Initial Results on Generating Macro Actions from a Plan Database for Planning on Autonomous Mobile Robots

Till Hofmann, Tim Niemueller, Gerhard Lakemeyer
Knowledge-Based Systems Group
RWTH Aachen University

At A Glance

Task Planning for Autonomous Robots

- Robotic systems plan and re-plan frequently

→ Planning needs to be especially fast

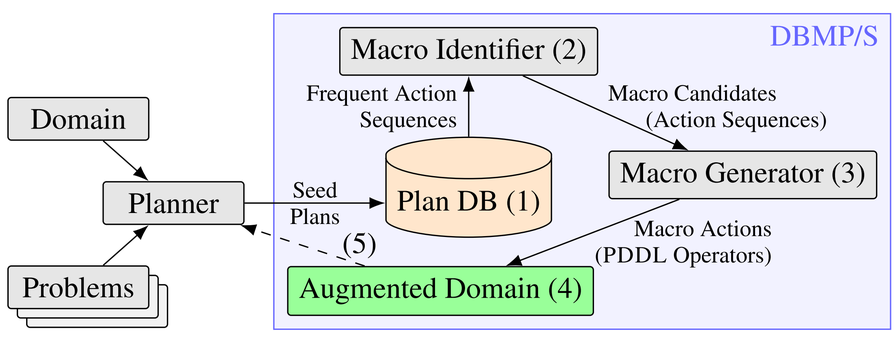
- In typical scenarios, robots perform similar or repeating tasks

→ Exploit repetitiveness by re-using previous planning solutions

Database-Driven Macro Planning for STRIPS (DBMP/S)

- Improve planner performance with additional *macro actions*
- A macro is an action consisting of several STRIPS actions
- Macros are generated from frequent action sequences

Architecture Overview



1. Collect planning results in a *plan database*
2. *Identify* frequent action sequences
3. *Generate* macros for those action sequences
4. Add macros to the domain to get an *augmented domain*
5. Use augmented domain for new planning problems

Plan Database

- Store domains, problems, plans, and macros in document-oriented database MongoDB
- Store data in a structured way in order to allow analysis
- MongoDB allows horizontal scaling and MapReduce natively

```
{
  "_id" : ObjectId("582c9443babe8d00010f4209"),
  "actions" : [
    { "operator" : "unstack",
      "parameters" : [ "b16", "b17" ]
    },
    { "operator" : "put-down",
      "parameters" : [ "b16" ]
    }, ... ],
  "domain" : "blocksworld",
  "problem" : "BW-rand-20-75",
  "raw" : "(UNSTACK B16 B17)\n(PUT-DOWN B16),...",
  "resources" : [ 0.0145 ]
}
```

Identification

Identify frequent action sequences with MapReduce

Map: Emit all action sub-sequences in all plans including all possible parameter assignments

Reduce: Count occurring action sequences

Result: A set of ungrounded frequent action sequences

Generation

Generate precondition and effects from an action sequence

Given an action sequence $\langle a_1, \ldots, a_n \rangle$ with preconditions $\pi_1, \ldots, \pi_n$ and effects $e_1, \ldots, e_n$:

1. Compute the macro precondition by *regressing* each precondition π_i over the effects $e_{i-1} \ldots, e_1$
2. Compute the macro effect by *chaining* the effects $e_1, \ldots, e_n$

Regressing a precondition π over effect e

1. If π is a conjunction $\pi = \bigwedge_j \pi_j$, we regress all sub-formulae π_j to π'_j and obtain $\pi' = \bigwedge_j \pi'_j$, analogously for disjunctions
2. If $\pi = P(\vec{s})$ and $e = P(\vec{t})$ for some predicate P, then we regress π to $\pi' = (\vec{s} = \vec{t} \vee P(\vec{s}))$
3. Similarly, if $\pi = \neg P(\vec{s})$ and $e = P(\vec{t})$ for some predicate P, we regress π to $\pi' = (\vec{s} \neq \vec{t} \wedge \neg P(\vec{s}))$
4. If $\pi = (\neg) P(\vec{s})$ and $e = Q(\vec{t})$ for some distinct predicates P and Q, the effect is unrelated to π and we get $\pi' = \pi$. For a negated effect $e = \neg P(\vec{t})$, we proceed analogously
5. Finally, if e is a conjunction $e = \bigwedge_j e_j$, we regress π on each e_j successively

Macro effect generation

- Merge all effects $\sigma = \langle e_1, \ldots, e_n \rangle$
- Start with $e = e_n$, i.e., the last action's effect
- Add each effect e_{i-1} consecutively with *effect chaining*
- After chaining $e_n, \ldots, e_1$, we obtain the macro effect e

Effect Chaining of two effects e and f

1. If e is a conjunction $e = \bigwedge_i e_i$, we chain each e_i with f to e'_i and obtain $e' = \bigwedge_i e'_i$
2. If $e = f$, we remove e
3. If $e = \neg f$, we remove e
4. If f is a conjunction $f = \bigwedge_i f_i$, we chain e with each f_i consecutively to e'_i and obtain $e' = \bigwedge_i e'_i$
5. If $e = P(\vec{s})$ and $f = Q(\vec{t})$ for some distinct predicates P and Q, we keep e, i.e., $e' = e$

DBMP/S Features

- DBMP/S collects plans to speed up subsequent planning tasks
- Previous results are used to identify frequent action sequences
- Frequent action sequences are merged into macro actions
- Macro actions are represented as normal PDDL actions

⇒ No adaption of the underlying planner necessary

- Common parameters are coalesced

⇒ Reduce the total number of parameters in resulting macros

- Macros are generated off-line

⇒ Shorter on-line planning time

Example: Blocksworld Macros

Actions

```
(:action unstack
 :parameters (?x - block ?y - block)
 :precondition (and (on ?x ?y) (clear ?x) (handempty))
 :effect (and (holding ?x) (clear ?y) (not (clear ?x))
          (not (handempty)) (not (on ?x ?y))))
(:action put-down
 :parameters (?b - block)
 :precondition (holding ?b)
 :effect (and (not (holding ?b)) (clear ?b)
          (handempty) (ontable ?b)))
```

Identification

```
[ { "problem": "p1", "actions" : [
    { "operator" : "unstack",
      "parameters" : [ "b19", "b1" ] },
    { "operator" : "put-down",
      "parameters" : [ "b19" ]
    }
  ]},
  { "problem": "p2", "actions" : [
    { "operator" : "unstack",
      "parameters" : [ "b12", "b3" ] },
    { "operator" : "put-down",
      "parameters" : [ "b12" ] }
  ]}
]
```

- Action sequence: `<unstack,put-down>`
- Assignment for `unstack`: `?x` → `?p1`, `?y` → `?p2`
- Assignment for `put-down`: `?b` → `?p1`

⇒ Parameters `?x` and `?b` are coalesced to reduce final parameters

Generated Macro

```
; MACRO unstack-put-down
; ACTIONS [unstack,put-down] PARAMETERS [[1,2],[1]]
(:action unstack-put-down
 :parameters (?p1 ?p2 - block)
 :precondition (and (on ?p1 ?p2) (clear ?p1)
                (handempty))
 :effect (and (not (holding ?p1)) (clear ?p1)
          (handempty) (ontable ?p1)
          (clear ?p2) (not (on ?p1 ?p2))))
```

- Precondition of `put-down` is regressed on `unstack`

→ `(holding ?p1)` can be simplified to `(true)`

- Effect of `unstack` is chained with effect of `put-down`

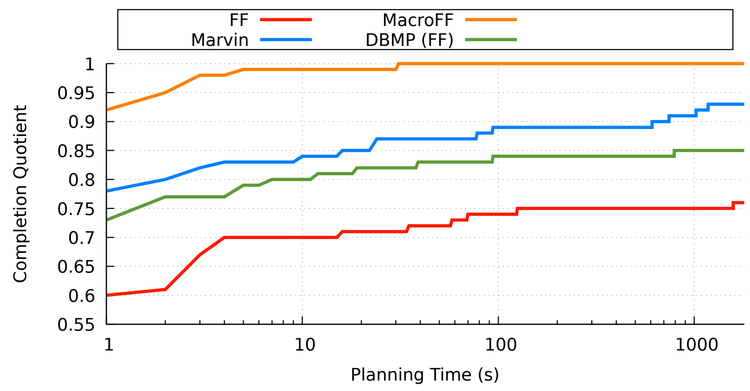
→ effect `(not (handempty))` is removed from macro effect

- The macro's actions and parameter assignment are stored as a comment to translate the macro to the original actions

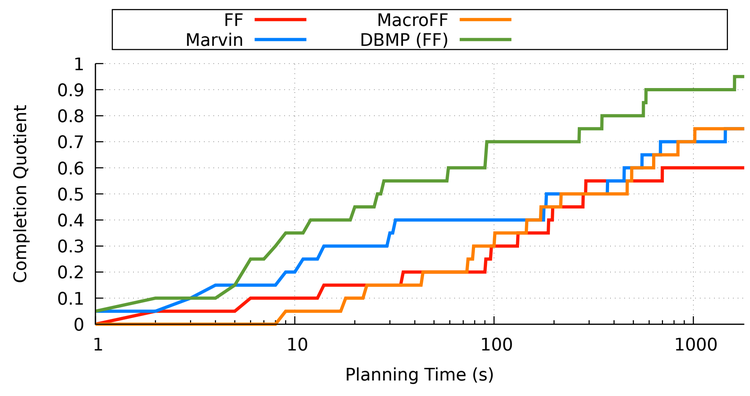
Benchmark Results

	FF	MacroFF	Marvin	DBMP/S (FF)	DBMP/S (Marvin)
Blocksworld (100 problems)					
# solved	76	**100**	93	85	93
mean (s)	25.2	**0.72**	41.1	11.5	33.0
Q1 (s)	**0.006**	0.16	0.12	**0.006**	0.15
Q2 (s)	0.12	0.23	0.23	**0.017**	0.25
Q3 (s)	124	**0.40**	0.64	1.57	0.94
Hiking (20 problems)					
# solved	12	15	15	**19**	15
mean (s)	260	349	264	**196**	284
Q1 (s)	91.0	86.6	10.9	**5.67**	11.8
Q2 (s)	280	339	182	**25.6**	273
Q3 (s)	×	1751	1447	**268**	1653
Barman (20 problems)					
# solved	0	0	6	19	**20**
mean (s)	×	×	125	**4.60**	147
Q1 (s)	×	×	256	**2.10**	6.76
Q2 (s)	×	×	×	**2.96**	17.3
Q3 (s)	×	×	×	**5.77**	35.4
Logistics Robots (81 problems)					
# solved	4	†	3	4	3
mean (s)	0.156	×	0.74	**0.094**	0.40

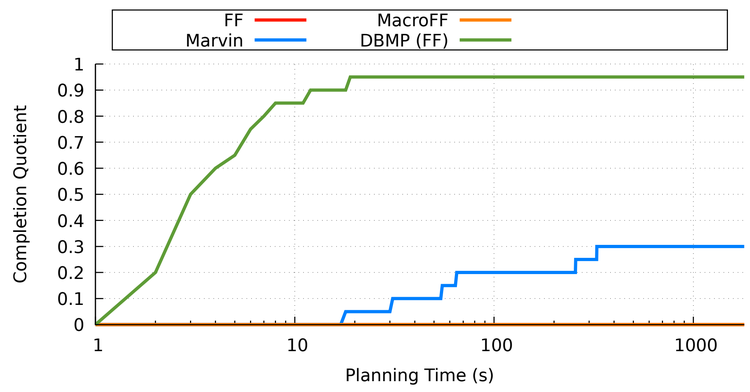
Blocksworld



Hiking



Barman



Acknowledgments

- DFG Research Unit FOR 1513 on Hybrid Reasoning for Intelligent Systems
- DFG Grant GL-747/23-1
- Festo Didactic SE (travel funding)