Heuristics for Cost-Optimal Classical Planning Based on Linear Programming (from ICAPS-14)

Florian Pommerening\textsuperscript{1} Gabriele Röger\textsuperscript{1} Malte Helmert\textsuperscript{1}

Blai Bonet\textsuperscript{2}

\textsuperscript{1}Universität Basel
\textsuperscript{2}Universidad Simón Bolívar

Control Problem in Autonomous Behavior

Let's consider an autonomous agent embedded in environment.

Agent faces:
- full or partial information about state of the system
- deterministic or non-deterministic effects of actions
- hard or soft goals
- discrete or continuous time
- etc

Key problem for agent is how to select next action to execute.

This is the control problem in autonomous behavior.
Three Approaches

Programming-based: specify control by hand

- **Advantage:** simple domain knowledge is easy to express
- **Disadvantage:** programmer cannot anticipate all situations

Learning-based: learn control from experience

- **Advantage:** requires little knowledge in principle
- **Disadvantage:** right features needed, incomplete information is problematic, and learning is slow

Model-based: specify problem by hand, derive control automatically

- **Advantage:** flexible, clear, and domain-independent
- **Disadvantage:** need a model; computationally intractable in general

Model-based approach to intelligent behavior called Planning
Classical Planning: Simplest Model

Deterministic actions, complete knowledge, discrete time, hard goals

Instance is tuple $\langle S, A, s_{init}, S_G, f, cost \rangle$:

- finite **state space** $S$
- known initial state $s_{init} \in S$
- actions $A(s) \subseteq A$ executable at state $s$
- subset $S_G \subseteq S$ of **goal states**
- deterministic transition function $f : S \times A \rightarrow S$ such that $f(s, a)$ is state after applying action $a \in A(s)$ in state $s$
- non-negative costs $cost(s, a)$ for applying action $a$ in state $s$

Solution (plan) is **sequence of actions** that map initial state into goal

Cost is the **sum of costs** of the actions in the plan
Factored Languages

STRIPS and SAS\(^+\) are languages based on propositions and multi-valued variables respectively.

**Atoms** in STRIPS are propositions; in SAS\(^+\) are assignments \(X = x\).

Description of instance, either STRIPS or SAS\(^+\), specifies:

- initial state

- goal description as subset of **atoms to achieve**

- finite set \(O\) of operators; for each operator \(o \in O\):
  
  - **precondition** \(\text{pre}(o) \subseteq \text{Atoms}\) that must hold for \(o\) to be executable
  
  - **effects** \(\text{post}(o) \subseteq \text{Atoms}^+ \cup \text{Atoms}^-\) that define the transitions

- non-negative costs \(c(o)\) for applying operators \(o \in O\)
Example: Moving Packages


Initial state: pkg-at-B, truck-at-A

Goal: pkg-at-A, truck-at-B


Costs: all operators have unit costs
Example: Moving Packages

Operator load-B:

- **precondition:** truck-at-B, pkg-at-B
- **positive effects:** pkg-in-truck
- **negative effects:** pkg-at-B
Solvers for Classical Planning

State-of-the-art solvers do **forward search in state space** to find path from initial state to a goal state (in exponential implicit graph)

**Satisficing planning:** suboptimal algorithms combining:
– weighted heuristics and re-starting
– multiple open lists ordered by different evaluation functions
– other techniques

**Optimal planning:** A* preferred over IDA* because:
– potentially huge number of **duplicate nodes** in search tree
– heuristics are **relatively expensive** to compute
Contribution

**Novel framework** for admissible heuristics that:

- it is based on **integer/linear programming**
- it captures most state-of-the-art heuristics for optimal planning
- it permits combination of existing heuristics into novel heuristics
- it permits analysis and deeper understanding of heuristics

New heuristics dominate existing heuristics and are cost effective
Heuristics calculated using LPs

Heuristic value $h(s)$ for state $s$ is value of LP of the form:

$$\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad [\text{set of linear inequalities}] 
\end{align*}$$

where $f(x)$ is linear function

Each time a value $h(s)$ is required, such an LP is solved

When solving a hard planning problem, thousands/millions of LPs are solved
For each operator $o$ in the problem we consider a **non-negative integer variable** variable $Y_o$. The set of all such variables is $\mathcal{Y}$.

For plan $\pi$, let $Y^{\pi}_o$ be the **number of occurrences** of $o$ in $\pi$.
Operator Counting Constraints (OCCs)

For each operator $o$ in the problem we consider a non-negative integer variable $Y_o$. The set of all such variables is $\mathcal{Y}$.

For plan $\pi$, let $Y_o^\pi$ be the number of occurrences of $o$ in $\pi$.

A set $C$ of linear inequalities over $\mathcal{Y}$ (and possibly other variables) is called an operator counting constraint (OCC) for state $s$ if:

- for each plan $\pi$ for $s$, there is a solution of $C$ with $Y_o = Y_o^\pi$. 
Operator Counting Constraints (OCCs)

For each operator $o$ in the problem we consider a non-negative integer variable $Y_o$. The set of all such variables is $\mathcal{Y}$.

For plan $\pi$, let $Y_o^\pi$ be the number of occurrences of $o$ in $\pi$.

A set $C$ of linear inequalities over $\mathcal{Y}$ (and possibly other variables) is called an operator counting constraint (OCC) for state $s$ if:

- for each plan $\pi$ for $s$, there is a solution of $C$ with $Y_o = Y_o^\pi$.

A constraint system for state $s$ is a set of OCCs for $s$ where the common variables between OCCs are operator-counting variables $Y_o$. 
The constraints:

\[ Y_{\text{drive-A-B}} \geq 1 \]
\[ Y_{\text{load-B}} \geq 1 \]
\[ Y_{\text{unload-A}} \geq 1 \]

is OCC for the initial state \( s_{\text{init}} \)
Integer Programs, LP Relaxations, and Heuristics

The integer program for constraint system $C$ is $\text{IP}_C$:

$$\text{minimize } \sum_o \text{cost}(o) \times Y_o \text{ subject to } C, \ Y_o \in \mathbb{Z}^*$$

The linear program $\text{LP}_C$ is the linear relaxation of $\text{IP}_C$ (i.e. $\text{IP}_C$ without the constraints $Y_o \in \mathbb{Z}^*$)
The integer program for constraint system $C$ is $\text{IP}_C$:

$$\text{minimize } \sum_o \text{cost}(o) \times Y_o \text{ subject to } C, \ Y_o \in \mathbb{Z}^*$$

The linear program $\text{LP}_C$ is the linear relaxation of $\text{IP}_C$ (i.e. $\text{IP}_C$ without the constraints $Y_o \in \mathbb{Z}^*$)

Let $C$ be function that maps states $s$ into constraint systems $C(s)$ for $s$

Heuristic $h_C^{\text{LP}}$ is the function that maps states $s$ into value of $\text{LP}_{C(s)}$
The integer program for constraint system $C$ is $\text{IP}_C$:

\[
\text{minimize } \sum_o \text{cost}(o) \times Y_o \quad \text{subject to } C, \ Y_o \in \mathbb{Z}^*
\]

The linear program $\text{LP}_C$ is the linear relaxation of $\text{IP}_C$ (i.e. $\text{IP}_C$ without the constraints $Y_o \in \mathbb{Z}^*$)

Let $C$ be function that maps states $s$ into constraint systems $C(s)$ for $s$

Heuristic $h^\text{LP}_C$ is the function that maps states $s$ into value of $\text{LP}_{C(s)}$

**Theorem**

The heuristic $h^\text{LP}_C$ is **admissible** for any function $C$ that maps states $s$ into constraint systems for $s$ and it is **polytime** computable (in $|C(s)|$)
Compilation of Heuristics into OCCs

In paper we show how to compile into OCCs the following heuristics:

- **Landmark heuristics with optimal cost partitioning**

- **Abstractions and optimal cost partitioning for abstractions**
  [Edelkamp, 2001; Katz & Domshlak, 2009; Pommerening et al., 2013; Helmert et al., 2014]

- **Post-hoc optimization heuristics** [Pommerening et al., 2013]

- **State equation heuristic** [van den Briel et al., 2007; B., 2013; B. & van den Briel, 2014]

- **Delete relaxation constraints** [Imai & Fukunaga, 2014]

Some compilations are straightforward, others are more complex
Delete-relaxation heuristics
- $h_{\text{max}}$, additive $h_{\text{max}}$, ...

Critical-path heuristics
- $h^1, h^2, \ldots, h^m, \ldots$

Landmark heuristics
- $h^L, h^{LA}, h^{\text{LM-cut}}, \ldots$

Abstraction heuristics
- PDBs, merge-and-shrink, structural patterns, \ldots
Example of OCCs: Landmarks

A **disjunctive action landmark** for state $s$ is a subset $L$ of actions such that every plan for $s$ contains at least one action in $L$.

For example, $\{\text{drive-A-B}\}$ is a disjunctive action landmark for $s_{\text{init}}$ in the example as **every plan** must drive the truck from location A to B.
Example of OCCs: Landmarks

A **disjunctive action landmark** for state \( s \) is a subset \( L \) of actions such that every plan for \( s \) contains at least one action in \( L \).

For example, \{drive-A-B\} is a disjunctive action landmark for \( s_{init} \) in the example as **every plan** must drive the truck from location A to B.

If \( \mathcal{L} \) is a set of disjunctive action landmarks for state \( s \), then

\[
\sum_{o \in L} Y_o \geq 1
\]

for each landmark \( L \in \mathcal{L} \) is an OCC for state \( s \).
Example of OCCs: Landmarks

A **disjunctive action landmark** for state $s$ is a subset $L$ of actions such that every plan for $s$ contains at least one action in $L$.

For example, $\{\text{drive-A-B}\}$ is a disjunctive action landmark for $s_{\text{init}}$ in the example as every plan must drive the truck from location A to B.

If $\mathcal{L}$ is a set of disjunctive action landmarks for state $s$, then

$$\sum_{o \in L} Y_o \geq 1$$

for each landmark $L \in \mathcal{L}$ is an OCC for state $s$.

**Remark:** LP for this OCC is the dual of the LP that computes the **optimal cost partitioning** for the collection $\mathcal{L}$ of landmarks.
Example of OCCs: Net Change Constraints

Number of times atoms appear/disappear along a plan are subject to constraints

For example, each time the truck moves right, the atom truck-at-B appears and the atom truck-at-A disappears
Example of OCCs: Net Change Constraints

Number of times atoms appear/disappear along a plan are subject to constraints.

For example, each time the truck moves right, the atom truck-at-B appears and the atom truck-at-A disappears.

Since truck is initially at A and goal is to have it at B, for valid plan $\pi$

$$Y_{\text{drive-A-B}}^{\pi} + Y_{\text{drive-B-A}}^{\pi} \geq 1$$
Example of OCCs: Net Change Constraints

Number of times atoms **appear/disappear** along a plan are subject to constraints.

Likewise, a plan $\pi$ cannot unload the package more times than it is loaded into the truck:

$$Y_{\text{load-A}}^\pi + Y_{\text{load-B}}^\pi - Y_{\text{unload-A}}^\pi - Y_{\text{unload-B}}^\pi \geq 0$$
Example of OCCs: State Equation Heuristic

For each atom $p$, there is a net change constraint $C_p$:

\[
\sum_{o \text{ adds } p} Y_o + \sum_{o \text{ may add } p} Y_o \geq \sum_{o \text{ consumes } p} Y_o \geq \Delta(p)
\]

where $\Delta(p)$ is **net change** for $p$ between goal and initial config., and

- $o$ adds $p$ iff $\text{pre}(o) \models \neg p$ and $p \in \text{post}(o)$
- $o$ consumes $p$ iff $\text{pre}(o) \models p$ and $\neg p \in \text{post}(o)$
- $o$ may add $p$ iff $\text{pre}(o) \not\models \neg p$ and $p \in \text{post}(o)$
Example of OCCs: State Equation Heuristic

For each atom $p$, there is a net change constraint $C_p$:

$$\sum_{o \text{ adds } p} Y_o + \sum_{o \text{ may add } p} Y_o - \sum_{o \text{ consumes } p} Y_o \geq \Delta(p)$$

where $\Delta(p)$ is net change for $p$ between goal and initial config., and

- $o$ adds $p$ iff $pre(o) \models \neg p$ and $p \in post(o)$
- $o$ consumes $p$ iff $pre(o) \models p$ and $\neg p \in post(o)$
- $o$ may add $p$ iff $pre(o) \not\models \neg p$ and $p \in post(o)$

The OCC for the state equation heuristic (SEQ) is the collection of all constraints $C_p$ for atoms $p$.
Experimental Results

- Experiments performed on Intel Xeon E5-2660 processors (2.2 GHz)

- **Time limit** of 30 minutes and **memory limit** of 2Gb

- Single OCCs:
  - **SEQ** Constraints for state-equation heuristic
  - **PhO-Sys**\(^1\) Post-hoc optimization constraints for projections on goal variables
  - **PhO-Sys**\(^2\) Post-hoc optimization constraints for projections up to 2 variables
  - **LMC** Landmark constraints for LM-cut landmarks
  - **OPT-Sys**\(^1\) Optimal cost partitioning for projections of goal variables
# Experimental Results: Coverage

<table>
<thead>
<tr>
<th></th>
<th>single OCCs</th>
<th></th>
<th>combined OCCs</th>
<th></th>
<th>( h_{\text{LM-cut}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SEQ</td>
<td>PhO-Sys(^1)</td>
<td>PhO-Sys(^2)</td>
<td>LMC</td>
<td>OPT-Sys(^1)</td>
</tr>
<tr>
<td>barman (20)</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>elevators (20)</td>
<td>7</td>
<td>9</td>
<td>16</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>floortile (20)</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>nomystery (20)</td>
<td>10</td>
<td>11</td>
<td>16</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>openstacks (20)</td>
<td>11</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>parcprinter (20)</td>
<td>20</td>
<td>11</td>
<td>13</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>parking (20)</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>pegsol (20)</td>
<td>18</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>scanalyzer (20)</td>
<td>11</td>
<td>9</td>
<td>4</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>sokoban (20)</td>
<td>16</td>
<td>19</td>
<td>20</td>
<td>20</td>
<td>13</td>
</tr>
<tr>
<td>tidybot (20)</td>
<td>7</td>
<td>13</td>
<td>14</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>transport (20)</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>visitall (20)</td>
<td>17</td>
<td>16</td>
<td>16</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>woodworking (20)</td>
<td>9</td>
<td>5</td>
<td>10</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>\textbf{Sum IPC 2011} (280)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\textbf{IPC 1998–2008} (1116)</td>
<td>487</td>
<td>446</td>
<td>478</td>
<td>586</td>
<td>357</td>
</tr>
<tr>
<td>\textbf{Sum} (1396)</td>
<td>630</td>
<td>587</td>
<td>631</td>
<td>744</td>
<td>443</td>
</tr>
</tbody>
</table>
Experimental Results: Synergy

Number of expansions (excluding nodes on the final $f$-layer)

Numbers ($x/y$) say that among the $y$ solved tasks, $x$ were solved with **perfect heuristic estimates**
Discussion

- Framework based on IP/LP that subsumes most state-of-the-art heuristics for optimal planning

- Heuristics can be synergistically combined inside the framework

- New combined heuristics dominate component heuristics and are cost effective

- Framework permits analysis of heuristics

- Critical-path heuristics had not been captured in framework

- **Future work**: adding more constraints to improve lower bounds (heuristics) and compile critical-path heuristics into OCCs
Thanks. Questions?