Heuristics for Cost-Optimal Classical Planning Based on Linear Programming

(from ICAPS-14)

Florian Pommerening[1] Gabriele Röger[1] Malte Helmert[1]

Blai Bonet[2]

[1]Universität Basel
[2]Universidad Simón Bolívar

IJCAI Sister Conf. Track. Buenos Aires, Argentina. 2015

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 $\langle S, A, s_{\mathit{init}}, S_G, f, cost) \rangle$:

- finite **state space** S
- **known** initial state $s_{\mathit{init}} \in S$
- actions $A(s) \subseteq A$ executable at state s
- subset $S_G \subseteq S$ of **goal states**
- **deterministic** transition function $f : S \times A \to S$ 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** pre$(o) \subseteq$ Atoms that must hold for o to be executable
- **effects** post$(o) \subseteq$ Atoms$^+ \cup$ Atoms$^-$ that define the transitions

– non-negative costs $c(o)$ for applying operators $o \in O$

Example: Moving Packages

Atoms: pkg-at-A, pkg-at-B, pkg-in-truck, truck-at-A, truck-at-B

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

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

Operators: load-A, load-B, unload-A, unload-B, drive-A-B, drive-B-A

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:

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

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

Operator Counting Constraints (OCCs)

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 π, let Y_o^π be the **number of occurrences** of o in π

Operator Counting Constraints (OCCs)

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 π, let Y_o^π be the **number of occurrences** of o in π

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 π for s, there is a solution of C with $\mathsf{Y}_o = Y_o^\pi$

Operator Counting Constraints (OCCs)

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 π, let Y_o^π be the **number of occurrences** of o in π

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 π for s, there is a solution of C with $\mathsf{Y}_o = Y_o^\pi$

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

Example: Moving Packages

The constraints:

$$\begin{aligned} \mathsf{Y}_{\text{drive-A-B}} &\geq 1 \\ \mathsf{Y}_{\text{load-B}} &\geq 1 \\ \mathsf{Y}_{\text{unload-A}} &\geq 1 \end{aligned}$$

is OCC for the initial state s_{init}

Integer Programs, LP Relaxations, and Heuristics

The **integer program** for constraint system C is IP_C:

$$\text{minimize} \quad \sum_{o} cost(o) \times \mathsf{Y}_o \quad \text{subject to} \quad C, \quad \mathsf{Y}_o \in \mathbb{Z}^*$$

The **linear program** LP_C is the **linear relaxation** of IP_C
(i.e. IP_C without the constraints $\mathsf{Y}_o \in \mathbb{Z}^*$)

Integer Programs, LP Relaxations, and Heuristics

The **integer program** for constraint system C is IP_C:

$$\text{minimize} \quad \sum_{o} cost(o) \times \mathsf{Y}_o \quad \text{subject to} \quad C, \quad \mathsf{Y}_o \in \mathbb{Z}^*$$

The **linear program** LP_C is the **linear relaxation** of IP_C
(i.e. IP_C without the constraints $\mathsf{Y}_o \in \mathbb{Z}^*$)

Let $\mathcal{C}$ be function that maps states s into constraint systems $\mathcal{C}(s)$ for s

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

Integer Programs, LP Relaxations, and Heuristics

The **integer program** for constraint system C is IP_C:

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

The **linear program** LP_C is the **linear relaxation** of IP_C
(i.e. IP_C without the constraints $\mathsf{Y}_o \in \mathbb{Z}^*$)

Let $\mathcal{C}$ be function that maps states s into constraint systems $\mathcal{C}(s)$ for s

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

Theorem

The heuristic $h^{\mathsf{LP}}_{\mathcal{C}}$ is **admissible** *for any function $\mathcal{C}$ that maps states s into constraint systems for s and it is* **polytime** *computable (in $|\mathcal{C}(s)|$)*

Compilation of Heuristics into OCCs

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

- **Landmark heuristics with optimal cost partitioning** [Karpas & Domshlak, 2009; Helmert & Domshlak, 2009; B. & Helmert, 2010]
- **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

Helmert & Domshlak's Classification (2009)

Delete-relaxation heuristics

– h_{max}, additive h_{max}, ...

Critical-path heuristics

– $h^1, h^2, \ldots, h^m, \ldots$

Landmark heuristics

– h^L, h^{LA}, $h^{\text{LM-cut}}$, ...

Abstraction heuristics

– PDBs, merge-and-shrink, structural patterns, ...

Example of OCCs: Landmarks

A **disjuntive 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

Example of OCCs: Landmarks

A **disjuntive 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

$$\sum_{o \in L} \mathsf{Y}_o \geq 1$$

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

Example of OCCs: Landmarks

A **disjuntive 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

$$\sum_{o \in L} \mathsf{Y}_o \ \geq \ 1$$

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

<u>Remark:</u> 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 π

$$Y^{\pi}_{\text{drive-A-B}} + Y^{\pi}_{\text{drive-B-A}} \geq 1$$

Example of OCCs: Net Change Constraints

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

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

$$Y^{\pi}_{\text{load-A}} + Y^{\pi}_{\text{load-B}} - Y^{\pi}_{\text{unload-A}} - Y^{\pi}_{\text{unload-B}} \geq 0$$

Example of OCCs: State Equation Heuristic

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

$$\sum_{o \text{ adds } p} \mathsf{Y}_o + \sum_{o \text{ may add } p} \mathsf{Y}_o - \sum_{o \text{ consumes } p} \mathsf{Y}_o \;\geq\; \Delta(p)$$

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

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

Example of OCCs: State Equation Heuristic

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

$$\sum_{o \text{ adds } p} \mathsf{Y}_o + \sum_{o \text{ may add } p} \mathsf{Y}_o - \sum_{o \text{ consumes } p} \mathsf{Y}_o \;\geq\; \Delta(p)$$

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

- o adds p iff $\mathit{pre}(o) \vDash \neg p$ and $p \in \mathit{post}(o)$
- o consumes p iff $\mathit{pre}(o) \vDash p$ and $\neg p \in \mathit{post}(o)$
- o may add p iff $\mathit{pre}(o) \nvDash \neg p$ and $p \in \mathit{post}(o)$

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

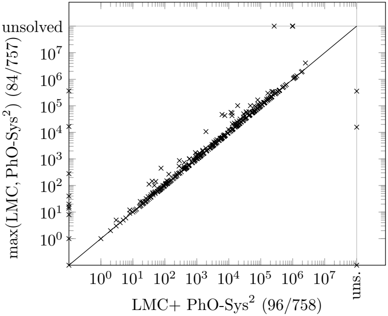
  LMC Landmark constraints for LM-cut landmarks

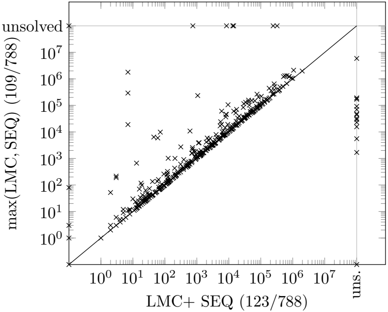
  OPT-Sys^1 Optimal cost partitioning for projections of goal variables

Experimental Results: Coverage

	single OCCs					combined OCCs				
	SEQ	PhO-Sys^1	PhO-Sys^2	LMC	OPT-Sys^1	LMC+ PhO-Sys^2	LMC+ SEQ	PhO-Sys^2+ SEQ	LMC+ PhO-Sys^2+ SEQ	$h^{\text{LM-cut}}$
barman (20)	4	4	4	4	4	4	4	4	4	4
elevators (20)	7	9	16	16	4	17	16	15	16	**18**
floortile (20)	4	2	2	6	2	6	6	4	6	**7**
nomystery (20)	10	11	**16**	14	8	**16**	12	14	14	14
openstacks (20)	11	**14**	**14**	**14**	5	**14**	11	11	11	**14**
parcprinter (20)	**20**	11	13	13	7	14	**20**	**20**	**20**	13
parking (20)	3	**5**	1	2	1	1	2	1	1	3
pegsol (20)	**18**	17	17	17	10	17	**18**	17	16	17
scanalyzer (20)	11	9	4	11	7	10	10	10	8	**12**
sokoban (20)	16	19	**20**	**20**	13	**20**	**20**	**20**	19	**20**
tidybot (20)	7	13	**14**	**14**	4	**14**	10	8	10	**14**
transport (20)	**6**	**6**	**6**	**6**	4	**6**	**6**	5	**6**	**6**
visitall (20)	17	16	16	10	15	17	**19**	17	18	11
woodworking (20)	9	5	10	11	2	13	**16**	10	**16**	12
Sum IPC 2011 (280)	143	141	153	158	86	169	**170**	156	165	**165**
IPC 1998–2008 (1116)	487	446	478	586	357	589	**618**	516	598	**598**
Sum (1396)	630	587	631	744	443	758	**788**	672	763	**763**

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?