Sequencing Operator Counts with State-Space Search

Wesley L. Kaizer[1] André G. Pereira[1] Marcus Ritt[1]

[1]Federal University of Rio Grande do Sul, Brazil

Introduction

Introduction

- [Davies et al., 2015] decomposes the process of solving planning tasks into a **master problem** and a **subproblem**;

Introduction

- [Davies et al., 2015] decomposes the process of solving planning tasks into a **master problem** and a **subproblem**;
- The master problem solves an **operator-counting integer program**;

Introduction

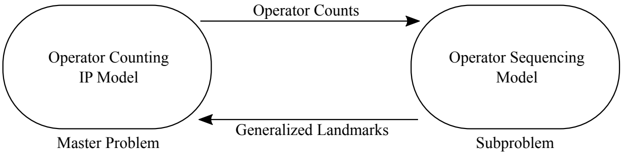
- [Davies et al., 2015] decomposes the process of solving planning tasks into a **master problem** and a **subproblem**;
- The master problem solves an **operator-counting integer program**;
- The subproblem tries to sequence the **operator counts**;

Introduction

- [Davies et al., 2015] decomposes the process of solving planning tasks into a **master problem** and a **subproblem**;
- The master problem solves an **operator-counting integer program**;
- The subproblem tries to sequence the **operator counts**;
- **Primal solution**: contains more information than the objective function value.

Introduction

- [Davies et al., 2015] decomposes the process of solving planning tasks into a **master problem** and a **subproblem**;
- The master problem solves an **operator-counting integer program**;
- The subproblem tries to sequence the **operator counts**;
- **Primal solution**: contains more information than the objective function value.



OpSeq and SAT

- *OpSeq* solves the subproblem using a **SAT solver**;

OpSeq and SAT

- *OpSeq* solves the subproblem using a **SAT solver**;
- Encodes the planning task and the operator counts in a SAT formula;

OpSeq and SAT

- *OpSeq* solves the subproblem using a **SAT solver**;
- Encodes the planning task and the operator counts in a SAT formula;
- If the formula is satisfiable, *OpSeq* can directly extract a plan;

OpSeq and SAT

- *OpSeq* solves the subproblem using a **SAT solver**;
- Encodes the planning task and the operator counts in a SAT formula;
- If the formula is satisfiable, *OpSeq* can directly extract a plan;
- Otherwise, *OpSeq* uses **assumptions** to generate a constraint.

OpSearch

OpSearch

- Heuristic search is the most common approach to solve classical planning tasks optimally;

OpSearch

- Heuristic search is the most common approach to solve classical planning tasks optimally;
- OpSearch is a new algorithm based on heuristic search to solve the operator counts sequencing subproblem;

OpSearch

- Heuristic search is the most common approach to solve classical planning tasks optimally;
- OpSearch is a new algorithm based on heuristic search to solve the operator counts sequencing subproblem;
- It uses information from the search graph, such as the f-values;

OpSearch

- Heuristic search is the most common approach to solve classical planning tasks optimally;
- OpSearch is a new algorithm based on heuristic search to solve the operator counts sequencing subproblem;
- It uses information from the search graph, such as the f-values;
- This approach generates smaller and more informed constraints;

OpSearch

- Heuristic search is the most common approach to solve classical planning tasks optimally;
- OpSearch is a new algorithm based on heuristic search to solve the operator counts sequencing subproblem;
- It uses information from the search graph, such as the f-values;
- This approach generates smaller and more informed constraints;
- Improves from advancements in planning research.

Constraint Generation Rule

Constraint Generation Rule

$$L = \{\, [Y_o \geq \mathcal{C}(o) + 1] \mid \exists s \xrightarrow{o} s' : f(s') \leq f_{\max} \wedge \\ ((v_o \notin vars(s) \wedge c(o) > 0) \vee s(v_o) = 0)\}.$$

Constraint Generation Rule

$$L = \{\, [Y_o \geq \mathcal{C}(o) + 1] \mid \exists s \xrightarrow{o} s' : f(s') \leq f_{\max} \wedge ((v_o \notin vars(s) \wedge c(o) > 0) \vee s(v_o) = 0)\}.$$

Constraint Generation Rule

$$L = \{\, [Y_o \geq \mathcal{C}(o) + 1] \mid \exists s \xrightarrow{o} s' : f(s') \leq f_{\max} \wedge \\ ((v_o \notin vars(s) \wedge c(o) > 0) \vee s(v_o) = 0)\}.$$

Constraint Generation Rule

$$L = \{\, [Y_o \geq \mathcal{C}(o) + 1] \mid \exists s \xrightarrow{o} s' : f(s') \leq f_{\max} \wedge \\ ((v_o \notin vars(s) \wedge c(o) > 0) \vee s(v_o) = 0)\}.$$

Constraint Generation Rule

$$L = \{\, [Y_o \geq \mathcal{C}(o) + 1] \mid \exists s \xrightarrow{o} s' : f(s') \leq f_{\max} \land \\ ((v_o \notin vars(s) \land c(o) > 0) \lor s(v_o) = 0)\}.$$

Constraint Generation Rule

$$L = \{\, [Y_o \geq \mathcal{C}(o) + 1] \mid \exists s \xrightarrow{o} s' : f(s') \leq f_{\max} \wedge$$
$$((v_o \notin vars(s) \wedge c(o) > 0) \vee s(v_o) = 0)\}.$$

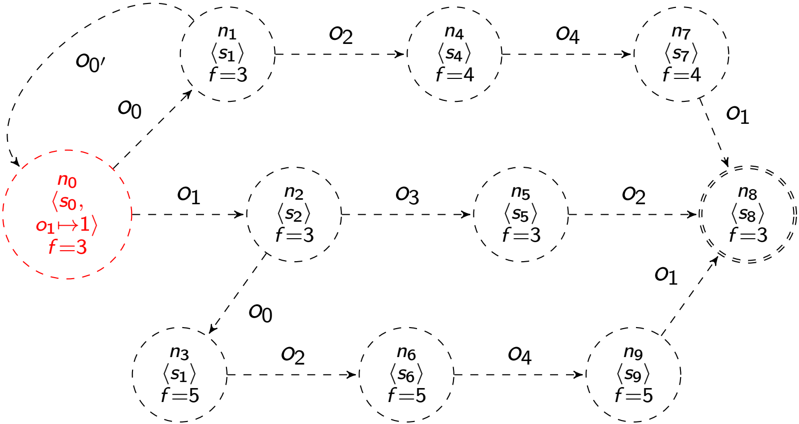
Constraint Generation Strategy Example

Example: First Iteration

$\mathcal{C} = \{o_1 \mapsto 1\}$ and $f_{\max} = 1$:

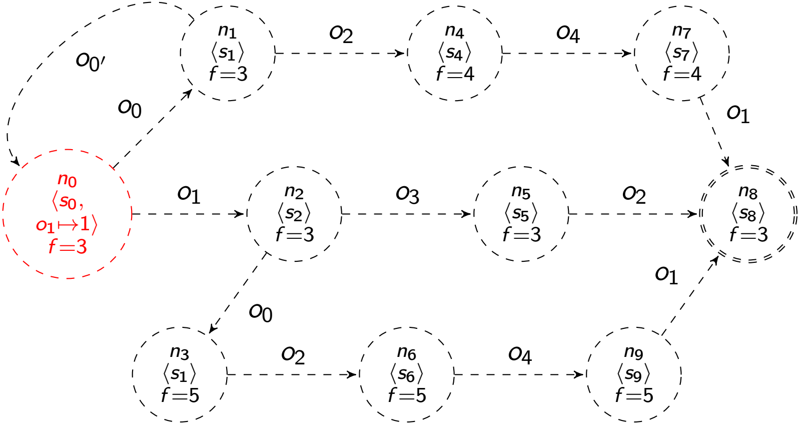
Example: First Iteration

$\mathcal{C} = \{o_1 \mapsto 1\}$ and $f_{\max} = 1$:



Example: First Iteration

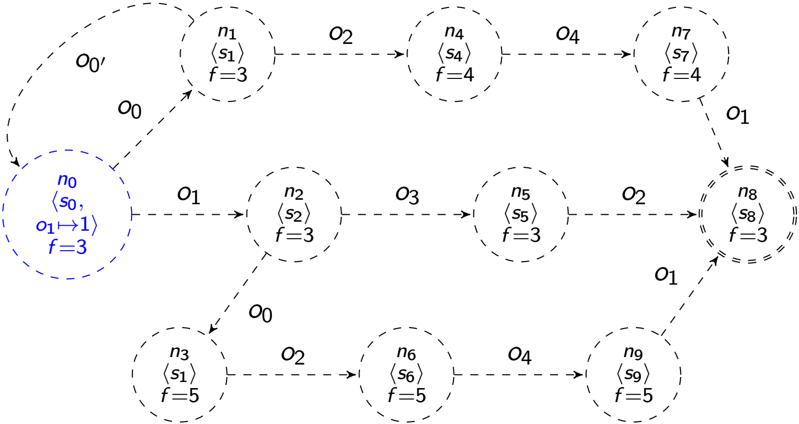
$\mathcal{C} = \{o_1 \mapsto 1\}$ and $f_{\max} = 1$:



GLC: $[Y_f \geq 3] \geq 1$.

Example: First Iteration

$\mathcal{C} = \{o_1 \mapsto 1\}$ and $f_{\max} = 1$:



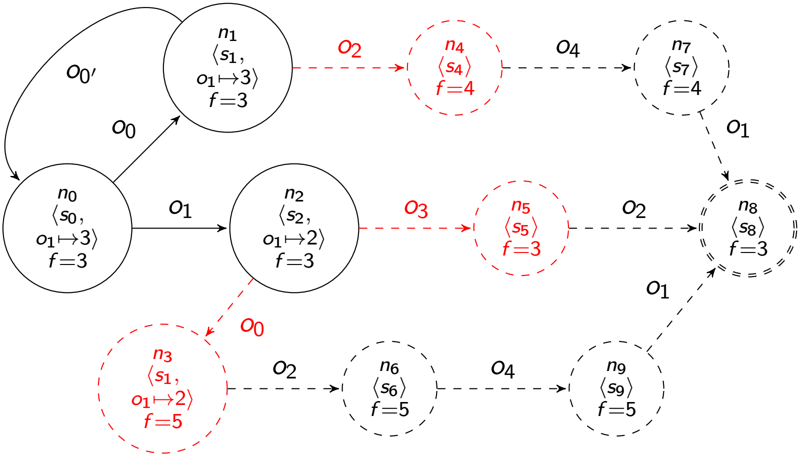
GLC: $[Y_f \geq 3] \geq 1$.

Example: Second Iteration

$\mathcal{C} = \{o_1 \mapsto 3\}$ and $f_{\max} = 3$:

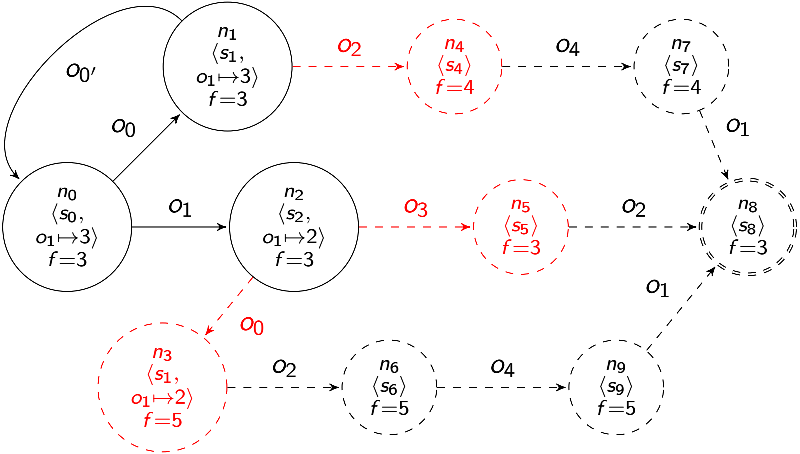
Example: Second Iteration

$\mathcal{C} = \{o_1 \mapsto 3\}$ and $f_{\max} = 3$:



Example: Second Iteration

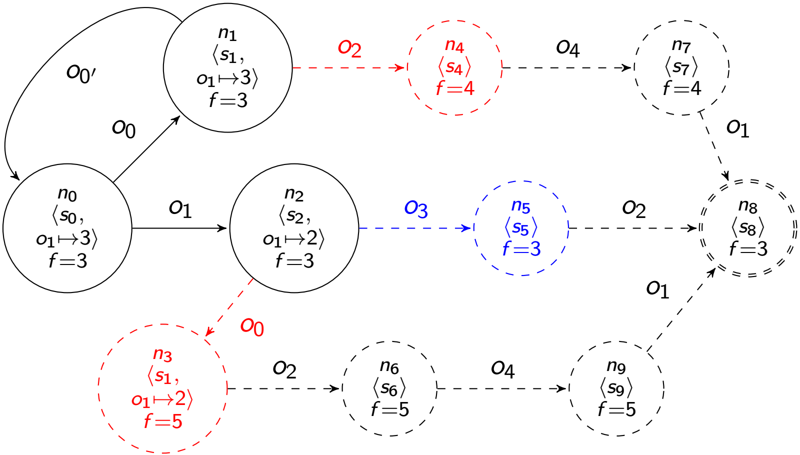
$\mathcal{C} = \{o_1 \mapsto 3\}$ and $f_{\max} = 3$:



GLC: $[\mathsf{Y}_{o_3} \geq 1] + [\mathsf{Y}_f \geq 4] \geq 1.$

Example: Second Iteration

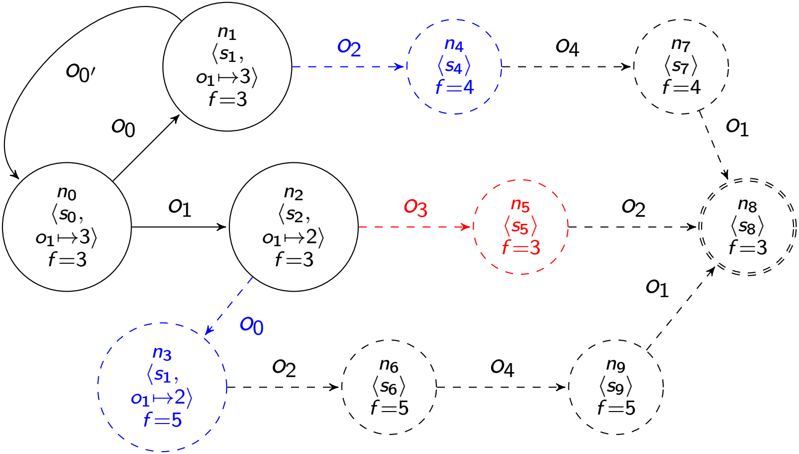
$\mathcal{C} = \{o_1 \mapsto 3\}$ and $f_{\max} = 3$:



GLC: $[\mathrm{Y}_{o_3} \geq 1] + [\mathrm{Y}_f \geq 4] \geq 1$.

Example: Second Iteration

$\mathcal{C} = \{o_1 \mapsto 3\}$ and $f_{\max} = 3$:



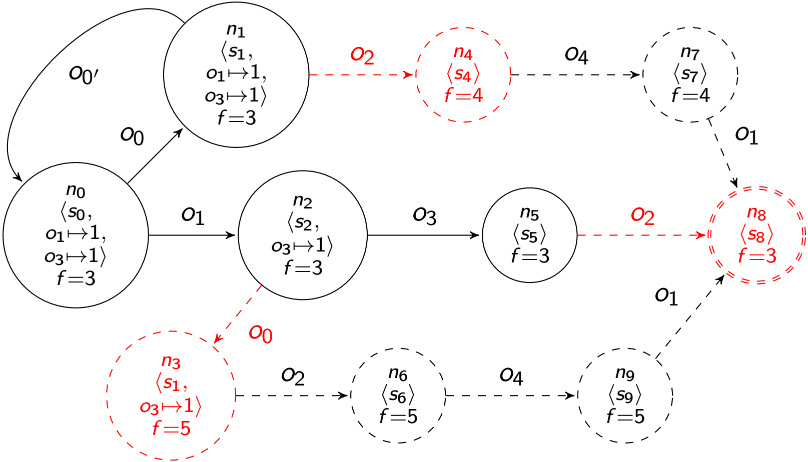
GLC: $[\mathrm{Y}_{o_3} \geq 1] + [\mathrm{Y}_f \geq 4] \geq 1$.

Example: Third Iteration

$\mathcal{C} = \{o_1 \mapsto 2, o_3 \mapsto 1\}$ and $f_{\max} = 3$:

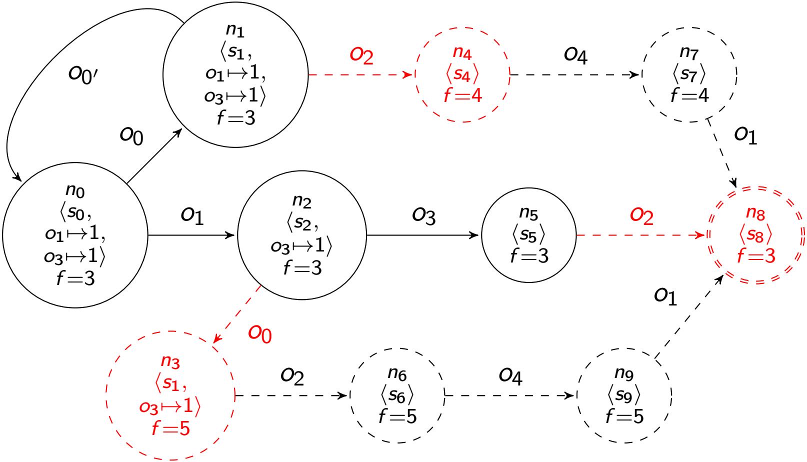
Example: Third Iteration

$\mathcal{C} = \{o_1 \mapsto 2, o_3 \mapsto 1\}$ and $f_{\max} = 3$:



Example: Third Iteration

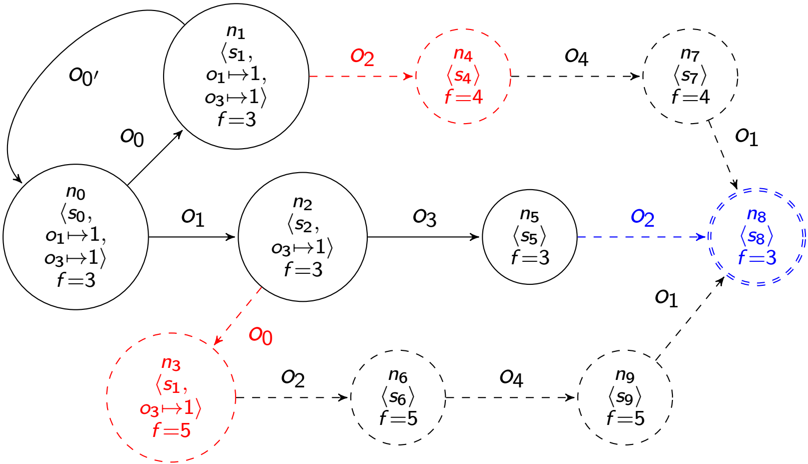
$\mathcal{C} = \{o_1 \mapsto 2, o_3 \mapsto 1\}$ and $f_{\max} = 3$:



GLC: $[\mathrm{Y}_{o_2} \geq 1] + [\mathrm{Y}_f \geq 4] \geq 1$.

Example: Third Iteration

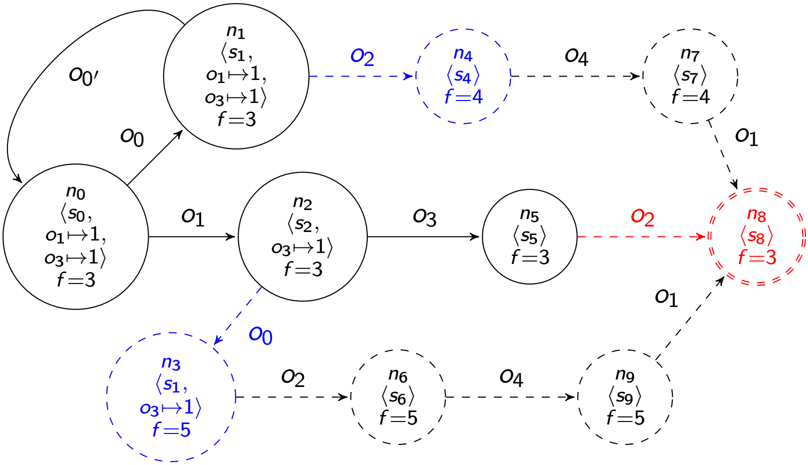
$\mathcal{C} = \{o_1 \mapsto 2, o_3 \mapsto 1\}$ and $f_{\max} = 3$:



GLC: $[\mathsf{Y}_{o_2} \geq 1] + [\mathsf{Y}_f \geq 4] \geq 1.$

Example: Third Iteration

$\mathcal{C} = \{o_1 \mapsto 2, o_3 \mapsto 1\}$ and $f_{\max} = 3$:



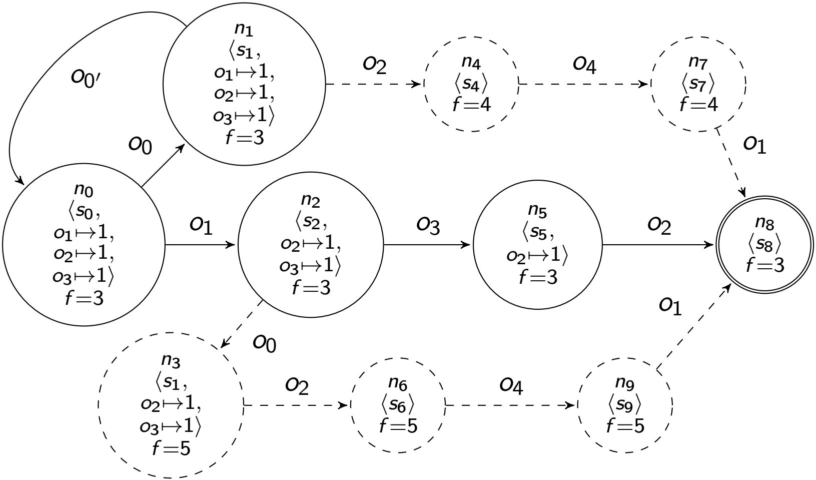
GLC: $[\mathrm{Y}_{o_2} \geq 1] + [\mathrm{Y}_f \geq 4] \geq 1$.

Example: Fourth Iteration

$\mathcal{C} = \{o_1 \mapsto 1, o_2 \mapsto 1, o_3 \mapsto 1\}$ and $f_{\mathsf{max}} = 3$:

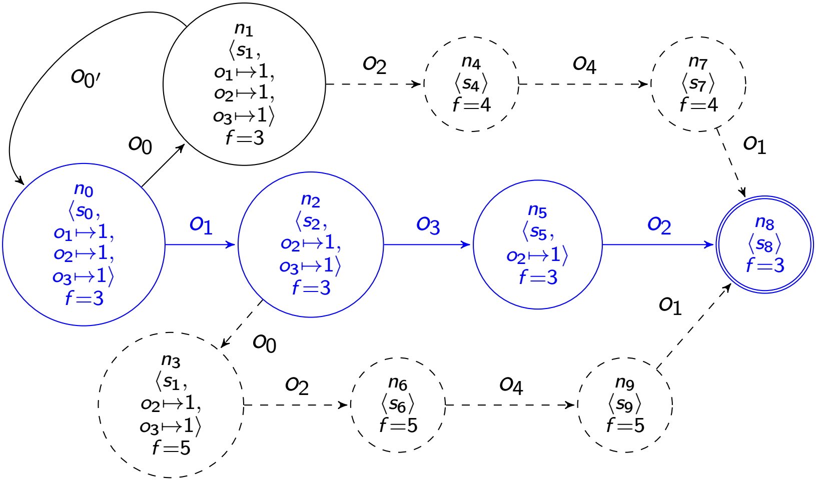
Example: Fourth Iteration

$\mathcal{C} = \{o_1 \mapsto 1, o_2 \mapsto 1, o_3 \mapsto 1\}$ and $f_{\max} = 3$:



Example: Fourth Iteration

$\mathcal{C} = \{o_1 \mapsto 1, o_2 \mapsto 1, o_3 \mapsto 1\}$ and $f_{\max} = 3$:



Plan: $\langle o_1, o_3, o_2 \rangle$.

Theorem

Theorem

Theorem 1. *For a solvable SAS^+ planning task Π, an operator counts $\mathcal{C}_s$ with an associated f-bound value f_{max}, such that OpSearch's modified A^* with an admissible heuristic function h cannot sequence $\mathcal{C}_s$, OpSearch always returns an admissible constraint to the master integer program.*

Results

OpSearch is Better than OpSeq

IPC 2011 - All Instances

	OpSeq	*OpSearch*
C	63	**73**

IPC 2011 - All Instances

	OpSeq	*OpSearch*
C	63	**73**
S	121202	**99437**

IPC 2011 - All Instances

	OpSeq	*OpSearch*
C	63	**73**
S	121202	**99437**
$\bar{T}_t$	1783	**1720**

IPC 2011 - All Instances

	OpSeq	*OpSearch*
C	63	**73**
S	121202	**99437**
$\bar{T}_t$	1783	**1720**
$\bar{M}$	865	**367**

IPC 2011 - All Instances

	OpSeq	*OpSearch*
C	63	**73**
S	121202	**99437**
$\bar{T}_t$	1783	**1720**
$\bar{M}$	865	**367**
$\bar{u}$	20	**6**

IPC 2011 - All Instances

	OpSeq	*OpSearch*
C	63	**73**
S	121202	**99437**
$\bar{T}_t$	1783	**1720**
$\bar{M}$	865	**367**
$\bar{u}$	20	**6**

OpSearch is better than *OpSeq*: **solves more tasks, solves less subproblems, uses less memory and generate smaller constraints**.

IPC 2011 - 50 Solved by Both

	OpSeq	*OpSearch*
S	2738	**2169**

IPC 2011 - 50 Solved by Both

	OpSeq	*OpSearch*
S	2738	**2169**
$\bar{T}_t$	**92**	191

IPC 2011 - 50 Solved by Both

	OpSeq	*OpSearch*
S	2738	**2169**
$\bar{T}_t$	**92**	191
$\bar{M}$	122	**118**

IPC 2011 - 50 Solved by Both

	OpSeq	*OpSearch*
S	2738	**2169**
$\bar{T}_t$	**92**	191
$\bar{M}$	122	**118**
$\bar{u}$	15	**9**

IPC 2011 - 50 Solved by Both

	OpSeq	*OpSearch*
S	2738	**2169**
$\bar{T}_t$	**92**	191
$\bar{M}$	122	**118**
$\bar{u}$	15	**9**

OpSearch is better than *OpSeq*: **solves more tasks, solves less subproblems, uses less memory and generate smaller constraints**.

OpSearch Improves with Better Heuristics

IPC 1998-2014 - 154 Instances

	OpSeq	h^{blind}	h^{LMCut}	h^{OC}	h^*
S	29106	25059	13304	7215	**3214**

IPC 1998-2014 - 154 Instances

	OpSeq	h^{blind}	h^{LMCut}	h^{OC}	h^*
S	29106	25059	13304	7215	**3214**
$\bar{T}_t$	37	**10**	11	39	13

IPC 1998-2014 - 154 Instances

	OpSeq	h^{blind}	h^{LMCut}	h^{OC}	h^*
S	29106	25059	13304	7215	**3214**
$\bar{T}_t$	37	**10**	11	39	13
$\bar{M}$	95	82	82	**81**	234

IPC 1998-2014 - 154 Instances

	OpSeq	h^{blind}	h^{LMCut}	h^{OC}	h^*
S	29106	25059	13304	7215	**3214**
$\bar{T}_t$	37	**10**	11	39	13
$\bar{M}$	95	82	82	**81**	234
$\bar{u}$	18	18	11	10	**8**

IPC 1998-2014 - 154 Instances

	OpSeq	h^{blind}	h^{LMCut}	h^{OC}	h^*
S	29106	25059	13304	7215	**3214**
$\bar{T}_t$	37	**10**	11	39	13
$\bar{M}$	95	82	82	**81**	234
$\bar{u}$	18	18	11	10	**8**
C	169	191	195	200	**241**

IPC 1998-2014 - 154 Instances

	OpSeq	h^{blind}	h^{LMCut}	h^{OC}	h^*
S	29106	25059	13304	7215	**3214**
$\bar{T}_t$	37	**10**	11	39	13
$\bar{M}$	95	82	82	**81**	234
$\bar{u}$	18	18	11	10	**8**
C	169	191	195	200	**241**

As a more informed heuristic is used by *OpSearch*, **the number of subproblems solved, the memory usage and the size of the generated constraints decrease and the number of solved tasks increases**.

Applications

Applications

- Our research results are relevant in practical applications besides the further development of automated planning;

Applications

- Our research results are relevant in practical applications besides the further development of automated planning;
- *OpSearch* can be used as an anytime method to obtain lower-bounds on plan costs;

Applications

- Our research results are relevant in practical applications besides the further development of automated planning;
- *OpSearch* can be used as an anytime method to obtain lower-bounds on plan costs;
- Also in *agile planning* to solve planning tasks for which informative heuristics are already known;

Applications

- Our research results are relevant in practical applications besides the further development of automated planning;
- *OpSearch* can be used as an anytime method to obtain lower-bounds on plan costs;
- Also in *agile planning* to solve planning tasks for which informative heuristics are already known;
- Another practical application of our approach is for *diverse planning*, used for example by IBM, that aims to find several plans while guaranteeing diversity.

Take Home Messages

Take Home Messages

- The operator counts sequencing problem can be efficiently solved using heuristic search;

Take Home Messages

- The operator counts sequencing problem can be efficiently solved using heuristic search;
- Our approach opens new research directions towards specialized methods or heuristics to this problem;

Take Home Messages

- The operator counts sequencing problem can be efficiently solved using heuristic search;
- Our approach opens new research directions towards specialized methods or heuristics to this problem;
- It is a novel research problem with great potential of development in both areas of *operations research* and *artificial intelligence*.

Thanks!

Wesley Luciano Kaizer
kaizerwesley@gmail.com

References I

Davies, T. O., Pearce, A. R., Stuckey, P. J., and Lipovetzky, N. (2015).
Sequencing operator counts.
In *International Conference on Automated Planning and Scheduling*, pages 61–69.