Learning for Classical Planning

Mgr. Lukáš Chrpa, Ph.D.

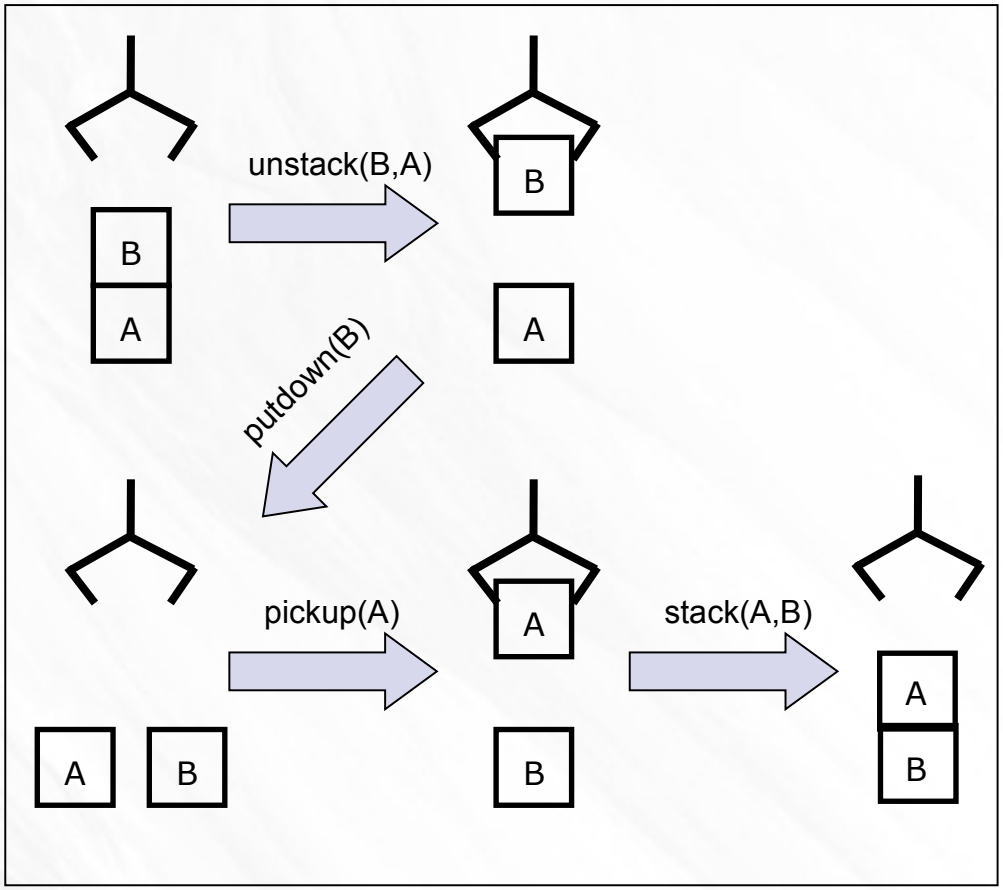
Seminar AI 11.3.2010

Outline

- Introduction
  - Classical Planning
  - Motivation for learning
- Contributions
  - Analysis of action dependencies and independencies in plans
  - Generation of Macro-operators
  - Eliminating unnecessary operators' instances
  - Putting together
- Conclusions & Future research plans

Classical Planning

- Given
  - Deterministic, fully observable and static environment
  - Actions with defined preconditions and effects (or operators whose instances are actions)
  - Initial state
  - Goal predicates
- We want
  - find a sequence of actions transforming the state from the initial state to the goal state = **plan**



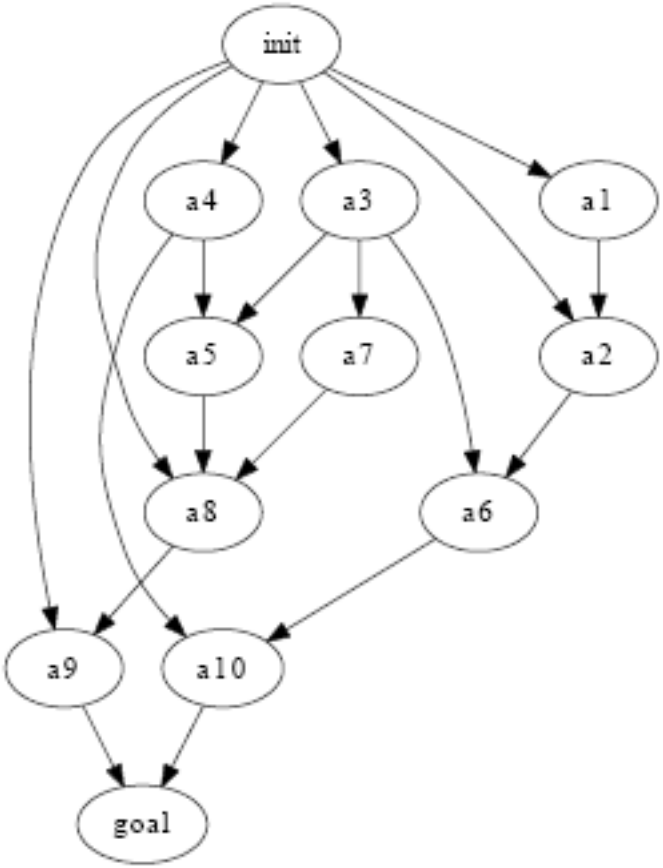
Motivation for Learning

- Huge complexity (up to EXPSPACE-complete – considering classical representation)
- Planners do not care about plan structures of previously generated plans
- Plan analysis may reveal useful knowledge (for instance, Macro-Operators, connections between operators' predicates and initial or goal predicates)
- Knowledge can be passed directly to domains allowing a usage of the best existing planners

Dependencies between actions

- Straight dependency
  - Earlier action provided some predicate(s) to the given action (the last one)
- Dependency
  - Transitive closure of straight dependency
- Independency
  - Not dependent, the later action does not delete precondition predicates the earlier one, the earlier action does not delete positive effects the later one
  - Allows the adjacent actions to swap
  - **Not necessarily complementary to the dependency relation**

Action dependencies - example



a1: LIFT(hoist0,crate1,pallet0,depot0)

a2: LOAD(hoist0,crate1,truck1,depot0)

a3: DRIVE(truck1,depot0,distributor0)

a4: LIFT(hoist1,crate0,pallet1,distributor0)

a5: LOAD(hoist1,crate0,truck1,distributor0)

a6: UNLOAD(hoist1,crate1,truck1,distributor0)

a7: DRIVE(truck1,distributor0,distributor1)

a8: UNLOAD(hoist2,crate0,truck1,distributor1)

a9: DROP(hoist2,crate0,pallet2,distributor1)

a10: DROP(hoist1,crate1,pallet1,distributor0)

Usage of action dependencies

- Decomposition to subplans and subproblems
- Plan Optimization (detecting and removing actions not necessary to reach goal, inverse actions)
- **Detection and generation of Macro-operators**

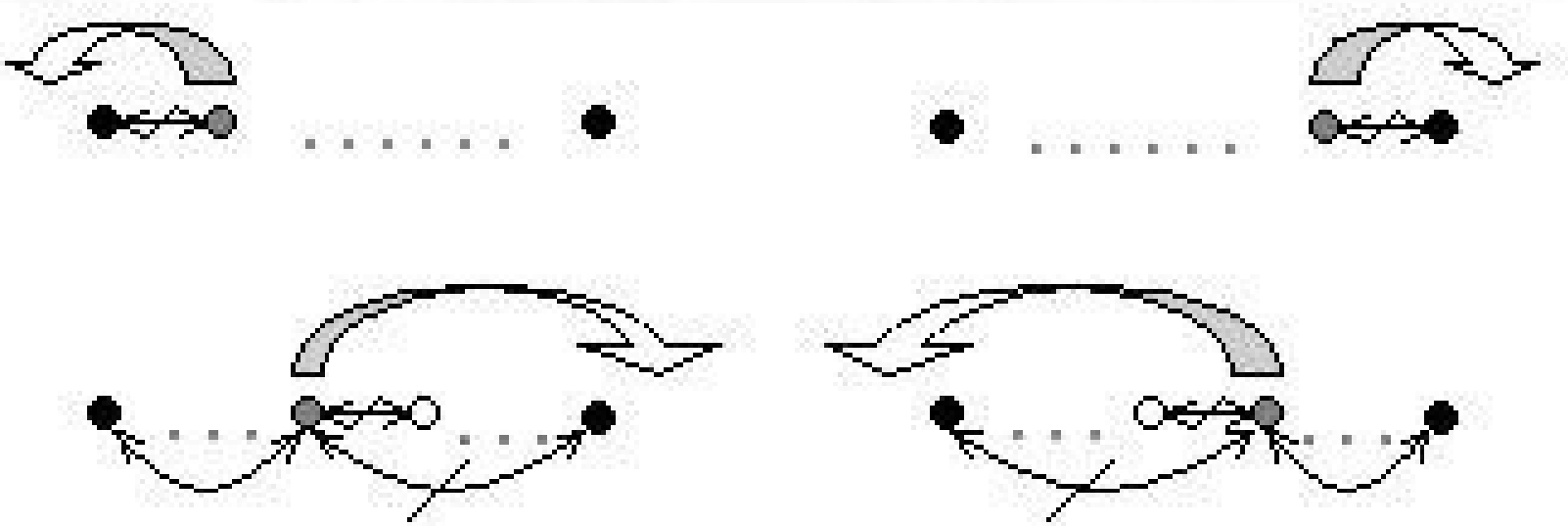
Macro-actions and Macro-operators

- Primitive actions can be assembled into one single action – macro-action
- Assemblage of actions a_i and a_j into $a_{i,j}$:
  - $p(a_{i,j}) = p(a_i) \cup (p(a_j) - e^+(a_i))$
  - $e^-(a_{i,j}) = (e^-(a_i) \cup e^-(a_j)) - e^+(a_j)$
  - $e^+(a_{i,j}) = (e^+(a_i) \cup e^+(a_j)) - e^-(a_j)$
- Macro-actions are Macro-operators' instances

Identifying actions that can be assembled – the main idea

- Adjacent actions can be easily assembled without loss of plans validity
- Intermediate actions can be moved before (respectively behind) selected actions to make them adjacent
- Recall that independent adjacent actions can be swapped

Identifying actions that can be assembled – algorithm sketch



Four different situations for moving the intermediate actions (grey-filled) before or after one of the boundary actions (black-filled).

Matrix of Candidates for becoming Macro-operators

	LIFT(H,C,S,P)	LOAD(H,C,T,P)	DRIVE(T,P1,P2)	UNLOAD(H,C,T,P)	DROP(H,C,S,P)
LIFT(H,C,S,P) / 5		**5** (H,C,P)			
LOAD(H,C,T,P) / 5	**1** (H,P)			**2** (H,T,P)	
DRIVE(T,P1,P2) / 5		**2** (T,P2)		**3** (T,P2)	
UNLOAD(H,C,T,P) / 5					**5** (H,C,P)
DROP(H,C,S,P) / 5					

Matrix contains a number of pairs of instances of particual operators (including shared arguments) that can be assembled in training plans

Inequality constraints

- Sometimes macro-actions, where some assigned arguments equal, are not valid (cannot be unfolded)
- Can be detected by simulating performance of primitive actions

```
(:action pickup_stack
   :parameters (?x ?y)
   :precondition (and (clear ?x)(ontable ?x)(handempty)(clear ?y))
   :effect (and (clear ?x)(on ?x ?y)(handempty)
         (not (ontable ?x))(not (holding ?x))(not (clear ?y)) )
 )
```

Generation of Macro-operators

- **Repeat**
  - Create-Matrix
  - **If** Select-Candidate **then**
    - Assign Inequality Constraints
    - Create-Macro-Operator
    - Update-Plans
  - **EndIf**
- **Until** not candidate-selected

Selection criteria

- Operators whose instances usually appear (or can appear) consecutively
- Number of assemblages regarding the total training plans length is not too low
- Number of arguments of generated macro-operators is not too high

Experimental evaluation – an approach

- Generate training plans by SGPLAN and SATPLAN
- Generate Macro-operators, remove replaced primitive operators and reformulate the domains
- Run the planners (SATPLAN, SGPLAN, LAMA) both on the original domains and on the reformulated domains and compare results.

Experimental results (fragment)

SGPlan						
	time (in sec)			plan length		
	orig	upd-SG	upd-SAT	orig	upd-SG	upd-SAT
Blocks14-0	>600	0,03	0,03	NA	48	48
Blocks14-1	>600	0,03	0,03	NA	44	44
Blocks15-0	>600	0,32	0,32	NA	88	88
Blocks15-1	179,84	0,05	0,05	114	54	54
depots1817	24,56	15,52	20,71	100	104	94
depots4534	>600	0,53	54,71	NA	112	110
depots5656	410,94	0,32	7,70	133	132	82
depots7615	8,48	1,88	2,14	98	102	91

SATPLAN						
	time (in sec)			plan length		
	orig	upd-SAT	upd-SG	orig	upd-SAT	upd-SG
gripper8	>600	8,14	0,03	NA	53	71
gripper9	>600	12,86	0,06	NA	59	79
gripper10	>600	19,78	0,04	NA	65	87
gripper11	>600	err	0,07	NA	err	95
gripper12	>600	err	0,06	NA	err	103

Experimental results (fragment)

LAMA						
	time (in sec)			plan length		
	orig	upd-SG	upd-SAT	orig	upd-SG	upd-SAT
depots1817	>600	93,68	>600	NA	122	NA
depots4534	243,61	1,39	9,81	122	67	107
depots5656	>600	0,53	7,70	NA	70	98
depots7615	>600	5,71	61,61	NA	77	78
goldminer-7x7-06	0,22	0,04	0,03	170	31	31
goldminer-7x7-07	0,04	0,04	0,03	65	34	65
goldminer-7x7-08	>600	0,03	0,03	NA	25	26
goldminer-7x7-09	0,14	0,04	0,03	130	29	32
goldminer-7x7-10	0,30	0,04	0,03	176	31	43

Discussion

- Removing of primitive operators did not affected the solvability of the given problems
- Macro-operators were mostly combined from two primitive operators
- Running time were often better in updated domains, plan quality sometimes significantly better (surprisingly)
- Success of the method depends on kinds of domains and particular planners

Eliminating unnecessary actions

- Planners must consider many useless actions
- Existing techniques focus mainly on pruning unreachable actions
- In many cases there exists a connection between operators' instances and initial or goal predicates

Entangelement and full entanglement

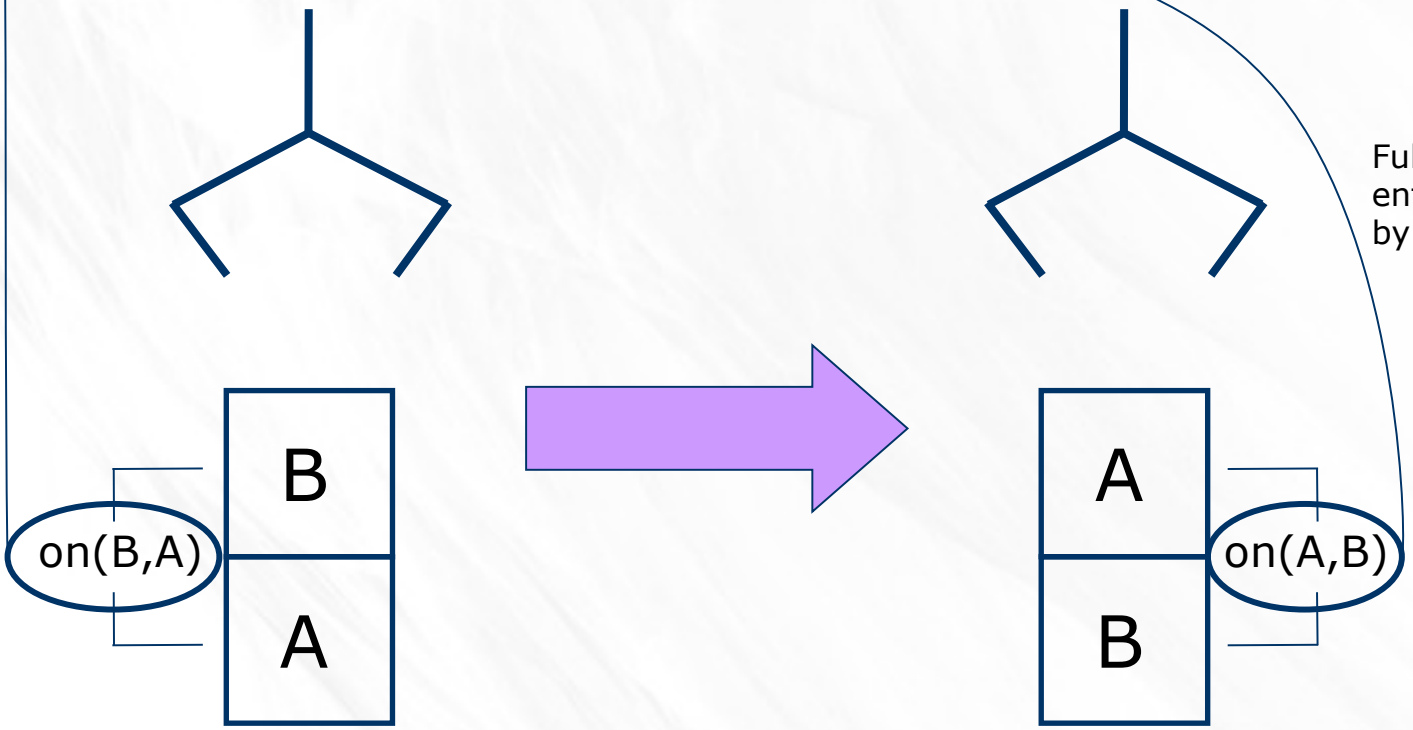
- An operator is entangled by init (goal) with a predicate in the given problem if there exists plan, where every instance of the operator is such that the corresponding instance of the predicate in the precondition (positive effects) is also present in the set of initial (goal) predicates
- Full entanglement extends entanglement for every solvable planning problem in the given domain

Unstack(X,Y) =
{ {on(X,Y),clear(X),handempty} //prec
{on(X,Y),clear(X),handempty} //neg eff
{holding(X),clear(Y)} } //pos eff

Stack(X,Y) =
{ {holding(X),clear(Y)} //prec
{holding(X),clear(Y)} //neg eff
{on(X,Y),clear(X),handempty} //pos eff
}

Fully entangled by init

Fully entangled by goal

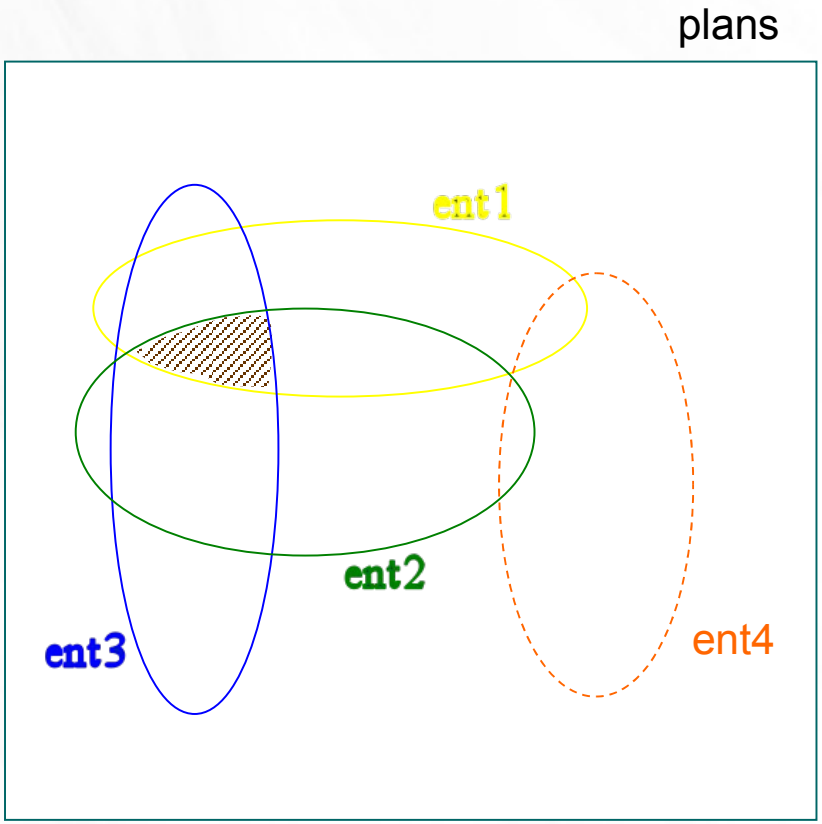


init

goal

Set of compatible full entanglements

- Every full entanglement somehow restricts the set of feasible plans (if non-static predicate involved)
- Full entanglements are compatible if for every solvable problems there remains (after all restrictions) at least one feasible plan (for instance – *ent1, ent2, ent3)*



Reformulating domains and problems

- Each static predicate is fully entangled by init
- If operator *o* is fully entangled by init (goal) with non-static predicate *p*, then
  - Create new static predicate *p'*
  - Add *p'* to the precondition of *o*
  - If an instance of *p* is in the initial state (goal predicates), then add the corresponding instance of *p'* to the initial state

```
(:action unstack
  :parameters  (?x ?y - block)
  :precondition (and (on ?x ?y) (clear ?x) (emptyhand) (stai_on ?x ?y))
  :effect (and (holding ?x) (clear ?y)
           (not (on ?x ?y)) (not (clear ?x)) (not (emptyhand))))
```

Heuristic detection of the (compatible) full entaglements

- All the detected entanglements in all the training plans $\Rightarrow$ set of the full entanglements
- The algorithm is testing if the conditions of entanglement are satisfied in for every operator instance and the corresponding predicate.

‘flaws’ ratio

- ‘flaws’ ratio – ratio between the number of operator’s instances, where the entanglement is broken, and the total number of operator’s instances in all the training plans
- It weakens the previous heuristics
- It is because planners do not provide optimal plans at all

Setting the ‘flaws’ ratio

1. set *flaws* to *n* according to our experiments we suggest starting with *n* = 0.1
2. generate the entanglements by the modified algorithm using `flaws` ratio *flaws*
3. compare the generated entanglements to the entanglements obtained by the original algorithm (without `flaws`). If same then quit
4. generate a reformulated domain and reformulated training problems
5. run the planner on all the reformulated training problems. If succeed then quit. Otherwise decrease *flaws* by 0.01 and go to the second step.

Experimental evaluation - approach

- Generate training plans by SATPLAN
- Generate reformulated domains and problems considering the detected entanglements
- Run the planners (SATPLAN, SGPLAN, LAMA) both on the original problems and on the reformulated problems and compare results.

Learning phase

Problem	no fr		with fr	
	Unary	Binary	Unary	Binary
Depots	1(2)	2(2)	1(2)	3(3)
Driverlog	1(1)	2(2)	1(1)	2(3)
Storage	2(2)	1(1)	2(2)	1(1)
Zenotravel	0(0)	2(2)	0(0)	2(2)
GoldMiner	3(3)	0(0)	3(3)	0(0)
MatchingBW	5(9)	2(4)	5(9)	3(5)
Parking	3(4)	2(2)	3(4)	2(2)
Thoughtful	15(47)	3(6)	15(47)	3(6)

The table shows how many unary and binary predicates were added and how many times they were added to operators' preconditions (in brackets).

Time comparison (only a fragment)

	SATPLAN		
	time (in sec)		
Problem	**orig**	**ref - no fr**	**ref - w fr**
matching-bw-n15a	27,81	19,93	1,95
matching-bw-n15b	34,25	10,18	1,37
matching-bw-n15c	26,80	9,39	1,20
matching-bw-n15d	40,74	14,41	1,45
matching-bw-n15e	59,00	14,62	1,73
matching-bw-n20a	>600	189,75	15,00
matching-bw-n20b	245,12	54,35	4,39
matching-bw-n20c	363,36	60,94	5,62
matching-bw-n20d	195,87	43,04	4,31
parking-a	>600	399,11	-
parking-b	>600	98,13	-
parking-c	304,47	17,68	-
parking-d	>600	>600	-
parking-e	>600	167,20	-

	SGPLAN		
	time (in sec)		
Problem	**orig**	**ref - no fr**	**ref - w fr**
matching-bw-n15a	>600	>600	0,25
matching-bw-n15b	>600	>600	170,39
matching-bw-n15c	>600	20,89	284,65
matching-bw-n15d	>600	>600	>600
matching-bw-n15e	>600	>600	47,17
matching-bw-n20a	>600	>600	>600
matching-bw-n20b	>600	>600	0,35
matching-bw-n20c	>600	533,89	237,64
matching-bw-n20d	>600	10,40	>600

	LAMA		
	time (in sec)		
Problem	**orig**	**ref - no fr**	**ref - w fr**
depotprob1817	331,03	95,94	3,64
depotprob1916	1,7	0,58	0,14
depotprob4321	4,96	3,69	0,03
depotprob4398	0,23	0,1	0,03
depotprob5646	0,17	0,07	0,02
depotprob5656	>600	>600	3,42
depotprob6178	11,66	6,06	0,06
depotprob6587	0,43	0,19	0,05
depotprob7654	1,48	46,58	12,41
depotprob8715	1,66	0,54	0,17

Plan lengths comparison (only a fragment)

	LAMA		
	plan length		
Problem	**orig**	**ref - no fr**	**ref - w fr**
gold-miner-7x7-01	176	31	-
gold-miner-7x7-02	161	28	-
gold-miner-7x7-03	257	32	-
gold-miner-7x7-04	130	41	-
gold-miner-7x7-05	157	39	-
gold-miner-7x7-06	182	33	-
gold-miner-7x7-07	65	34	-
gold-miner-7x7-08	N/A	25	-
gold-miner-7x7-09	130	29	-
gold-miner-7x7-10	176	31	-
storage_11	32	20	-
storage_12	32	20	-
storage_13	38	20	-
storage_14	32	24	-
storage_15	22	20	-

	SGPLAN		
	plan length		
Problem	**orig**	**ref - no fr**	**ref - w fr**
depotprob1817	100	99	95
depotprob1916	83	N/A	57
depotprob4321	41	35	34
depotprob4398	28	28	28
depotprob5646	26	26	28
depotprob5656	133	70	62
depotprob6178	48	48	37
depotprob6587	28	26	24
depotprob7654	35	33	33
depotprob8715	34	34	36

Discussion

- SATPLAN's running times on reformulated problems were almost always better
- SGPLAN's and LAMA's running times on reformulated problems were occasionally much worse
- Plans quality on reformulated problems were usually the same or slightly better (except gold-miner and storage (LAMA) – significantly better, parking (SGPLAN) – significantly worse)
- All reformulated problems were solvable except two thoughtful ones

Putting together

- Macro-operators serve like `shortcuts` in planning process
- Macro-operators have many instances – high branching factor
- Entanglements can be inherited from primitive operators
- Elimination of actions via entanglements should be helpful

Entanglements and Macro-operators

- If a primitive operator *o* is fully entangled (by init or goal) with a predicate *p* $\Rightarrow$ a macro-operator *m* where *o* is included in *m* is also fully entangled (by init or goal) with a predicate *p* (if present in *o*)

Experimental evaluation - approach

- `Merge` domains reformulated by macro-operators and entanglements
- Run the planners (SATPLAN, SGPLAN, LAMA) both on the original problems and on the reformulated problems and compare results.

Number of reachable actions (in average)

Domain	Macro	Entanglements	Both
Depots	266%	20%	7%
Zeno	85%	85%	70%
Gold-miner	108%	96%	76%

Experimental results (a fragment)

	SATPLAN				LAMA			
	Time (in secs)				Time (in secs)			
Problem	orig	macro	ent	both	orig	macro	ent	both
depotprob1817	>600	err	>600	>600	331.11	93.68	3.68	0.12
depotprob1916	137.85	err	5.36	0.62	1.74	2.46	0.14	0.05
depotprob4321	5.25	2.07	0.46	0.02	4.94	0.25	0.03	0.01
depotprob4398	1.11	2.76	0.29	0.03	0.23	0.41	0.03	0.01
depotprob4534	>600	err	>600	218.62	362.81	1.39	5.86	0.02
depotprob5646	0.38	6.08	0.09	0.14	0.17	0.25	0.02	0.00
depotprob5656	222.28	143.33	5.92	1.10	>600	0.53	3.41	0.02
depotprob6178	6.92	26.11	1.49	0.19	11.61	1.44	0.06	0.02
depotprob6587	3.36	19.02	0.59	0.08	0.42	0.98	0.05	0.02
depotprob7615	>600	err	>600	>600	>600	5.71	>600	0.06
depotprob7654	10.08	16.45	1.41	0.11	1.45	0.59	12.32	0.01
depotprob8715	36.04	err	8.70	0.96	1.65	6.35	0.17	0.04
depotprob9876	>600	err	>600	67.86	580.18	1.27	0.19	0.02

Discussion

- Combining Macro-operators and Entanglements brought benefits as we expected
- In occasional cases the results were much worse (SGPLAN)

Open Issues

- **Losing completeness (reformulated problems might become unsolvable) – find more theoretical aspects**
- Better cooperation of the presented learning methods (+ some other methods maybe)
- Bypass the necessity of defining selection criteria for Macro-operator generation
- Extend the methods for non-classical planning (ADL, time, uncertainty etc.)

Conclusions

- Learning for planning seems to be a reasonable way to improve the planning process
- Presented methods showed that the improvement can be significant
- **There is still a lot to do !**