Sampling-Based SAT/ASP Multi-Model Optimization as a Framework for Probabilistic Inference

MATTHIAS NICKLES

SCHOOL OF ENGINEERING & INFORMATICS

NATIONAL UNIVERSITY OF IRELAND, GALWAY

Overview

- Introduction
- SAT and Answer Set Programming
- Approach overview
- Measured atoms and parameter atoms
- Cost backtracking
- Deductive inference and parameter learning
- CDCL/CDNL-based implementation (vs. Differential SAT/ASP (PLP’18))
- Results
- Conclusion

Introduction (1)

- Modern SAT and ASP solvers are mature and fast inference tools
- Geared towards complex search, combinatorial and optimization problems (ASP & SAT)
- Strong foothold in industry (SAT)
- Prolog-like syntax but fully declarative (ASP)
- Non-monotonic reasoning (ASP)
- Similar solving techniques, ASP solving ≈ SAT solving + loop handling
- Closely related to Satisfiability Modulo Theories (SMT) and Constraint Programming
- Can translate, e.g., First-Order Logic syntax, action logics, event calculus, … to ASP

Introduction (2)

- Inherently, ASP/SAT inference is a *multi-model* approach
- Solving can produce some or all models as witnesses (if input is satisfiable)
- Models: *stable models* (a.k.a. *answer sets*) (ASP) or satisfying Boolean assignments (SAT)
- Multiple alternative models as a natural way to express non-determinism
- Models as a natural way to represent possible worlds (as for PLP)

Introduction (3)

- How can we utilize SAT/ASP solving to compute not just models but also probability distributions over models?
- We could then solve deductive probabilistic inference tasks in the usual way
- How can we utilize SAT/ASP solving for parameter learning or abduction?
- Idea: formulate these tasks as a multi-model optimization task, using a suitable cost function over multiple models
- Also, we use *sampling*, for higher efficiency: a *sample* represents an approximate solution of the cost function
- Model frequencies in sample (multi-set of models) = possible world probabilities
- Generalized to (in principle) arbitrary multi-model cost functions

Logical input languages

- SAT: we assume (DIMACS-)CNF input (formula in Conjunctive Normal Form)
- ASP: Ground *Answer Set program* consisting of a finite set of normal rules:

  a :- b_1, …, b_k, not b_{k+1},…, not b_m

- Example for an Answer Set program (before grounding):

```
man(dilbert).

single(X) :- man(X), not husband(X).
husband(X) :- man(X), not single(X).
```

…which has two so-called stable models ("answer sets"):

```
Sm1 = { man(dilbert), single(dilbert) }
Sm2 = { man(dilbert), husband(dilbert) }
```

Approach outline (1)

- More powerful than approach presented at PLP'18 (+*measured atoms*, +*cost backtracking*) as well as in a sense *less* general (we use a more specialized approach to differentiation)
- Besides CNF-clauses or an Answer Set program, we require
  - a user-specified *cost function (loss function)*
  - sets of *measured atoms* and *parameter atoms*
- Parameter and measured atoms: user-defined subsets of all atoms/variables
- Measured atoms carry frequencies (normalized atom counts within a sample)
- Parameter atoms make up the solution search space
- Cost function: in principle, arbitrary differentiable function, parameterized with a vector of measured atom frequencies. (But of course not all cost functions are solvable.)

Approach outline (2)

- Idea: incrementally add “customized” models to sample until cost function value ≤ threshold
- Models are computed as usual, except for decisions on *parameter atom* truth assignments:
  Among all unassigned parameter atoms, select a parameter literal (un-/negated parameter atom) with the intend to decrease the cost function value
- If set of parameter atoms = set of measured atoms: approach largely identical to an instance of *Differentiable SAT/ASP* (=> PLP'18 workshop), i.e., a form of discretized gradient descent
- If measured atoms ≠ parameter atoms, we can perform weight learning and abduction - but gradient descent over cost function not usable for this
  (Remark: gradient descent is used for weight learning in other ASP-frameworks)

Approach outline: Using differentiation

- If measured atoms are parameter atoms:
  - Each time we decide on which parameter atom to add positively/negatively to the partial assignment (the current incomplete model candidate):

    compute how this decision would influence the overall cost function

    => partial derivatives of cost function wrt. parameter atoms (as variables representing their occurrence frequencies in the incomplete sample)

    Select parameter atom and its truth value (signed literal) with minimum derivative

  - Otherwise: Cost backtracking…

Approach outline: Using cost backtracking

- Cost backtracking (where previous approach not sufficient):
  - If the cost did not improve at a certain “check point” (e.g., after full candidate model has been generated):
    Undo the latest parameter atom truth value assignment and try another parameter atom truth value on the same branching decision level.
    If no untried parameter atom exists, backtrack further into the past (branching history).
  - Searches the parameter space exhaustively. Can be implemented utilizing already existing (fast) CDCL/CDNL-style backtracking (which is normally used to resolve conflicts).

  - Cost backtracking can be used alone or in combination with differentiation (for mixed deductive probabilistic inference + weight learning scenarios)
  - Less efficient than gradient descent

Cost functions for deductive probabilistic inference (1)

- Various possibilities for cost function. For deductive probabilistic inference, we can use *Mean Squared Error* (MSE):

$$cost(\theta_1^v, \theta_2^v, ...) := \frac{1}{n}\sum_{i=1}^{n}(\beta(\theta_i) - \phi_i)^2$$

- Here: set of measured atoms = set of parameter atoms.
- *Measured* atoms θ_i here: atoms carrying probabilities (*weights*) ϕ_i (provided by user)
- Measured atom frequencies $\beta(\theta_i)$ updated after each sampled model
- Weighted rules and weighted models can be de-sugared as instances of this approach (similar to normalization step in PSAT)
- Arbitrary MSE cost, parameter and measured atoms; no required independence assumptions. (But of course not all cost functions and background logic programs/formulas have a solution.)

Cost functions for deductive probabilistic inference (2)

- The previous approach using MSE as cost effectively computes one(*) of the solutions of an implicitly given system of linear equations (in the exact case)
- Model frequencies (approximate possible world probabilities) in the resulting sample are the unknowns of this system
- We get an (approximated) probability distribution over models (possible worlds)
- Additional constraints ensuring Σ $Pr(m_i)=1$ and $0 \leq Pr(m_i) \leq 1$ implicitly hold
- Telling whether or not this system has a solution => *PSAT*
- For probabilistic inference, we can then query the distribution for the probabilities of facts and rules as usual (by summing up the model probabilities in the sample where the query holds)

(*) though not necessarily the one with maximum entropy

Cost functions for deductive probabilistic inference (3)

- The Answer Set input program (or analogously SAT formula) can contain arbitrary rules and facts
- For parameter atoms, it is sensible to add so-called *spanning rules* which make the parameter atoms nondeterministic (although this is not a requirement for our algorithms)

```
0{a}1. % spanning rule for parameter atom a
0{b}1.
:- a, b. % an example for a hard rule
```

- Suitable MSE cost function here, e.g.,: $\frac{1}{2}((\beta(a)-0.2)^2+(\beta(b)-0.6)^2)$

(assigns atom a weight 0.2 and atom b weight 0.6)

Cost functions for parameter learning (1)

- Task: Learning weights of measured atoms (*parameter learning*) from examples e_i
- General Approach: Maximization of $\prod \Pr(e_i|sample)$
- This translates directly into a suitable cost function $1 - \prod_i \beta(sample)_i$
- The β(sample)$_i$ represent again measured atom frequencies.
  These measured atoms are *not* identical to parameter atoms now.
- Each measured atom represents a given learning example (some nondeterministic atom which occurs in the Answer Set program or SAT formula)
- The hypotheses are the parameter atoms (not part of the cost function)

Cost functions for parameter learning (2)

- Minimizing cost here thus means: search for parameter atom frequencies which maximize example frequencies
- We cannot use differentiation of the cost function here but use cost backtracking
- A simple example: We are looking for the weight of hypothesis h, given background rule $e :- h.$ (remark: h is also an abducible here)
- We do so by minimizing a cost function which maximizes Pr(e), such as $(1 - \beta(sample)(e))^2$
- Parameter search space is $\{h, \neg h\}$. Each time we generate a new model, we add h positively or negatively to the model, depending on whether this decreases the cost function, using cost backtracking. Mixed scenarios (with weighted rules and deduction) also possible.

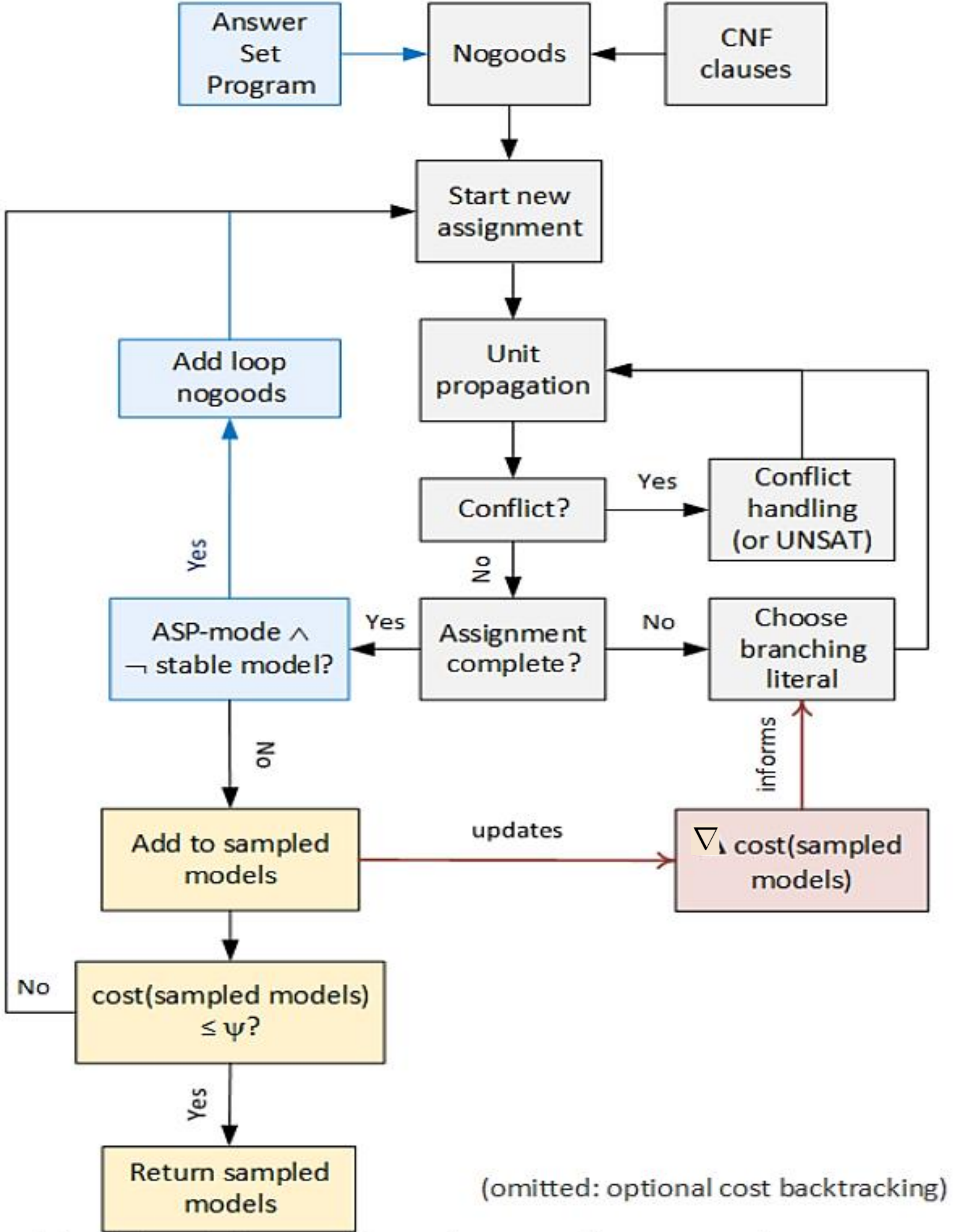
ΔSAT/ASP-CDNL (1)

- Enhancement of current state of the art SAT/ASP solving algorithm, i.e., CDCL/CDNL
- Opposed to $\partial SAT/ASP$ (PLP’18), the Δ refers to “difference” (as approach comprises also a non-derivative-based method)
- CDCL (*Conflict-Driven Clause Learning*) based on older DPLL algorithm but with clause learning capability and non-chronological backtracking
- CDNL-ASP (*Conflict-Driven Nogood Learning*): variant of CDNL with *nogoods* (think of clauses with negated literals) as basic representative concept
- Also comprises loop handling (required for non-tight Answer Set programs)
- CDNL used by Clingo/Clasp (but we created an independent implementation in Scala). Instances suitable for SAT as well as ASP solving
- We enhance CDNL-ASP with a new decision literal selection (branching) policy and cost backtracking.

ΔSAT/ASP-CDNL (2)

- In case measured = parameter atoms and cost function = MSE (or other differentiable cost):
  We just need a branching literal selection approach which selects the unassigned parameter literal which decreases the cost function most.
- Recall that cost function is a term parameterized with the frequencies vector β(sample) of the measured atoms in the current incomplete sample
- Efficient: Automatic or symbolic differentiation which is applied (per each parameter literal) only after a new model has been sampled (!).
- Less efficient but easy to implement: Approximate numerical differentiation which evaluates the full cost function in each step (cost term might be very large).

Without cost backtracking, we get a variant of *Differentiable* SAT/ASP using Diff-CDNL-ASP/SAT (=> PLP’18)



Answer Set Program

Nogoods

CNF clauses

Start new assignment

Unit propagation

Conflict?

Yes

Conflict handling (or UNSAT)

No

Assignment complete?

No

Choose branching literal

Yes

ASP-mode ∧ ¬ stable model?

Yes

Add loop nogoods

No

Add to sampled models

updates

∇ cost(sampled models)

informs

cost(sampled models) ≤ ψ?

No

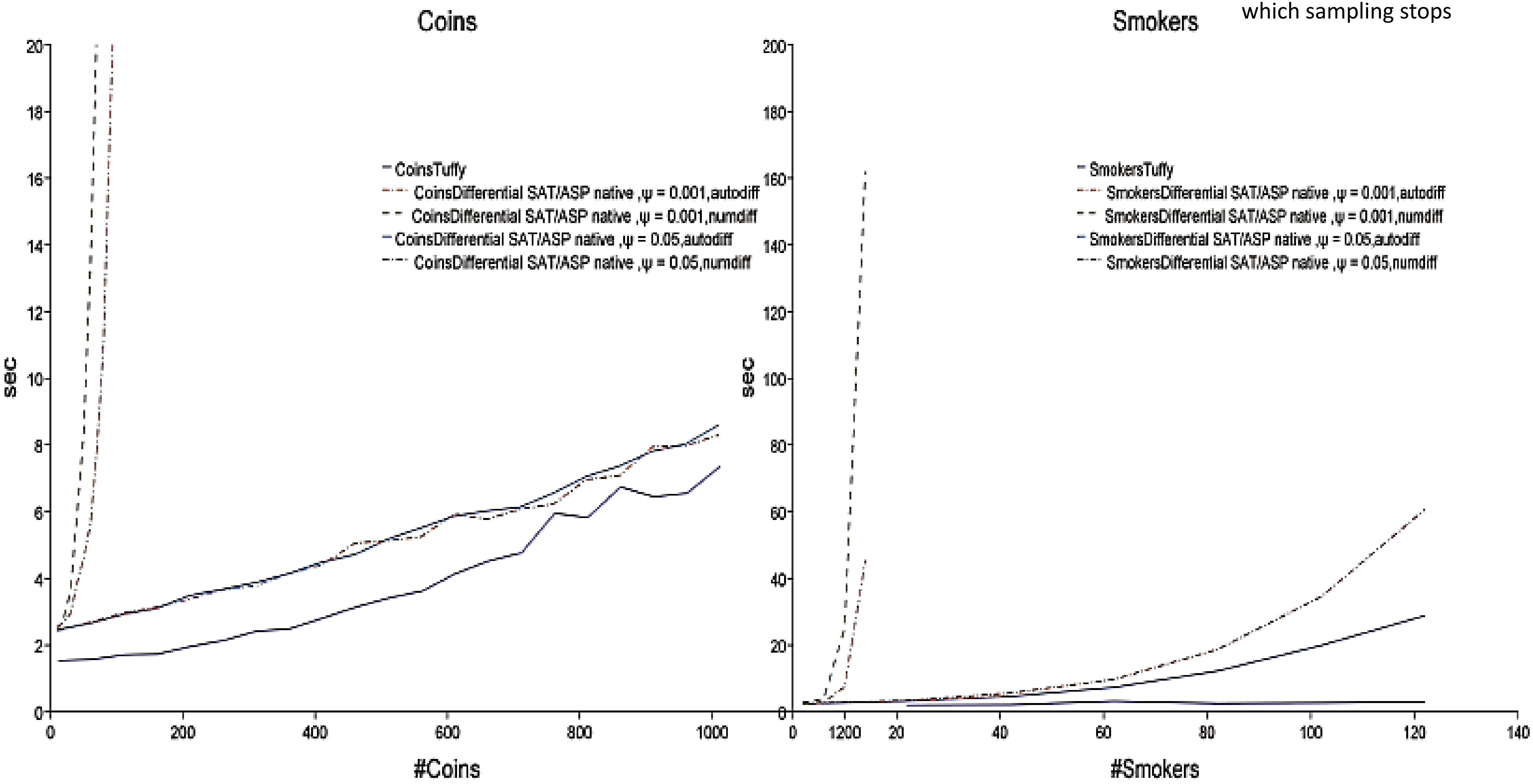
Yes

Return sampled models

(omitted: optional cost backtracking)

Initial experiments

ψ = MSE cost convergence threshold (accuracy) at which sampling stops



Conclusion

- Multi-model optimization with custom cost functions and user-specified accuracy
- Uses iterative Boolean assignment / answer set sampling for better scalability
- Resulting sample represents one (approximate) solution of the cost function
- Probabilistic inference as instance of approach. No restriction of random variable dependency.
- Separability of parameter (i.e., search space) and measured atoms (i.e., learning examples) for weight learning
- Prototype implementation as enhancement of CDCL/CDNL
- Model generation controlled by a form of gradient descent or cost backtracking
- Future work: alternative to cost backtracking for weight learning, further experiments, maximum-entropy solutions, theoretical criteria for termination (beyond convex cost functions)

Any questions?