Try to find a good excuse!

BRA-2015 (Workshop on Belief Revision and Argumentation)

Bernhard Nebel & Moritz Göbelbecker
Department of Computer Science
Foundations of Artificial Intelligence

Albert-Ludwigs-Universität Freiburg

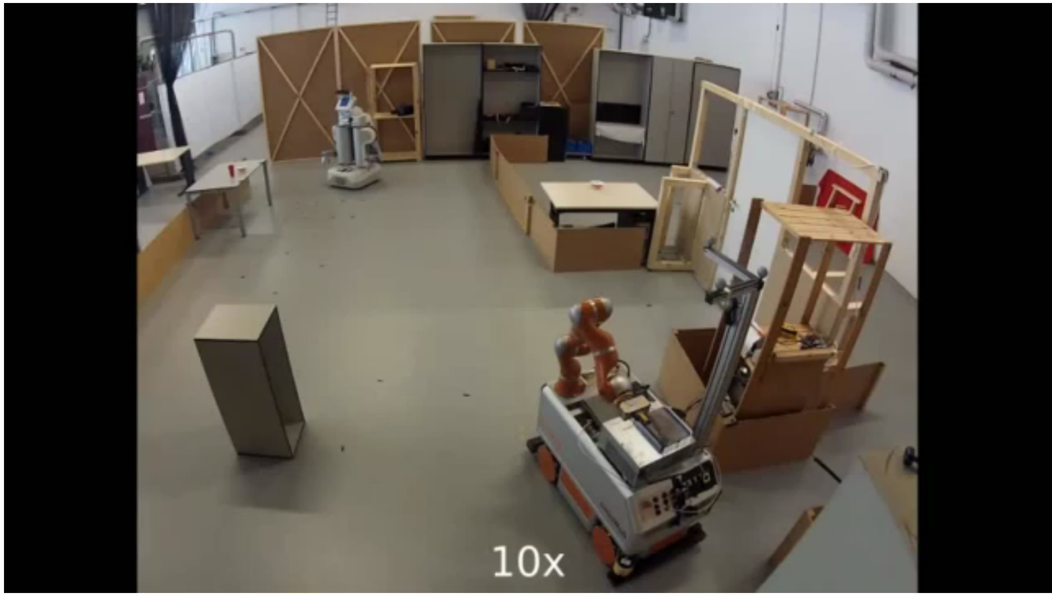
UNI
FREIBURG

Finding excuses

- Motivation
- What is action planning?
- What can be an excuse?
- Possible orderings over excuses
- Computational complexity
- Some computational experiments

Planner-Based Agent Architectures

- Planner-based agents can
  - anticipate the future
  - compose behaviors / motor programs into complex action sequences
  - in order to achieve goals
- Continual planning:
  - monitoring
  - replanning



From final demonstration of our TIDY-UP project

Incompetence: No plan can be found!

- If the robot fails to execute an action, it possibly can recover from it
- If the robot fails to come up with a plan, this is really annoying!
  - domain is not correctly modeled
  - perhaps there are intrinsic reasons (no resources available)
- At least, we want to know what went wrong
- Come up with a counterfactual explanation (excuse)
  - *if only the door were unlocked, I could find a plan to get the coffee and the book for you*
  - Determine a minimal perturbation of the planning task

Finding excuses

- Motivation
- What is action planning?
- What can be an excuse?
- Possible orderings over excuses
- Computational complexity
- Some computational experiments

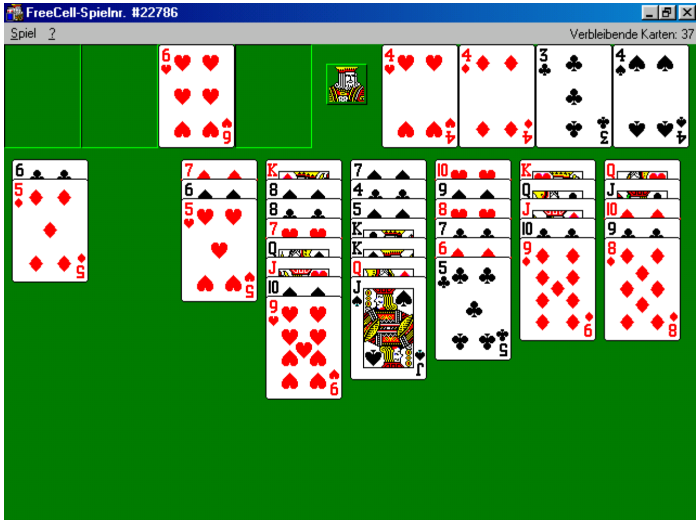
What is planning (in our context)?

- Planning is the process of generating (possibly partial) representations of future behavior prior to the use of such plans to constrain or control that behavior:
  - Planning is the art and practice of thinking before acting [Haslum]

- Kinds of planning:
  - Trajectory planning
  - Manipulation planning
  - Action (or mission) planning

Action planning

- Given
  - an initial state (usually described by using Boolean state variables),
  - a set of possible actions,
  - a specification of the goal conditions,
  - ➢ generate a plan that transforms the current state into a goal state – if there exist one.



Verbleibende Karten: 37

Verbleibende Karten: 0

Ende des Spiels

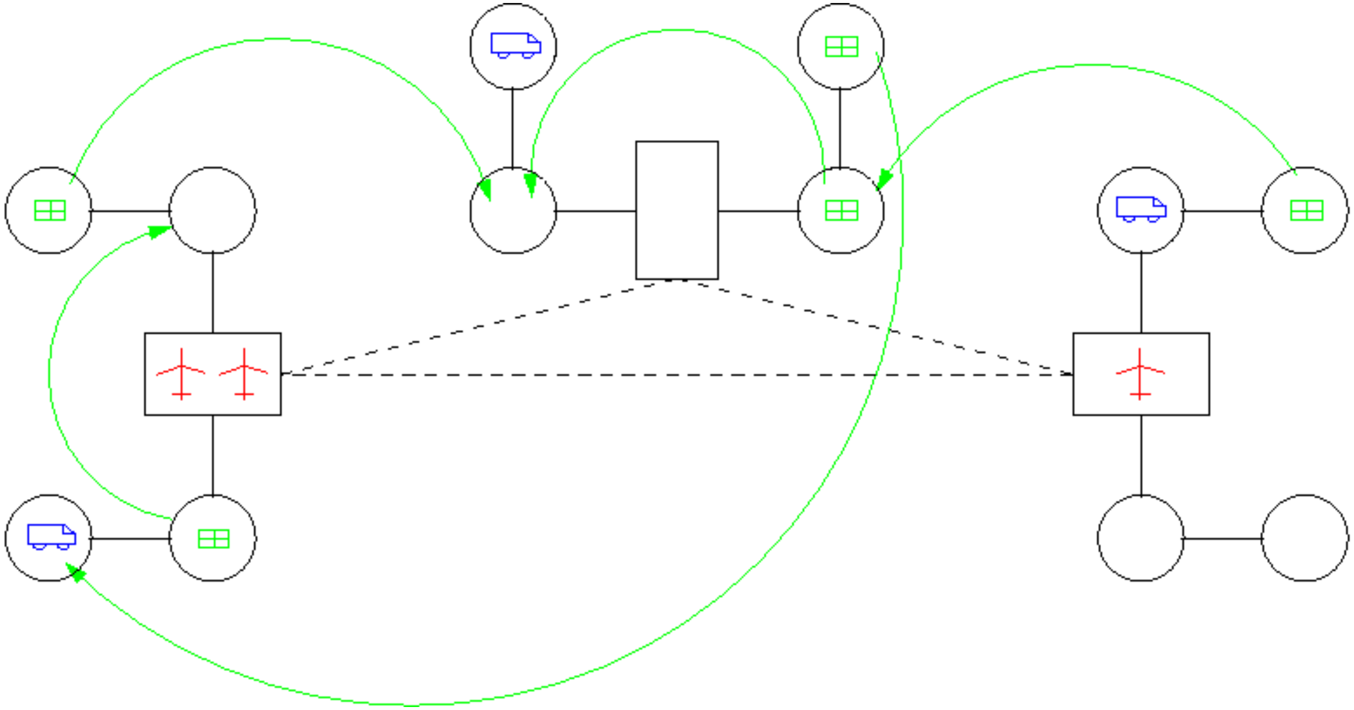
Sie haben gewonnen!

Möchten Sie ein neues Spiel beginnen?

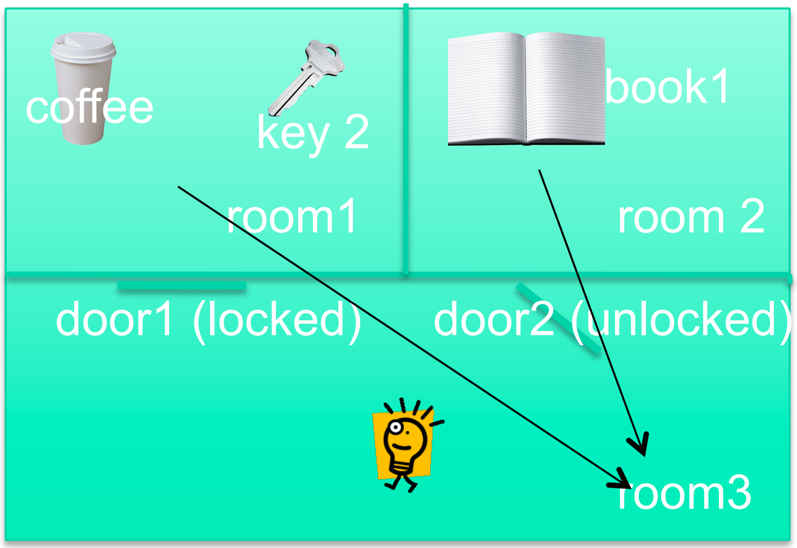
☐ Spiel wählen

Another planning task: *Logistics*

- Given a road map, and a number of trucks and airplanes, make a plan to transport objects from their start positions to their destinations.



Household Robot domain



Given a floor plan, the position of objects and the state of the doors, make a plan to transport objects from their start positions to their destinations.

Domain-independent action planning

- We would like to solve these problems using a general domain-independent solver.

- Start with a declarative specification of the planning task at hand.
- Use a domain-independent planning system to solve the general planning problem
- Issues:
  - What specification language shall we use?
  - How can we solve such planning tasks efficiently?
  - …

A planning formalism: Basic STRIPS

- STRIPS: STanford Research Institute Problem Solver
- Operators: <*para, pre, eff*>
  - *para*: parameters
  - *pre*: conjunctive precondition of atomic facts
  - *effects*: literals that become true after execution of the action
- Actions: variable-free (instantiated) operators
- Initial state description: all positive ground atoms
- Goal description: conjunction of ground literals
- Example for *move* operator in the *Robot domain:*
  - < *(R,S,D), and(room(R), room(S), door(D), unlocked(D), , conn(D,R,S), rin(R)), (¬rin(R), rin(S))* >
- Plan: sequence of actions transforming initial state into a goal state

Household example (1)

- Logical atoms:
  - *room(R), door(D), keyfor(O,D), object(O), rin(R), rholds(O), rfree(), in(O,R), conn(D,R1,R2), unlocked(D)*

- Operators:
  - Move operator *(R, S, D)*: …
  - Take operator *(O,R)*:
    - Precondition: and(*object(O), room(R), in(O,R), rfree()*)
    - Effects: ¬*in(O,R)*, ¬*rfree(), rholds(O)*
  - Put operator *(O,R): …*
  - Unlock operator *(K,D,R,S)*
    - Precondition: and(*object(K),door(D), room(R), room(S), rin(R), conn(D,R,S), keyfor(K,D)*, ¬*unlocked(D), rholds(K)*)
    - Effects: *unlocked(D)*

Household example (2)

- Initial state (described by true ground atoms):
  - S = {*object(c), object(k), room(r1), room(r2), door(d), rin(r1), in(c,r2), conn(d,r1,r2), conn(d,r2,r1), keyfor(k,d), rholds(k)*}
- Goal description:
  - G = {*in(c,r1)*}
- Executing *unlock(k,d,r1,r2)*:
  - $S' = S \cup$ {*unlocked(d)*}
- Succesful plan:
  - Δ = <*unlock(k,d,r1,r2), put(k,r1), move(r1,r2,d), take(c,r2), move(r2,r1,d), put(c,r1)*>

Datalog- and propositional STRIPS

- STRIPS as described allows for unrestricted first-order terms, i.e., arbitrarily nested function terms
  - Infinite state space
  - ➢ semi-decidability
- Simplification: No function terms (only 0-ary terms = constants)
  - DATALOG-STRIPS
  - ➢ EXPTIME-complete
- Simplification: No variables in operators (=actions) or only fixed arity of predicates
  - Propositional STRIPS → used in planning algorithms nowadays (but specification is done using DATALOG-STRIPS)
  - ➢ PSPACE-complete

Finding excuses

- Motivation
- What is action planning?
- What can be an excuse?
- Possible orderings over excuses
- Decidability and computational complexity
- Some computational experiments

Changing a planning task: Excuse types

- One could modify operators (teleport through closed doors):
  - weaken preconditions
  - delete unwanted side effects
  - add wanted effects
- One could change/reduce the goals (bring only the book)
  - only reduction makes sense
- One could change the initial state (door unlocked)

What is a reasonable excuse?

- Reducing goals is sensible, but is already dealt with by *oversubscription planning*, i.e. we will ignore that here.
- For operator modifications, every type of modification seems to be reasonable.
- For initial state modification, making goals directly true does not seem to make sense (which could lead to non-existence of excuses!).
- There are many more operator modifications than state modifications (2^{2n} compared to 2^n).
- For every state mod. we can find an op. mod, but not *vice versa.*
- *We focus on initial state modifications as excuses!*

Excuses formally

Given a planning task $\Pi=(A,O,I,G)$, with A being the set of ground atoms, O being the operators, I the initial state description, and G the goal description, the set $E \subseteq A$ is an excuse iff

- Π is unsolvable,
- E does not contain atoms mentioned in G,
- $I[E]$ is a set such that $a \in I[E]$ iff
  1. $a \in I$ and $a \notin E$ or
  2. $a \notin I$ and $a \in E$,
- $\Pi[E]=(A,O,I[E],G)$ is solvable.

That is, E describes which for which atoms the truth value has to be changed to make Π solvable.

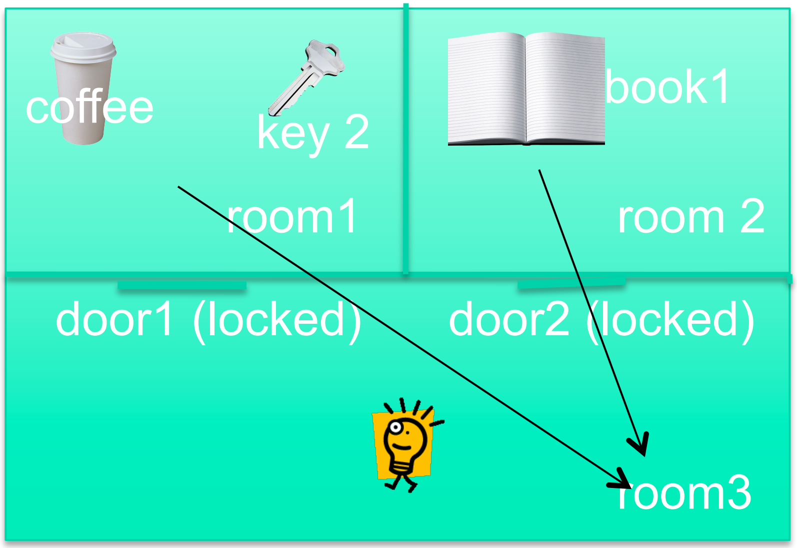
Finding excuses

- Motivation
- What is action planning?
- What can be an excuse?
- Possible orderings over excuses
- Decidability and computational complexity
- Some computational experiments

Preferring Excuses

- Even excluding excuses that make goals true directly (or more restrictively excluding mutex-classes), many possibilities remain.
- One could order them (*E* and *E'* being excuses) by:
    - set inclusion: *E* is preferred over *E'* if $E \subset E'$;
    - cardinality: *E* is preferred over *E'* if $|E|<|E'|$;
    - accumulated weight: Given a weight function *w* from ground atoms to real numbers, *E* is preferred over *E'* if $\sum_{e \in E} w(e) < \sum_{e' \in E'} w(e')$;
    - lexical ordering over linearly ordered priority classes.

Excuses with causal relations

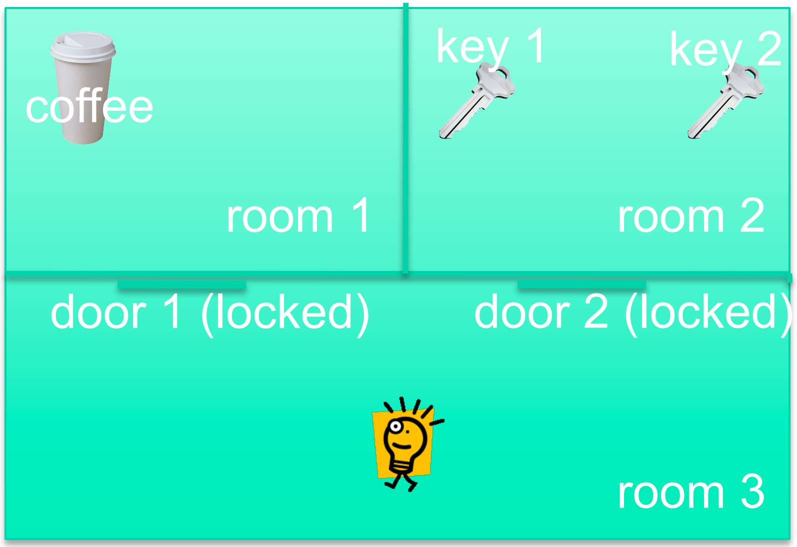


- We could get *book1*, if *door2* were *unlocked*.
- We could get *book1*, if we *had key2*.
- We could get *book1*, if *door1* were *unlocked*.

Preferring causes

- We prefer an excuse *E* over *E’* if there is a plan from I[E] to the goal that contains a state “satisfying the excuse E’”.
- Interestingly, this preference relation by itself is not transitive (since changes by actions are non-monotonic), but we could take the transitive closure.
- The relation is orthogonal to the other preference relations and can be combined with it arbitrarily.

There is a Hole in the Bucket …



The robot could get the coffee, if

- door1 were unlocked,
- we had key 1,
- door2 were unlocked
- we had key 2
- door2 were unlocked
- …

- All excuses in a cycle appear to be equally plausible, and should therefore be equivalent.

Finding excuses

- Motivation
- What is action planning?
- What can be an excuse?
- Possible orderings over excuses
- Computational complexity
- Some computational experiments

Computational Complexity

- Three different reasoning problems:
  - Existence of an excuse (i.e. original task is unsolvable and excuse is possible).
  - Relevance of a ground atom: it is part of one preferred excuse.
  - Necessity of a ground atom: it is part of every preferred excuse.
- All these problems are not harder than planning, provided the underlying planning problem is in a complexity class closed under complementation (e.g. PSPACE) and allows to force operators applied in phases.

Reductions for excuse existence

- Turing reduction from planning to excusing:
  - Given a task Π, construct planning task Π' with new atom *a;*
  - this atom is added to all preconditions and false initially;
  - test whether there are excuses for Π', but not for Π*;*
  - if so, Π is solvable, otherwise not
- Turing reduction from excusing to planning:
  - Given a task Π, construct Π' by adding "initial change operators" for allowed atoms/fluents.
  - If there exists a plan for Π', but not for Π, then there exists some excuse for Π.

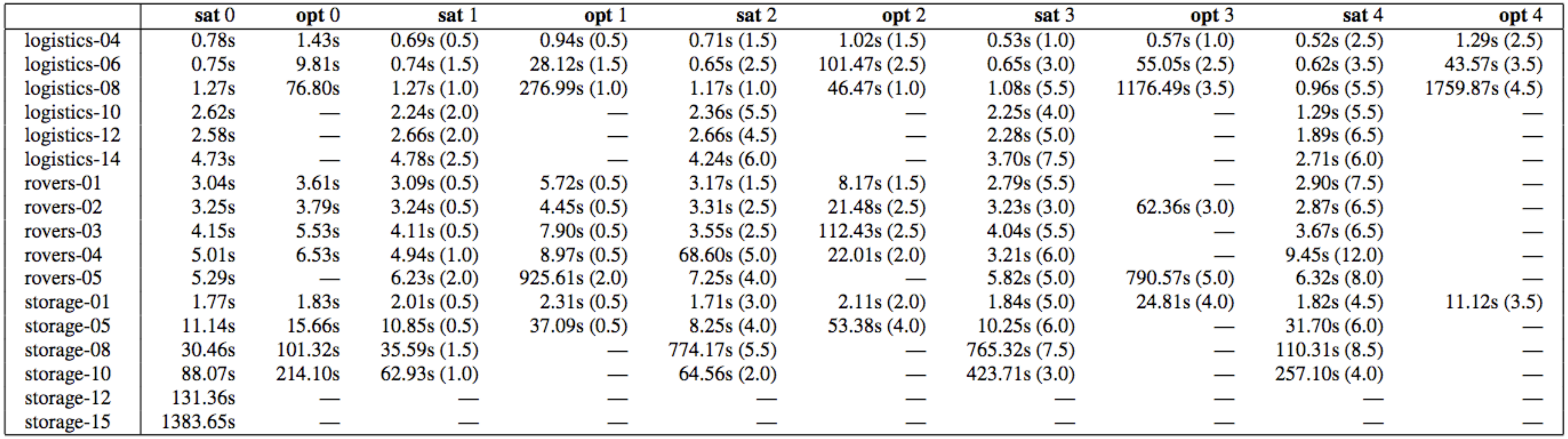
Finding excuses

- Motivation
- What is action planning?
- What can be an excuse?
- Possible orderings over excuses
- Computational complexity
- Some computational experiments

Computing Excuses

- We use our (optimizing) planning system (*Fast Downward*)
- Using the idea from the reduction, we introduce change operators, which can only be applied in an initial phase
- The main issue (for efficiency) is to limit the number of these operators!
- We consider only static facts
- Possible cycles are detected using the causal graph
- This is enough on domains with a certain structure (mutex-free static fluents, strongly connected fluents)
- On general domains, we might not get all possible excuses!

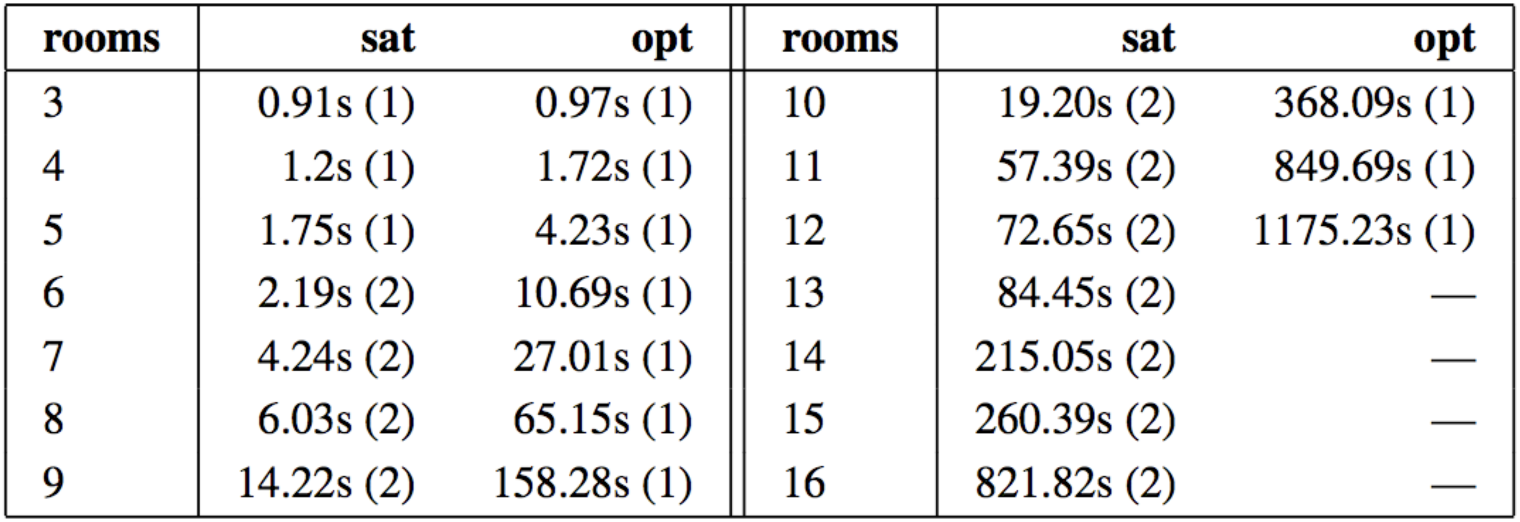
Empirical Results (1)



	sat 0	opt 0	sat 1	opt 1	sat 2	opt 2	sat 3	opt 3	sat 4	opt 4
logistics-04	0.78s	1.43s	0.69s (0.5)	0.94s (0.5)	0.71s (1.5)	1.02s (1.5)	0.53s (1.0)	0.57s (1.0)	0.52s (2.5)	1.29s (2.5)
logistics-06	0.75s	9.81s	0.74s (1.5)	28.12s (1.5)	0.65s (2.5)	101.47s (2.5)	0.65s (3.0)	55.05s (2.5)	0.62s (3.5)	43.57s (3.5)
logistics-08	1.27s	76.80s	1.27s (1.0)	276.99s (1.0)	1.17s (1.0)	46.47s (1.0)	1.08s (5.5)	1176.49s (3.5)	0.96s (5.5)	1759.87s (4.5)
logistics-10	2.62s	—	2.24s (2.0)	—	2.36s (5.5)	—	2.25s (4.0)	—	1.29s (5.5)	—
logistics-12	2.58s	—	2.66s (2.0)	—	2.66s (4.5)	—	2.28s (5.0)	—	1.89s (6.5)	—
logistics-14	4.73s	—	4.78s (2.5)	—	4.24s (6.0)	—	3.70s (7.5)	—	2.71s (6.0)	—
rovers-01	3.04s	3.61s	3.09s (0.5)	5.72s (0.5)	3.17s (1.5)	8.17s (1.5)	2.79s (5.5)	—	2.90s (7.5)	—
rovers-02	3.25s	3.79s	3.24s (0.5)	4.45s (0.5)	3.31s (2.5)	21.48s (2.5)	3.23s (3.0)	62.36s (3.0)	2.87s (6.5)	—
rovers-03	4.15s	5.53s	4.11s (0.5)	7.90s (0.5)	3.55s (2.5)	112.43s (2.5)	4.04s (5.5)	—	3.67s (6.5)	—
rovers-04	5.01s	6.53s	4.94s (1.0)	8.97s (0.5)	68.60s (5.0)	22.01s (2.0)	3.21s (6.0)	—	9.45s (12.0)	—
rovers-05	5.29s	—	6.23s (2.0)	925.61s (2.0)	7.25s (4.0)	—	5.82s (5.0)	790.57s (5.0)	6.32s (8.0)	—
storage-01	1.77s	1.83s	2.01s (0.5)	2.31s (0.5)	1.71s (3.0)	2.11s (2.0)	1.84s (5.0)	24.81s (4.0)	1.82s (4.5)	11.12s (3.5)
storage-05	11.14s	15.66s	10.85s (0.5)	37.09s (0.5)	8.25s (4.0)	53.38s (4.0)	10.25s (6.0)	—	31.70s (6.0)	—
storage-08	30.46s	101.32s	35.59s (1.5)	—	774.17s (5.5)	—	765.32s (7.5)	—	110.31s (8.5)	—
storage-10	88.07s	214.10s	62.93s (1.0)	—	64.56s (2.0)	—	423.71s (3.0)	—	257.10s (4.0)	—
storage-12	131.36s	—	—	—	—	—	—	—	—	—
storage-15	1383.65s	—	—	—	—	—	—	—	—	—

- Instances from the international planning competition
- Limits: 2GB memory and 30 min CPU time
- sat*x* is satisficing while opt*x is* optimal planning
- *x* shows difficulty in repairing, whereby *x=0* is the original (solvable) problem
- Numbers in parentheses are weights
- All in all, it appears that it is possible to find excuses in reasonable time – provided the task was not too difficult

Empirical Results (2)



rooms	sat	opt	rooms	sat	opt
3	0.91s (1)	0.97s (1)	10	19.20s (2)	368.09s (1)
4	1.2s (1)	1.72s (1)	11	57.39s (2)	849.69s (1)
5	1.75s (1)	4.23s (1)	12	72.65s (2)	1175.23s (1)
6	2.19s (2)	10.69s (1)	13	84.45s (2)	—
7	4.24s (2)	27.01s (1)	14	215.05s (2)	—
8	6.03s (2)	65.15s (1)	15	260.39s (2)	—
9	14.22s (2)	158.28s (1)	16	821.82s (2)	—

- Results for cycles with a varying number of rooms (and keys)
- Otherwise the same conditions as before

Related Work

- Similar to abduction (Pierce)
  - Given a consistent logical theory T, a set of literals A (abducibles), and a set O (observations)
  - Find a (minimal) subset E ⊆ A s.t. T,E ⊨ O
- Similar to diagnosis (Reiter):
  - Given a logical theory T and a set of literals N (normality assumptions) s.t. T ∪ N is consistent and measurments M
  - Find a (minimal) subset F ⊆ N s.t. T ∪ (N-F) ∪ M is consistent
- Similar to counterfactuals (Lewis)
  - Given a logical theory L and an implication *a* ℰ *b*
  - Determine the truth of the implication by (minimally) changing the theory in order to make a true.
- Revision and Update
  - when using DL formulae (Herzig)
- Excuses are a bit different
  - action sequences
  - ➢ notion of causality
  - ➢ for this reason, regression and cyclic excuses!

Outlook

- With planner-based agent things can go wrong.
- In particular, it is possible that no plan can be found.
- We may want to know why: Find an excuse!
- This appears to be possible in most case.

- What happens for other types of planning?
- Are there reasonable definitions for operator-based excuses?