for t then continue
    repeat
      Let M be the methods of t in the TDT.
      I(t) ← ⋂_{m∈M} m
      R(t) ← (⋃_{m∈M} m) \ {∅}
      foreach primitive task t' ∈ I(t) do
        if t' can be proven infeasible then
          remove all m ∈ M from the TDT, including all sub-nodes.
          break
        end
      end
      foreach remaining task set r ∈ R(t) do
        foreach primitive task t' ∈ r do
          if t' can be proven infeasible then
            remove the method m = ⟨t, P⟩, with
            Tasks(P) = I(t) ∪ r from the TDT, including all
            sub-nodes.
            continue
          end
        end
      end
    until no method was removed from TDT
    LT ← LT ∪ {(t, I(t), R(t))}
    if I(t) = R(t) = ∅ then
      infeasible ← infeasible ∪ {t}
    end
  end
end
return propagate(LT,TDT,infeasible)
```

- The reachability of each primitive task t' in each set r in R(t) is investigated by estimating the achievability of the preconditions of a task.

  If test succeeds: TDT is updated by pruning *failed* methods of t (this *may* trigger further updates).

Landmark Extraction

```
Input  : A task decomposition tree TDT.
Output: The filled landmark table LT.
LT ← ∅, infeasible ← ∅
for i ← 1 to TDT.maxDepth() do
  foreach abstract task t in level i of TDT do
    if LT contains an entry for t then continue
    repeat
      Let M be the methods of t in the TDT.
      I(t) ← ⋂̂_{m∈M} m
      R(t) ← ( ⋃̂_{m∈M} m) \ {∅}
      foreach primitive task t′ ∈ I(t) do
        if t′ can be proven infeasible then
          remove all m ∈ M from the TDT, including all sub-nodes.
          break
        end
      end
      foreach remaining task set r ∈ R(t) do
        foreach primitive task t′ ∈ r do
          if t′ can be proven infeasible then
            remove the method m = ⟨t, P⟩, with
            Tasks(P) = I(t) ∪ r from the TDT, including all
            sub-nodes.
            continue
          end
        end
      end
    until no method was removed from TDT
    LT ← LT ∪ {(t, I(t), R(t))}
    if I(t) = R(t) = ∅ then
      infeasible ← infeasible ∪ {t}
    end
  end
end
return propagate(LT,TDT,infeasible)
```

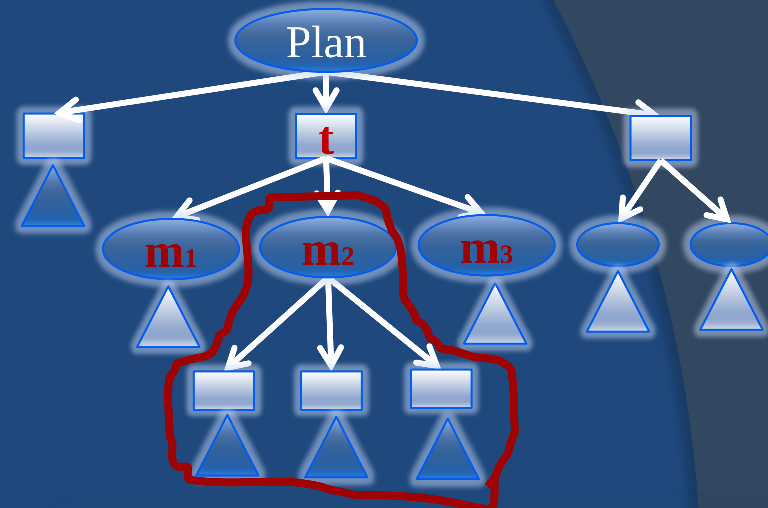
- Landmark table is updated

Landmark	*Intersection*(I)	*Options*(O)
$Task_1$	$\{t_{11}, t_{12}, \cdots\}$	$\{\{t_{h1}, t_{h2}, \cdots\}_{m\_h}, \{t_{l1}, t_{l2}, \cdots\}_{m\_l}, \cdots\}$
$\cdots$	$\cdots$	$\cdots$
$Task_n$	$\{t_{n1}, t_{n2}, \cdots\}$	$\{\{t_{k1}, t_{k2}, \cdots\}_{m\_k}, \{t_{o1}, t_{o2}, \cdots\}_{m\_o}, \cdots\}$
t	**I(t)**	**R(t)**

- Recursive procedure is called for propagating the results of the feasibility analysis.

Landmark Exploitation – Model Reduction

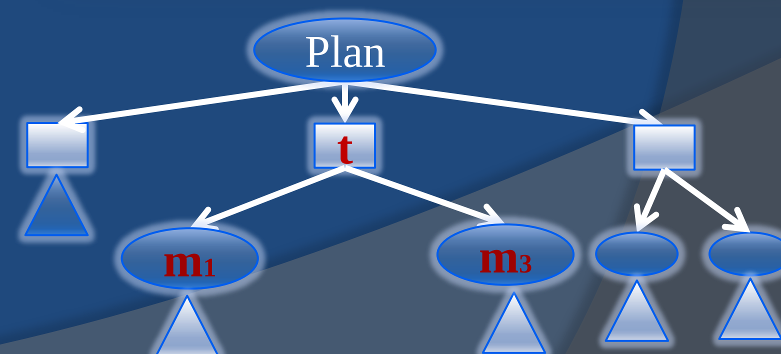
- Hierarchical planning refines an abstract task by considering all decomposition methods in the domain model that implement it.



- The process of refining abstract tasks in our system is deployed with a reference to the Landmark Table of the planning problem.

Landmark	*Intersection(I)*	*Options(O)*
$Task_1$	$\{t_{11}, t_{12}, \cdots\}$	$\{\{t_{h1}, t_{h2}, \cdots\}_{m\_h}, \{t_{l1}, t_{l2}, \cdots\}_{m\_l}, \cdots\}$
$\cdots$	$\cdots$	$\cdots$
$Task_n$	$\{t_{n1}, t_{n2}, \cdots\}$	$\{\{t_{k1}, t_{k2}, \cdots\}_{m\_k}, \{t_{o1}, t_{o2}, \cdots\}_{m\_o}, \cdots\}$

- It operates on a *reduced* set of applicable methods according to the respective options R(t) in the Landmark Table



Landmark Exploitation – Strategies

- Modification Ordering Functions implement preferences on refinements
- Landmark-Aware Strategy
  - For two given modification m_i and m_j, let f_i and f_j be the addressed (abstract task) flaws
  - Let t_i and t_j be the tasks referenced by f_i and f_j, then

$$R^*(t) = \{r \mid r \in R(t) \text{ for } (t, I(t), R(t)) \in \mathrm{LT} \text{ or } r \in R(t') \text{ for } (t', I(t'), R(t')) \in \mathrm{LT},\ t' \in r',\ \text{and } r' \in R^*(t)\}.$$

Criterion: $|R^*(t_i)| < |R^*(t_j)|$

  - Implements the least commitment principle by prefering a lower branching factor estimate

Evaluation

- We run our evaluations over two distinguished benchmark domains:
- UM-Translog
    - Logistics, difficulty of problems due to various transportation means.
- Satellite
    - Earth observation, problems become difficult when modeling a repetition of observations: small number of methods is used multiple times in different contexts of the plan.

Evaluation – Model Reduction

UM-Translog

Problem	abstr. Tasks (of 21)	Methods (of 51)
Regular Truck Problems Hopper Truck, Auto Truck, Regular Truck_3 Locations Regular Truck_2 Region, Regular Truck_1, Regular Truck_2,	12 (57%)	30 (59%)
Various Truck Type Problems Flatbed Truck, Armored-R-Truck	12 (57%)	32 (63%)
Traincar Problems Auto Traincar, Mail Traincar, Auto Traincar bis, Refrigerated Regular Traincar	14 (67%)	32 (63%)
Airplane Problem Airplane	14 (67%)	37 (73%)

Average performance improvement over all strategies and problems is about 40% in search space size and about 30% in CPU time.

Satellite does not benefit significantly from landmark technique due to a shallow decomposition hierarchy

Evaluation – Strategies

UM-Translog

Mod. ordering function f^{ModOrd}	Refrigerated Regular Traincar Space	Refrigerated Regular Traincar Time	Auto Traincar bis Space	Auto Traincar bis Time	Airplane Space	Airplane Time
lcf	90	225	227	926	247	798
hz	**76**	**196**	701	1616	345	1323
lm_1	**72**	**180**	**183**	**608**	**142**	**441**
lm_2	89	212	**158**	**543**	189	676
ems	500	1048	2558	6447	784	2517
da	588	1958	184	705	172	620
du	307	775	1390	4018	643	2134
SHOP	173	353	247	963	**150**	**450**

In all cases, one of the Landmark-Aware strategies outperforms all benchmark candidates.

Conclusion

- Landmark table is generated automatically.
- Avoids unsuitable plan refinements.
- Domain- and strategy independent technique.
- Information helps any hierarchical planner to improve its performance.
- Significance performance gain, especially for problems with a deep hierarchy of tasks.
- … but many open issues left