18th International Conference on Automated Planning and Scheduling
September 14-18. 2008 Sydney, Australia

ICAPS-08 Tutorial on

First-Order Planning Techniques

Organizers

Scott Sanner, NICTA (Australia)
Kristian Kersting, Fraunhofer IAIS (Germany)

ICAPS 2008 Tutorial

Techniques for First-order Planning

Motivation

NICTA

Fraunhofer Institut Intelligente Analyse- und Informationssysteme IAIS

Scott Sanner
NICTA

Kristian Kersting
Fraunhofer IAIS

with an invited presentation by Saket Joshi, Tufts University

---

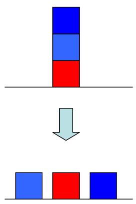
Tutorial Outline

- **Motivation**
- Deductive First-order Planning
  - Situation Calculus
  - Symbolic Dynamic Programming
  - Relational Bellman Algorithm (ReBeL)
  - First-order Decision Diagrams (FODDs)
- Inductive First-order Planning
- Conclusion

Planning Languages

- Common languages:
  - STRIPS
  - PDDL
    - more expressive than STRIPS
    - for example, *universal* and *conditional* effects:

```
(:action put-all-blue-blocks-on-table
     :parameters ( )
     :precondition ( )
     :effect (forall (?b)
               (when (Blue ?b)
                     (not (OnTable ?b)))))
```



  - General Game Playing (GGP)
    - one or more agents

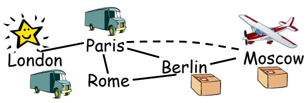
Benefits of Relational Languages

- STRIPS, PDDL, GGP are relational languages…
  - Refer to relational *fluents:*
    - e.g., *BIn(?b,?c), OnTable(?b)*
    - specify relations between objects
    - change over time
  - Use first-order logic to specify…
    - action preconditions
    - action effects
    - goals / rewards
      - e.g., *(forall (?b ?c) ((Destination ?b ?c) ⇒ (Bin ?b ?c)))*
  - Are *domain-independent* and often compact!

How to Solve?

- Relational planning *problem*:

BoxWorld:




```
(:action load-box-on-truck-in-city
     :parameters (?b - box ?t - truck ?c – city)
     :precondition (and (BIn ?b ?c) (TIn ?t ?c))
     :effect (and (On ?b ?t) (not (BIn ?b ?c))))
```

- Solve *ground problem* for *each domain instance*?
  - 3 trucks: 2 planes: 3 boxes:
- Or solve lifted specification for *all* domains at once?

Full Specification: BoxWorld

- **Relational Fluents:** $BoxIn(Box, City), TruckIn(Truck, City), BoxOn(Box, Truck)$
- **Goal:** $[\exists Box : b.BoxIn(b, paris)]$
- **Actions:**
  - $load(Box : b, Truck : t)$:
    - Effects:
      - $when\ [\exists City : c.\ BoxIn(b,c) \wedge TruckIn(t,c)]\ then\ [BoxOn(b,t)]$
      - $\forall City : c.\ when\ [BoxIn(b,c) \wedge TruckIn(t,c)]\ then\ [\neg BoxIn(b,c)]\}$
  - $unload(Box : b, Truck : t)$:
    - Effects:
      - $\forall City : c.\ when\ [BoxOn(b,t) \wedge TruckIn(t,c)]\ then\ [BoxIn(b,c)]$
      - $when\ [\exists City : c.\ BoxOn(b,t) \wedge TruckIn(t,c)]\ then\ [\neg BoxOn(b,t)]$
  - $drive(Truck : t, City : c)$:
    - Effects:
      - $when\ [\exists City : c_1.\ TruckIn(t,c_1)]\ then\ [TruckIn(t,c)]$
      - $\forall City : c_1.\ when\ [TruckIn(t,c_1)]\ then\ [\neg TruckIn(t,c_1)]$

Solving Ground BoxWorld

- Apply planner to BoxWorld grounded w.r.t. domain, e.g.,

#states exponential in #state-vars!

- **Domain Object Instantiation**:
  - $Box = \{box_1, box_2, box_3\}$, $Truck = \{truck_1, truck_2\}$, $City = \{paris, berlin, rome\}$
- **Ground Fluents (i.e., binary state variables)**: Exponential #state-vars in arity
  - $BoxIn$: $\{BoxIn(box_1, paris), BoxIn(box_2, paris), BoxIn(box_3, paris), BoxIn(box_1, berlin), BoxIn(box_2, berlin), BoxIn(box_3, berlin), BoxIn(box_1, rome), BoxIn(box_2, rome), BoxIn(box_3, rome)\}$
  - $TruckIn$: $\{TruckIn(truck_1, paris), TruckIn(truck_1, berlin), TruckIn(truck_1, rome), TruckIn(truck_2, paris), TruckIn(truck_2, berlin), TruckIn(truck_2, rome)\}$
  - $BoxOn$: $\{BoxOn(box_1, truck_1), BoxOn(box_2, truck_1), BoxOn(box_3, truck_1), BoxOn(box_1, truck_2), BoxOn(box_2, truck_2), BoxOn(box_3, truck_2)\}$
- **Ground Actions**: Exponential #actions in arity
  - $load$: $\{load(box_1, truck_1), load(box_2, truck_1), load(box_3, truck_1)\ load(box_1, truck_2), load(box_2, truck_2)\}, load(box_3, truck_2)\}$
  - $unload$: $\{unload(box_1, truck_1), unload(box_2, truck_1), unload(box_3, truck_1), unload(box_1, truck_2), unload(box_2, truck_2)\}, unload(box_3, truck_2)\}$
  - $drive$: $\{drive(truck_1, paris), drive(truck_1, berlin), drive(truck_1, rome)\ drive(truck_2, paris), drive(truck_2, berlin), drive(truck_2, rome)$
- **Goal**: $[BoxIn(box_1, paris) \vee BoxIn(box_2, paris) \vee BoxIn(box_3, paris)]$

Exponential in #nested quantifiers

A First-order Solution to BoxWorld

- Derive solution deductively at lifted PDDL level:
  - if $(\exists b.BoxIn(b, paris))$ then do $noop$
  - else if $(\exists b^*, t^*.TruckIn(t^*, paris) \wedge BoxOn(b^*, t^*))$ then do $unload(b^*, t^*)$
  - else if $(\exists b, c, t^*.BoxOn(b, t^*) \wedge TruckIn(t, c))$ then do $drive(t^*, paris)$
  - else if $(\exists b^*, c, t^*.BoxIn(b^*, c) \wedge TruckIn(t^*, c))$ then do $load(b^*, t^*)$
  - else if $(\exists b, c_1^*, t^*, c_2.BoxIn(b, c_1^*) \wedge TruckIn(t^*, c_2))$ then do $drive(t^*, c_1^*)$
  - else do $noop$

Optimal for *any* domain instantiation!

- Great, but how do I obtain this solution?

Tutorial Overview

- Foundational theory for exploiting first-order structure in planning
  - deterministic and probabilistic
  - representations and implementation
- The first part covers a *deductive* approach
  - plan solely based on model
  - no simulations or sampled data
    - requires grounding
- The second part reviews *inductive* approaches

ICAPS 2008 Tutorial

Techniques for First-order Planning

Deterministic Planning in the Situation Calculus

Scott Sanner
NICTA

Tutorial Outline

- Motivation
- **Deductive First-order Planning**
  - **Situation Calculus**
  - Symbolic Dynamic Programming
  - Relational Bellman Algorithm (ReBeL)
  - First-order Decision Diagrams (FODDs)
- Inductive First-order Planning
- Conclusion

Situation Calculus: Ingredients

- Actions
  - first-order terms with action parameters
  - e.g., *load(b,t), unload(b,t), drive(t,c)*
- Situations
  - term that encodes action history
  - e.g., *s*, s_0, *do(load(b,t),s), do(load(b,t),drive(t,c),s)*
- Fluents
  - relation whose truth value varies b/w situations
  - e.g., *BoxOn(b,t,s), TruckIn(t,c,s), BoxIn(t,c,s)*

Situation Calculus: PDDL to Effects

- Recall BoxWorld PDDL specification…

- $load(Box : b, Truck : t)$:
  - Effects:
    - $when\ [\exists City : c.\ BoxIn(b,c) \wedge TruckIn(t,c)]\ then\ [BoxOn(b,t)]$
    - $\forall City : c.\ when\ [BoxIn(b,c) \wedge TruckIn(t,c)]\ then\ [\neg BoxIn(b,c)]\}$
- $unload(Box : b, Truck : t)$:
  - Effects:
    - $\forall City : c.\ when\ [BoxOn(b,t) \wedge TruckIn(t,c)]\ then\ [BoxIn(b,c)]$
    - $when\ [\exists City : c.\ BoxOn(b,t) \wedge TruckIn(t,c)]\ then\ [\neg BoxOn(b,t)]$
- $drive(Truck : t, City : c)$:
  - Effects:
    - $when\ [\exists City : c_1.\ TruckIn(t,c_1)]\ then\ [TruckIn(t,c)]$
    - $\forall City : c_1.\ when\ [TruckIn(t,c_1)]\ then\ [\neg TruckIn(t,c_1)]$

Situation Calculus: PDDL to Effects

- Translate to positive and negative effect axioms

- $load(Box : b, Truck : t)$:
  - Effects:
    - $[\exists c.\ a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)] \supset BoxOn(b,t,do(a,s))$
    - $[\exists t.\ a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)] \supset \neg BoxIn(b,c,do(a,s))$
- $unload(Box : b, Truck : t)$:
  - Effects:
    - $[\exists t.\ a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)] \supset BoxIn(b,c,do(a,s))$
    - $[\exists c.\ a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)] \supset \neg BoxOn(b,t,do(a,s))$
- $drive(Truck : t, City : c)$:
  - Effects:
    - $[\exists c_1.\ a = drive(t,c) \wedge TruckIn(t,c_1,s)] \supset TruckIn(t,c,do(a,s))$
    - $[\exists c.\ a = drive(t,c) \wedge TruckIn(t,c_1,s)] \supset \neg TruckIn(t,c_1,do(a,s))$

Situation Calculus: PDDL to Effects

- Now, merge into positive effect axioms

$$\gamma_F^+(\vec{x},a,s) \supset F(\vec{x},do(a,s))$$

and negative effect axioms

$$\gamma_F^-(\vec{x},a,s) \supset \neg F(\vec{x},do(a,s))$$

- Use **rule** to combine multiple effects

$$[(C_1 \supset F) \wedge (C_2 \supset F)] \equiv [(C_1 \vee C_2) \supset F]$$

Frame Problem

- Now we have positive and negative effects

$\gamma^+_{BoxIn}(\vec{x},a,s) \supset BoxIn(\vec{x},do(a,s))$ $\quad$ $\gamma^-_{BoxIn}(\vec{x},a,s) \supset \neg BoxIn(\vec{x},do(a,s))$

$\gamma^+_{TruckIn}(\vec{x},a,s) \supset TIn(\vec{x},do(a,s))$ $\quad$ $\gamma^-_{TruckIn}(\vec{x},a,s) \supset \neg TruckIn(\vec{x},do(a,s))$

$\gamma^+_{BoxOn}(\vec{x},a,s) \supset BoxOn(\vec{x},do(a,s))$ $\quad$ $\gamma^-_{BoxOn}(\vec{x},a,s) \supset \neg BoxOn(\vec{x},do(a,s))$

so we have compactly specified what changes.

- How to compactly specify what does not change?
  - Infamous **Frame Problem**
  - Intuition:
    - "what does not change, remains same"
    - this is Reiter's **Default Solution**
    - but we have to logically formalize it…

Successor State Axioms (SSAs)

- Default solution to frame problem given as SSAs:

Unique names axioms for terms

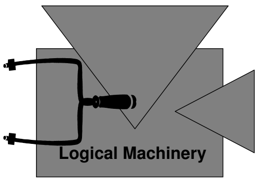
Effect axioms

$\gamma_F^+(\vec{x},a,s) \supset F(\vec{x},do(a,s))$

$\gamma_F^-(\vec{x},a,s) \supset \neg F(\vec{x},do(a,s))$

Unique names for actions / arguments.

Explanation closure axioms



$F(\vec{x},do(a,s)) \equiv \gamma_F^+(\vec{x},a,s) \vee F(\vec{x},s) \wedge \neg\gamma_F^-(\vec{x},a,s)$

SSAs

SSAs

- Shorthand:

$$F(\vec{x},do(a,s)) \equiv \Phi_F(\vec{x},a,s)$$
$$\equiv \gamma_F^+(\vec{x},a,s) \vee F(\vec{x},s) \wedge \neg\gamma_F^-(\vec{x},a,s)$$

What changes and does not change!

- Reality check:

$BoxOn(b,t,do(a,s)) \equiv \Phi_{BoxOn}(b,t,a,s)$
$\equiv [\exists c.\ a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)]$
$\vee\ BoxOn(b,t,s) \wedge \neg[\exists c.\ a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)]$

$BoxIn(b,c,do(a,s)) \equiv \Phi_{BoxIn}(b,c,a,s)$
$\equiv [\exists t.\ a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)]$
$\vee\ BoxIn(b,c,s) \wedge \neg[\exists t.\ a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)]$

$TruckIn(t,c,do(a,s)) \equiv \Phi_{TruckIn}(t,c,a,s)$
$\equiv [\exists c_1.\ a = drive(t,c) \wedge TruckIn(t,c_1,s)]$
$\vee\ TruckIn(t,c,s) \wedge \neg[\exists c_1.\ a = drive(t,c) \wedge TruckIn(t,c_1,s)]$

Regression

- Why have we defined SSAs?
- Regression:
  - If φ held after action *a* then *regression* is the φ' that held before action *a*
- Exploit following properties:
  - $Regr(\neg\psi) = \neg Regr(\psi)$
  - $Regr(\psi_1 \wedge \psi_2) = Regr(\psi_1) \wedge Regr(\psi_2)$
  - $Regr((\exists x)\psi) = (\exists x) Regr(\psi)$
  - $Regr(F(\vec{x}, do(a,s))) = \Phi_F(\vec{x}, a, s)$

Regression Example

- Given $\exists b.\ BoxIn(b, paris, do(unload(b^*, t^*), s))$
- Regress through *unload(b*,t*)*

$$\begin{aligned}
&Regr(\exists b.\ BoxIn(b, paris, do(unload(b^*,t^*),s)))\\
&= \exists b.\ \Phi_{BoxIn}(b, paris, unload(b^*,t^*), s)\\
&= \exists b.\ [\exists t.\ unload(b^*,t^*) = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t, paris, s)]\\
&\quad \vee BoxIn(b, paris, s)\\
&\quad\ \wedge \neg[\exists t.\ unload(b^*,t^*) = load(b,t) \wedge BoxIn(b, paris, s) \wedge TruckIn(t, paris, s)]\\
&= [\exists b, t.\ b = b^* \wedge t = t^* \wedge BoxOn(b,t,s) \wedge TruckIn(t, paris, s)]\\
&\quad \vee \exists b.\ BoxIn(b, paris, s)
\end{aligned}$$

make non-empty domain assumption

$$\begin{aligned}
&= [(\exists b. b = b^*) \wedge (\exists t. t = t^*) \wedge BoxOn(b^*,t^*,s) \wedge TruckIn(t^*, paris, s)]\\
&\quad \vee \exists b.\ BoxIn(b, paris, s)
\end{aligned}$$

note free vars b^*, t^*; why?

$$= [BoxOn(b^*,t^*,s) \wedge TruckIn(t^*, paris, s)] \vee \exists b.\ BoxIn(b, paris, s)$$

Regression Example

- But what action instantiation of *unload(b*,t*)* leads to:

$$\exists b.\ BoxIn(b, paris, do(unload(b^*,t^*), s))$$

- Just have to existentially quantify b^*, t^*
  - Can obtain instances via query extraction w.r.t. state KB

$$\begin{aligned}
&\exists b^*, t^*.\ Regr(\exists b.\ BoxIn(b, paris, do(unload(b^*,t^*),s)))\\
&= \exists b^*, t^*.\ [BoxOn(b^*,t^*,s) \wedge TruckIn(t^*, paris, s)]\\
&\quad \vee \exists b.\ BoxIn(b, paris, s)\\
&= [\exists b^*, t^*.\ BoxOn(b^*,t^*,s) \wedge TruckIn(t^*, paris, s)]\\
&\quad \vee \exists b.\ BoxIn(b, paris, s)
\end{aligned}$$

First-order state & action abstraction!
Don't have to enumerate all states, *b**, *t**!

Recap

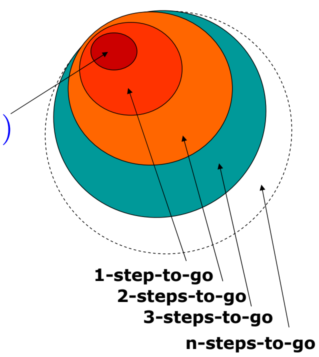
- We translated PDDL to SitCalc theory
  - converted PDDL effects to SitCalc effect axioms
  - derived SSAs from effect axioms
    - using default solution to Frame Problem
- Introduced regression operator
  - extracted action instantiation to achieve goal
- Let the planning begin…

Regression Planning

- Define abstract goal state, e.g.,

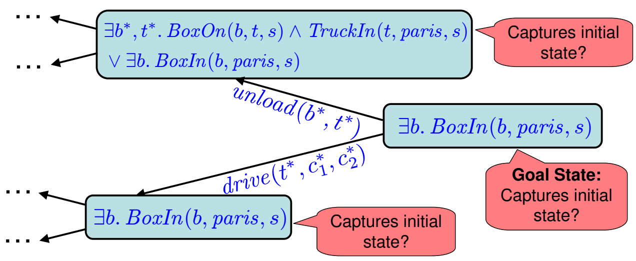
$$\exists b.\ BoxIn(b, paris, s)$$

- Check if regression through action sequence holds in initial state



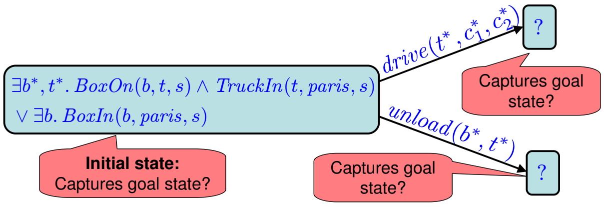
First-order Goal-regression

- We can now do goal regression planning!
  - Provide initial state and sequence of actions
  - Use regression, ∃ to tell whether goal will hold



Progression and Forward-search?

- Can we do lifted *forward-search planning*?



  - *Progression* not first-order definable! (Reiter, 01)
  - Could progress *ground* state
    - But this does not exploit first-order structure

Golog: Restricted Plan Search

- AlGOl in LOGic
  - Search the space of sequential action plans
  - Regress actions to initial state to test reachability
  - *Restrict* action space with program:

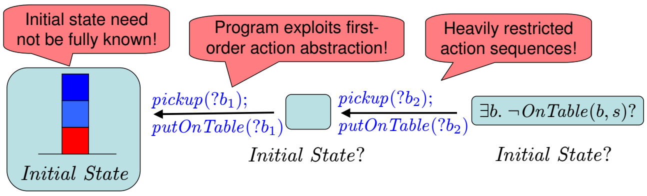
α	primitive action
$\phi?$	condition test
(δ_1, δ_2)	sequence
if ϕ **then** δ_2 **endIf**	conditional
while ϕ **then** δ **endWhile**	loop
$(\delta_1 \vert \delta_2)$	nondeterministic choice of action
$\pi\ \vec{x}\ [\delta]$	nondeterministic choice of arguments
δ^*	nondeterministic iteration
proc $\beta(\vec{x})\ \delta$ **endProc**	procedure call definition
$\beta(\vec{t})$	procedure call

Golog Example

- Golog Program:

$$(\pi\, b\, [\neg OnTable(b, s)?, pickup(b), putOnTable(b)])^*,$$
$$\forall b.\ OnTable(b, s)?$$

- Diagram of Golog Planning:



For Further Reading

- Knowledge in Action:
  In-depth coverage of SitCalc default solution, applications
  (Reiter, 2001)
- Golog
  (Levesque, Reiter, Lesperance, Lin,
  Journal Logic Programming, 1997)
- Extensions
  - ConGolog: concurrent Golog
    (de Giacomo, Lesperance, Levesque, AIJ-00)
  - DT-Golog: decision-theoretic, covered next
    (Soutchanski, Boutilier, Reiter, Thrun, AAAI-20)

For MDPs, covered next.

Conclusion

- Situation Calculus
  - First-order specification of action theory
  - Default solution addresses Frame Problem
    - Effective approach to PDDL-expressive planning
- Supports Regression Planning
  - Initial state need not be fully specified
  - Can restrict action space with Golog program
  - Exploits state & action abstraction
    - Avoids enumerating all state & action instances!

ICAPS 2008 Tutorial

Techniques for First-order Planning

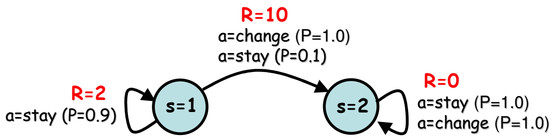
FOMDPs and Symbolic Dynamic Programmming

Scott Sanner
NICTA

Tutorial Outline

- **Motivation**
- **Deductive First-order Planning**
  - Situation Calculus
  - **Symbolic Dynamic Programming**
  - Relational Bellman Algorithm (ReBeL)
  - First-order Decision Diagrams (FODDs)
- Inductive First-order Planning
- Conclusion

MDPs <S,A,T,R,γ>

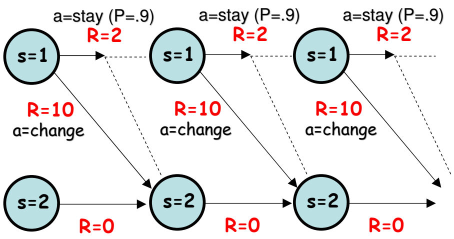


- S = {1,2}; A = {stay, change}
- Reward
  - R(s=1,a=stay) = 2
  - …
- Transitions
  - T(s=1,a=stay,s'=1) = P(s'=1 | s=1, a=stay) = .9
  - …
- Discount γ

How to act in an MDP?

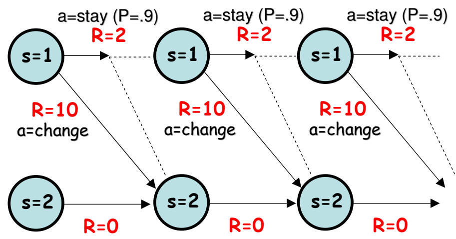
Define policy π: S → A

What's the best Policy?



- Immediate vs. long-term gain?

What's the best Policy?



- Must define reward criterion to optimize!
  - Discount factor γ important (γ=.9 vs. γ=.1)

MDP Policy, Value, & Solution

- Define value of a policy π:

$$V_\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t \middle| s = s_0 \right]$$

- Tells how much value you expect to get by following π starting from state *s*
- MDP Optimal Solution:
  - Find optimal policy π^* that maximizes value
  - Fortunately: $\exists \pi^* . \forall s, \pi .\ V_{\pi^*}(s) \geq V_\pi(s)$

Value Iteration: from finite to ∞ decisions

- Given optimal *t-1-stage-to-go* value function
- How to act optimally with *t* decisions?
  - Take action *a* then act so as to achieve V^{t-1} thereafter

$$Q^t(s,a) := R(s,a) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \cdot V^{t-1}(s')$$

  - What is expected value of best action *a* at decision stage *t*?

$$V^t(s) := \max_{a \in A} \left\{ Q^t(s,a) \right\}$$

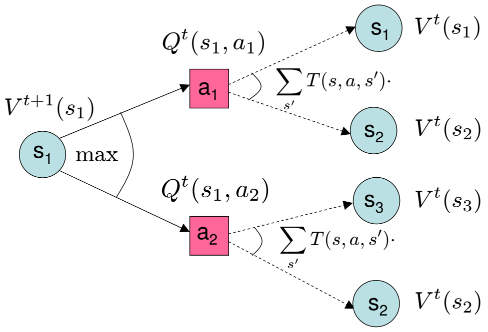
  - At ∞ horizon, get same value (=V*)

$$\lim_{t \to \infty} \max_s |V^t(s) - V^{t-1}(s)| = 0$$

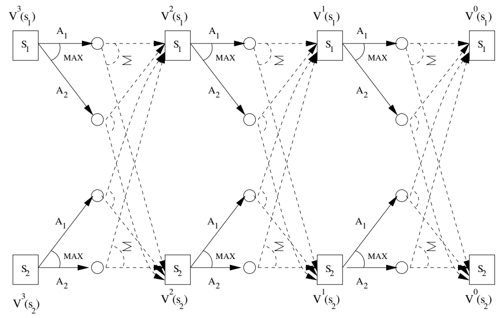
    - π^* says act same at each decision stage for ∞ horizon!

Single Dynamic Programming Step

- Graphical view:



Synchronous DP Updates (Value Iteration)



Value Function → Policy

- Can derive policy from value function V
- Given arbitrary value V (optimal or not)…
  - A *greedy policy* π_V takes action in each state that maximizes expected value w.r.t. *V*:

$$\pi_V(s) = \arg\max_a \left\{ R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \right\}$$

  - If can act so as to obtain *V* after doing action *a* in state *s*, π_V guarantees *V(s)* in expectation

How to Specify & Solve "First-order MDPs"?

Following:
[Boutilier, Reiter, Price, IJCAI-01]

First-order (FO)MDPs: Case Statement

- *<S,A,T,R>* for FOMDPs defined in terms of *cases*
  - E.g., express *reward* in *BoxWorld* FOMDP as…

rCase(s) =

$\forall b,c.\ Dest(b,c) \Rightarrow BIn(b,c,s)$	1
¬ "	0

- **Operators:** Define unary, binary case operations
  - E.g., can take "cross-sum" ⊕ (or ⊗, ⊖) of cases…

φ	10
¬φ	20

⊕

ϕ	3
¬ϕ	4

=

φ ∧ ϕ	13
φ ∧ ¬ϕ	14
¬φ ∧ ϕ	23
¬φ ∧ ¬ϕ	24

Stochastic Actions & FODTR

- Stochastic actions using deterministic SitCalc:
  - User's stochastic action: *A(x) = load(b,t)*
  - Nature's choice: *n(x)∈{loadS(b,t), loadF(b,t)}*
  - Probability distribution over Nature's choice:

P(loadS(b,t) | load(b,t)) =

snow(s)	.1
¬ snow(s)	.6

P(loadF(b,t) | load(b,t)) =

snow(s)	.9
¬ snow(s)	.4

- First-order decision-theoretic regression
  - FODTR = *expectation* of regression:

$$FODTR[vCase(s),A(x)] = \mathbf{E}_{P(n(x)|A(x))}[Regr[vCase(s),n(x)]]$$

Q-functions and Backups

- **FODTR almost gives us a Q-function**

$FODTR[vCase(unload(b,t))] =$

$On(b,t,s)$	5
$\neg On(b,t,s)$	0

  – FODTR specific to action variables
  – Also need to add reward, discount

- **Specify a backup operator for this**

$B^{unload}[vCase(s)] = rCase(s) \oplus \gamma$

$\exists b,t.\ On(b,t,s)$	5
$\exists b,t.\ \neg On(b,t,s)$	0

  – Yields a first-order Q-function

Symbolic Dynamic Programming

- What value if 0-stages-to-go?
  – Obviously $V^0(s) = rCase(s)$
- What value if 1-stage-to-go?
  – We know value for each action

$V^1(s) =$

φ_1	9
else φ_3	3
else φ_4	1
else φ_2	0

$= \max_S$

φ_1	9
φ_2	0

$= B^{A_1}[rCase(s)]$

φ_3	3
φ_4	1

$= B^{A_2}[rCase(s)]$

  – Now just need max for every state

- Value iteration: (BoutReiPr, IJCAI-01)
  – Obtain V^{n+1} from V^n until $(V^{n+1} \ominus V^n) < \varepsilon$

[Sanner, Thesis]

First-order ADDs

- **Want to compactly represent:**

case =

$\exists x.[A(x) \vee \forall y.A(x) \wedge B(x) \wedge \neg A(y)]$	1
$\neg$ "	0

- **Push down quantifiers, expose prop. structure:**

$[\exists x.A(x)] \vee ([\exists x.A(x) \wedge B(x)] \wedge [\forall y.\neg A(y)])$

Var	Var ⇔ FOL KB
a	$\equiv [\exists x.A(x)]$
b	$\equiv [\exists x.A(x) \wedge B(x)]$

case =

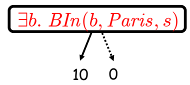
$a \vee (b \wedge \neg a)$	1
$\neg$ "	0

- **Convert to first-order ADD**

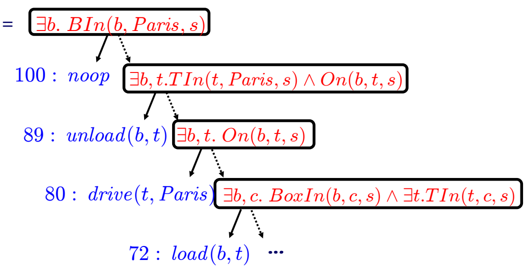
case =


Results for SDP with FOADDs

- Replace case with FO(A)ADDs, e.g. *BoxWorld*

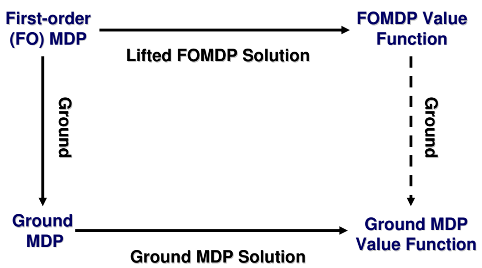
$rCase(s) =$


- Use FO(A)ADD ops for structured SDP (using γ=.9)…

$vCase(s) =$


Correctness of SDP

- **Show SDP for FOMDPs is correct w.r.t. ground MDP:**



Related Purely Deductive Approaches

- Value Iteration:
  – **ReBel algorithm**
    (Kersting, van Otterlo, de Raedt, ICML-04)
  – **FOVIA algorithm for fluent calculus**
    (Karabaev & Skvortsova, UAI-05)
  – **First-order decision diagrams (FODDs)**
    (Wang, Joshi, Khardon, IJCAI-07; JK, ICAPS-08; WJK, JAIR-08)

Kristian covers this.

Saket covers this.

- Approximate Linear Programming (ALP)
  – **First-order ALP (FOALP)**
    (Sanner & Boutilier, UAI-05)

3rd place in ICAPS IPPC5 (after FPG, FF-Replan)

- Policy Iteration
  – **Approximate policy iteration (FOAPI)**
    (Sanner & Boutilier, UAI-06)
  – **Modified policy iteration with FODDs**
    (Wang & Joshi, UAI-07)
- Factored FOMDPs – FOMDP extension
  – **Factored SDP and Factored FOALP**
    (Sanner & Boutilier, ICAPS-07)

Conclusions

- MDP: model of decision-theoretic planning
  - Common solution is dynamic programming
- "FOMDPs" are one model for lifted decision-theoretic planning
  - Use SitCalc specified action theory
  - Use case to represent reward, probabilities
  - Symbolic dynamic programming = lifted DP
  - State & action abstraction for MDPs & DP

ICAPS 2008 Tutorial

Techniques for First-order Planning

Relational Bellman Algorithm (ReBeL)

Kristian Kersting
Fraunhofer IAIS

• Thanks to Prasad Tadepalli, Alan Fern, Kurt Driessens, Martijn Van Otterlo,

---

Tutorial Outline

- **Motivation**
- **Deductive First-order Planning**
  - Situation Calculus
  - Symbolic Dynamic Programming
  - **Relational Bellman Algorithm (ReBeL)**
  - First-order Decision Diagrams (FODDs)
- Inductive First-order Planning
- Conclusion

[Kersting, Van Otterlo, De Raedt ICML04]

ReBeL

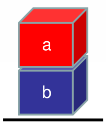
A relational Bellman algorithm

- Instance of SDP
- Sacrifices expressivity/compactness for „simplicity"
- Abstract state = existenially quantified conjunction of atoms (logical query) with equality constraints
- „Direct treatment" of probabilistic actions
- Basic data structure = decision lists

ReBeL's Abstract States

Definition 2 *An abstract state is a conjunction Z of logical atoms, i.e., a logical query.*

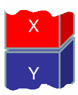
Abstract states represent sets of states. More formally, a state is an interpretation, i.e. a set of grounds facts.

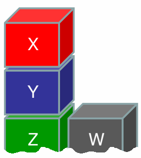


$\mathtt{on}(\mathtt{a},\mathtt{b}), \mathtt{on}(\mathtt{b},\mathtt{floor}), \mathtt{cl}(\mathtt{a})$

$\mathtt{on}(\mathtt{X},\mathtt{Y}), \mathtt{on}(\mathtt{Y},\mathtt{Z}), \mathtt{cl}(\mathtt{X}), \mathtt{cl}(\mathtt{W})$

$\mathtt{on}(\mathtt{X},\mathtt{Y})$





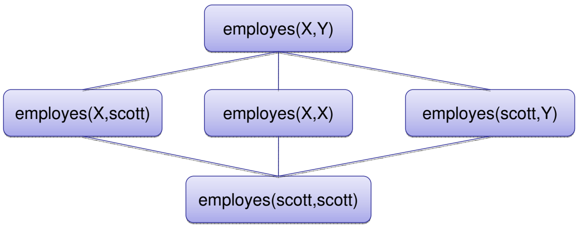
Aside: Subsumption

- Recall that a clause like
  C: grandparent(X,Z):- parent(X,Y), parent(Y,Z)
  is **C: {grandparent(X,Z), ~parent(X,Y), ~parent(Y,Z)}**

Clause **C1 subsume C2 iff** there exists a substitution θ *s.t.* ***θC1 is a subset of C2***

- Thus for the following pair of clauses, **C1** subsumes **C2**:
  C1: mem(A,[B|C]):- mem(A,C).
  C2: mem(0,[1,0]):- nat(0), nat(1), mem(0,[0]).
- **Note that C1 "looks" more general.**

Aside: Subsumption



- Subsumption induces a „generality“ lattice

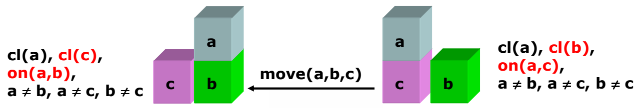
ReBeL's Abstract Actions

An action is an expression of the from $H \xleftarrow{p:A} B$

postcondition — Probability, action name, and action parameters — precondition

[Semantics]: *If current state b subsumed by B, i.e., $b \preceq_\theta B$, then taking action A will result in $[b \setminus B\theta] \cup H\theta$ with probability p, and b remains unchanged otherwise.*

on(X,Y), cl(X), cl(Z) $\xleftarrow{0.9:\text{move}(X,Y,Z)}$ cl(X), cl(Y), on(X,Z)
X ≠ Y, Y ≠ Z, X ≠ Z ... X ≠ Y, Y ≠ Z, X ≠ Z

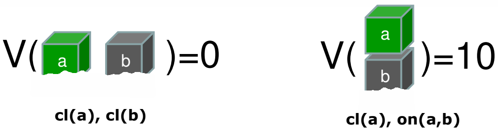


ReBeL's Reward Model

Definition 3 *An abstract state value function V_t is a finite list of value rules of the form $v(c) \leftarrow B$ were B is an abstract state and $c \in \mathbb{R}$ is the value of all states subsumed by B.*

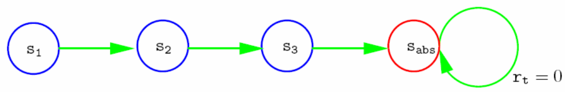
$$v(10) \leftarrow \texttt{on(a,b)}.$$
$$v(0) \leftarrow \texttt{true}.$$

For abstract state Z, $V_t(Z) = \max\{c \mid v(c) \leftarrow B \in V_t \text{ and } Z \preceq_\theta B\}$.



State-action values are modeled by Q-rules: $q(c) \leftarrow B, A$

Absorbing States



- In RL, episodic tasks can be encoded using absorbing states
  - transition only to themselves
  - generate zero reward
- In ReBeL, we encode absorbing states using artificial deterministic functions

$$\text{on(a,b)} \xleftarrow{1.0:\text{absorbing}} \text{on(a,b)}.$$

Integrity Constraints …

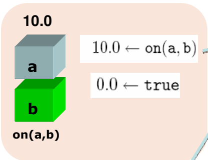
… are „simply“ Horn clauses

$$\texttt{false} \leftarrow \texttt{on(X,Y)}, \texttt{cl(Y)}$$
$$\texttt{X} \neq \texttt{Y} \leftarrow \texttt{on(X,Y)}$$

Note that actions are constraint, too. They cannot yield illegal states.

- We used Frühwirth's *Constraint Handling Rules* and Buntine's *Generalized Subsumption* (i.e., subsumption w.r.t. to a background theory)

Summary of ReBel's MDPs



$\texttt{cl(a)}, \texttt{cl(b)}, \texttt{on(a,c)}$

- **States: interpretations**, i.e., set of ground atoms
- **Abstract states**: **conjunction** of atoms („query“)
- **Actions**: each outcome a probabilistic STRIP rule
- **Abstract value functions**: set of rules of the form $v(c) \leftarrow B$ where c is a value and B an abstract state
- **Reward Function**: initial value function $\mathbf{V_0}$
- **Integrity Constraints:** : horn clauses

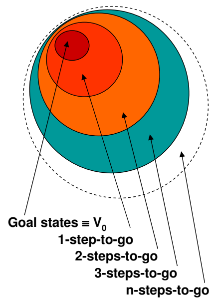
on(X,Y), cl(X), cl(Z) $\xleftarrow{0.9:\text{move}(X,Y,Z)}$ cl(X), cl(Y), on(X,Z)
X ≠ Y, Y ≠ Z, X ≠ Z ... X ≠ Y, Y ≠ Z, X ≠ Z

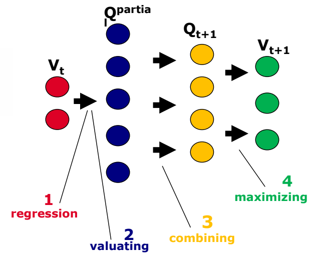
postcondition — Action name and parameters — precondition

Theorem:
Every ReBeL MDP induces a (possibly infinte) MDP

Recap: SDP

Given an initial, abstract reward model $\mathbf{R}(\equiv V_0)$,
Compute the t-steps-to-go abstract state value functions V_t, $t = 1, 2, \ldots$



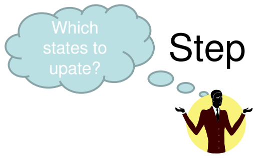


Value Iteration
repeat for each t
for each state do

$$V_{t+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_t(s')]$$

until 'convergence'

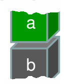
Step 1: Regression



on(X,Y), cl(X), cl(Z) X ≠ Y, Y ≠ Z, X ≠ Z $\xleftarrow{0.9:\text{move}(X,Y)}$ cl(X), cl(Y), on(X,Z) X ≠ Y, Y ≠ Z, X ≠ Z

- Abstract actions define how states change
- In order to do the update from staet Z, we need to consider all states that can reach Z by applying the action
- Intuition: ‚Revers' the action effects
- Compute the weakest precondition of Z given the action

on(a,b)



Step 1: Regression

on(X,Y), cl(X), cl(Z) X ≠ Y, Y ≠ Z, X ≠ Z $\xleftarrow{0.9:\text{move}(X,Y)}$ cl(X), cl(Y), on(X,Z) X ≠ Y, Y ≠ Z, X ≠ Z

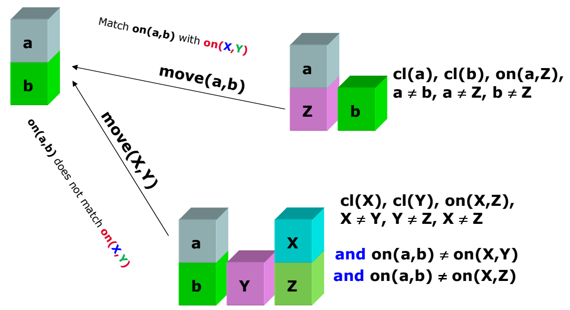
Two cases:

1. move caused on(a,b): We have been in abstract state
$$S_1 \equiv (\mathtt{cl}(\mathrm{a}), \mathtt{cl}(\mathrm{b}), \mathrm{on}(\mathrm{a}, \mathrm{Z}), \mathrm{a} \neq \mathrm{b}, \mathrm{b} \neq \mathrm{Z}, \mathrm{a} \neq \mathrm{Z}.$$
2. move did not couse on(a,b): We moved X on Y but not a on b. So, we have been in abstract state
$$T \equiv (\mathtt{cl}(\mathrm{X}), \mathtt{cl}(\mathrm{Y}), \mathrm{on}(\mathrm{X}, \mathrm{Z}), \mathrm{on}(\mathrm{a}, \mathrm{b}), \mathrm{X} \neq \mathrm{Y}, \mathrm{Y} \neq \mathrm{Z}, \mathrm{X} \neq \mathrm{Z})$$
satisfying $c \equiv (\mathrm{on}(\mathrm{X}, \mathrm{Y}) \neq \mathrm{on}(\mathrm{a}, \mathrm{b}) \wedge \mathrm{on}(\mathrm{X}, \mathrm{Z}) \neq \mathrm{on}(\mathrm{a}, \mathrm{b}))$
$\equiv (\mathrm{X} \neq \mathrm{a} \vee \mathrm{Y} \neq \mathrm{b} \vee Z \neq b)$.

which guarantees that applying move(X,Y) will not affect on(a,b).
Simplifies to
$$S_2 \equiv (T \wedge \mathrm{X} \neq \mathrm{a}),\ S_3 \equiv (T \wedge \mathrm{Y} \neq \mathrm{b}) \text{ and } S_4 \equiv (T \wedge \mathrm{Z} \neq \mathrm{b}).$$

Step 1: Regression

on(X,Y), cl(X), cl(Z) X ≠ Y, Y ≠ Z, X ≠ Z $\xleftarrow{0.9:\text{move}(X,Y)}$ cl(X), cl(Y), on(X,Z) X ≠ Y, Y ≠ Z, X ≠ Z



Step 1: Regression

- on(a,b) is simple effect. In general, we have multiple effects, i.e., conjuctions of atoms.

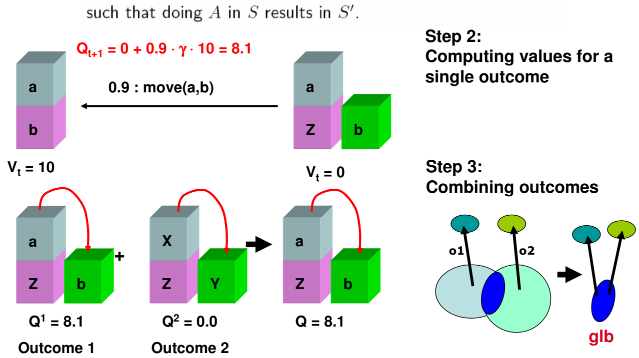
If $Post \xleftarrow{A} Pre$ and the 'next' state is Z
we take all possible overlapping subsets of $Post$ and Z
apply the resulting substitution θ on $Post$ and Pre
add 'unexplained' effects in Z to $Pre\theta$
calculate and add constraints

$\mathrm{on}(\mathrm{a}, \mathrm{b}), \mathrm{on}(\mathrm{c}, \mathrm{d})$: $\mathrm{on}(\mathrm{a}, \mathrm{b})$ was the effect and $on(c,d)$ was already true
$\mathrm{on}(\mathrm{a}, \mathrm{b}), \mathrm{on}(\mathrm{c}, \mathrm{d})$: $\mathrm{on}(\mathrm{c}, \mathrm{d})$ was the effect and $on(a,b)$ was already true
$\mathrm{on}(\mathrm{a}, \mathrm{b}), \mathrm{on}(\mathrm{c}, \mathrm{d})$: $\mathrm{on}(\mathrm{X}, \mathrm{Y})$ was the effect and $on(a,b), on(c,d)$ were already true
and constraints make sure that $\mathrm{on}(\mathrm{X}, \mathrm{Y})$ is not $\mathrm{on}(\mathrm{a}, \mathrm{b})$ nor $\mathrm{on}(\mathrm{c}, \mathrm{d})$.

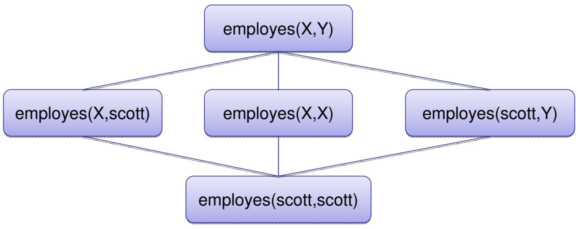
Steps 2&3: Valuation & Combination …

… that means computing the Q rules

From regression, for each state S' in V_t we obtain (S, A)-pairs such that doing A in S results in S'.



Aside: Greatest Lower Bound (GLB)



- Recall that subsuption induces a lattice (over the reduced clauses). Hence, for each node, we can compute the greatest lower bound.
- GLB: **set of ground states, in which both abstract states hold**.

Q-Rules after first „Iteration“

$$
\begin{array}{llll}
q(10) & \leftarrow & \mathtt{on(a,b), a \neq b, absorb.} & \langle 1 \rangle \\
q(10) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z), on(a,b), X \neq Y,} & \\
 & & \mathtt{Y \neq Z, X \neq Z, X \neq a, a \neq b, move(X,Y).} & \langle 2 \rangle \\
q(10) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z), on(a,b), X \neq Y,} & \\
 & & \mathtt{Y \neq Z, X \neq Z, Y \neq b, a \neq b, move(X,Y).} & \langle 3 \rangle \\
q(10) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z), on(a,b), X \neq Y} & \\
 & & \mathtt{Y \neq Z, X \neq Z, a \neq b, Z \neq b, move(X,Y).} & \langle 4 \rangle \\
q(9) & \leftarrow & \mathtt{cl(a), cl(b), on(a,Z),} & \\
 & & \mathtt{a \neq b, b \neq Z, a \neq Z, move(a,b)} & \langle 5 \rangle \\
q(0) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z),} & \\
 & & \mathtt{X \neq Y, Y \neq Z, X \neq Z, move(X,Y).} & \langle 6 \rangle
\end{array}
$$

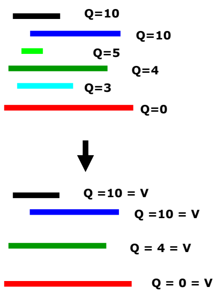
Step 4: Maximizing …

… that means computing the next value function by maximizing the Q-rules

$V_{t+1}(s) = \max_a Q_{t+1}(s,a)$

10 : move(X,Y) ← cl(X), cl(Y)
5 : move(a,b) ← cl(a), cl(b)

- Sort the Q-rules in descending order
- Remove top $q(d) \leftarrow B, A$ from the rules
- **if** no other more general rule
- **then** add $v(d) \leftarrow B$ to V_{t+1} and remove all Qrules that are subsumed
- continue until no more rules



Step 4: Maximizing …

$$
\begin{array}{llll}
q(10) & \leftarrow & \mathtt{on(a,b), a \neq b, absorb.} & \langle 1 \rangle \\
q(10) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z), on(a,b), X \neq Y,} & \\
 & & \mathtt{Y \neq Z, X \neq Z, X \neq a, a \neq b, move(X,Y).} & \langle 2 \rangle \\
q(10) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z), on(a,b), X \neq Y,} & \\
 & & \mathtt{Y \neq Z, X \neq Z, Y \neq b, a \neq b, move(X,Y).} & \langle 3 \rangle \\
q(10) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z), on(a,b), X \neq Y} & \\
 & & \mathtt{Y \neq Z, X \neq Z, a \neq b, Z \neq b, move(X,Y).} & \langle 4 \rangle \\
q(9) & \leftarrow & \mathtt{cl(a), cl(b), on(a,Z),} & \\
 & & \mathtt{a \neq b, b \neq Z, a \neq Z, move(a,b)} & \langle 5 \rangle \\
q(0) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z),} & \\
 & & \mathtt{X \neq Y, Y \neq Z, X \neq Z, move(X,Y).} & \langle 6 \rangle
\end{array}
$$

Step 4: Maximizing …

$$
\begin{array}{llll}
q(9) & \leftarrow & \mathtt{cl(a), cl(b), on(a,Z),} & \\
 & & \mathtt{a \neq b, b \neq Z, a \neq Z, move(a,b)} & \langle 5 \rangle \\
q(0) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z),} & \\
 & & \mathtt{X \neq Y, Y \neq Z, X \neq Z, move(X,Y).} & \langle 6 \rangle
\end{array}
$$

Step 4: Maximizing …

$$
\begin{array}{llll}
q(0) & \leftarrow & \mathtt{cl(X), cl(Y), on(X,Z),} & \\
 & & \mathtt{X \neq Y, Y \neq Z, X \neq Z, move(X,Y).} & \langle 6 \rangle
\end{array}
$$

Finally, we get ...

$$V_{t+1}$$

$$
\begin{array}{rcl}
v(10) & \leftarrow & \mathtt{on(a,b), a \neq b.} \\
v(9) & \leftarrow & \mathtt{on(a,Y), cl(b), cl(a), b \neq Y, a \neq Y, a \neq b.} \\
v(0) & \leftarrow & \mathtt{on(X,Y), cl(X), cl(Z), Z \neq Y, X \neq Y, X \neq Z.}
\end{array}
$$

Blocks World

Predicates: $\mathtt{on}/2$ and $\mathtt{cl}/1$

$$\mathtt{on(X,Y), cl(X), cl(Z),}\ \mathtt{X \neq Y, Y \neq Z, X \neq Z} \xleftarrow{1.0:\mathtt{move(X,Y)}} \mathtt{cl(Y), cl(X), on(X,Z),}\ \mathtt{X \neq Y, Y \neq Z, X \neq Z}$$

Note that the number of blocks is unspecified

Value function, V_0 e.g.:

$$
\begin{array}{rcl}
v(10) & \leftarrow & \mathtt{cl(a).} \\
v(0) & \leftarrow & \mathtt{true.}
\end{array}
$$

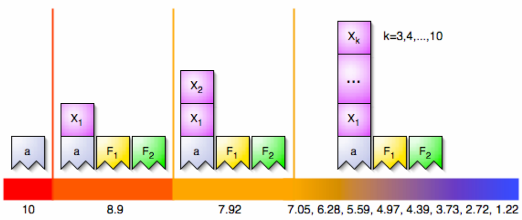
IC & sample constraint

$$\mathtt{cl(a)} \xleftarrow{1.0:\mathrm{absorbing}} \mathtt{cl(a).}$$

$$\mathtt{X \neq Y} \leftarrow \mathtt{on(X,Y).}$$

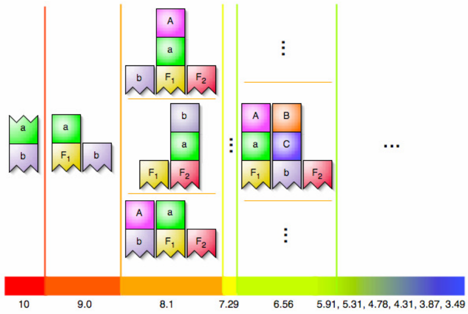
In the experiments, the discount $\gamma = 0.9$.

Blocks World: cl(a)



Observation 1 *Abstraction does not guarantee convergence in infinite domains because an infinite number of abstract states can be required.*

Blocks World: on(a,b)



ReBel: Logistics Domain

$$\mathtt{bin(B,C), tin(T,C)}, R \xleftarrow{pr:\mathtt{unload(B,T)}} \mathtt{on(B,T), tin(T,C)}, R$$

$$\mathtt{on(B,T), tin(T,C)}, R \xleftarrow{pr:\mathtt{load(B,T)}} \mathtt{bin(B,C), tin(T,C)}, R$$

$$\mathtt{tin(T,C'), C \neq C'} \xleftarrow{1.0:\mathtt{drive(T,C')}} \mathtt{tin(T,C)}$$

(pr is 0.9 if R is $\mathtt{rain}$ and 0.7 if R is $\mathtt{not\_rain}$)

Reward model

$$
\begin{array}{rcl}
v(10) & \leftarrow & \mathtt{bin(b,p)} \\
v(0) & \leftarrow & \mathtt{true.}
\end{array}
$$

IC and absorbing

$$\mathtt{false} \leftarrow \mathtt{rain, not\_rain}$$

$$\mathtt{bin(b,p)} \xleftarrow{1:\mathrm{absorbing}} \mathtt{bin(b,p)}$$

$$\gamma = 0.9$$

ReBel: Logistics Domain

	V_t									
abstract states	1	2	3	4	5	6	7	8	9	10
bin(b, p).	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000
tin(A, p), on(b, A), not_rain.	8.100	8.829	8.895	8.901	8.901	8.901	8.901	8.901	8.901	8.901
tin(A, p), on(b, A), rain.	6.300	8.001	8.460	8.584	8.618	8.627	8.629	8.630	8.630	8.630
tin(A, B), on(b, A), not_rain.		7.290	7.946	8.005	8.010	8.011	8.011	8.011	8.011	8.011
tin(A, B), on(b, A), rain.		5.670	7.201	7.614	7.726	7.756	7.764	7.766	7.767	7.767
tin(A, B), bin(b, B), not_rain.			5.905	6.968	7.111	7.128	7.130	7.131	7.131	7.131
tin(A, B), bin(b, B), rain.			3.572	5.501	6.282	6.563	6.658	6.689	6.699	6.702
tin(A, B), bin(b, C), not_rain.				5.314	6.271	6.400	6.416	6.417	6.418	6.418
tin(A, B), bin(b, C), rain.				3.215	4.951	5.654	5.907	5.993	6.020	6.029
tin(A, B).	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Convergence on both structural and value level.

(In about 2 min.)

Conclusions

- ReBeL is an instance of SDP
  - Avoids the full state and action enumeration of classical approaches
  - Lifted solution applies to any (ground) instance
  - Basic tool: Constraint-Logic Programming
- Sacrifices expressivity/compactness for „simplicity“
- Employs constraint logic programming
- Background Knowledge is not a feature, but a necessity
- Convergence: Structural and Value level

ICAPS 2008 Tutorial

Techniques for First-order Planning

First-order Decision Diagrams (FODDs)

Saket Joshi
Tufts University

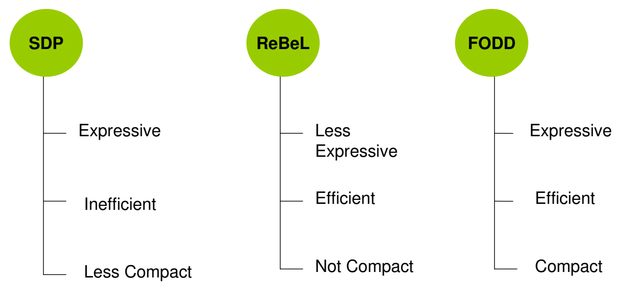
• Thanks to Chenggang Wang and Roni Khardon

Tutorial Outline

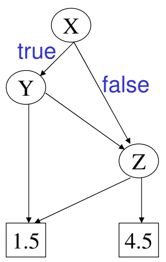
- **Motivation**
- **Deductive First-order Planning**
  - Situation Calculus
  - Symbolic Dynamic Programming
  - Relational Bellman Algorithm (ReBeL)
  - **First-order Decision Diagrams (FODDs)**
- Inductive First-order Planning
- Conclusion

[Wang, Joshi, Khardon, IJCAI 07, JAIR08]

Motivation for FODDs

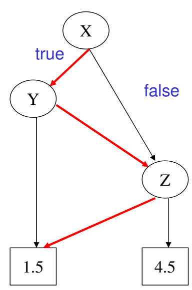


Propositional ADDs - Syntax

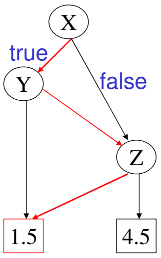


Propositional ADDs - Semantics

- Variable valuation
  - {X = true, Y = false, Z= true}
- Function output: 1.5



Propositional ADDs – Normal Form



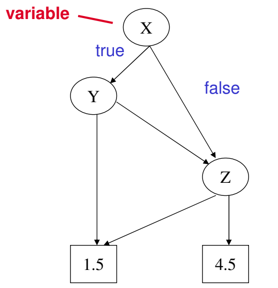
Propositional ADD

For a given variable ordering, every function has a unique representation.

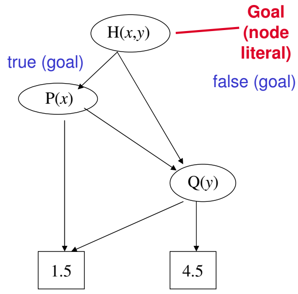
But, how do we deal with logical atoms ?

First-Order Decision Diagrams

Idea: nodes are existentially quantified atoms (goals)



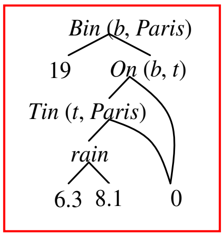
• Propositional ADD



• First-order decision diagram

First-Order Decision Diagrams (FODDs) – Semantics based on Single Paths

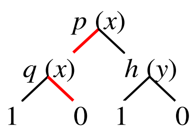
Same as for decision trees and their relational variants such as TILDE [Blockeel, De Raedt 98].



1. $\exists b, Bin(b, Paris) : 19$
2. $\neg\exists b, Bin(b, Paris) \wedge$
   $\exists b, t, On(b,t) \wedge Tin(t, Paris) \wedge rain : 6.3$
3. $\neg\exists b, Bin(b, Paris) \wedge$
   $\exists b, t, On(b,t) \wedge Tin(t, Paris) \wedge \neg rain : 8.1$
4. Otherwise $: 0$

First Order Decision Diagrams (FODDs) – Semantics based on Multiple Paths

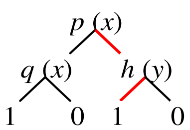
Semantics defined in terms of variable valuations $MAP_B(I, \zeta)$



- Domain: {1, 2, 3}
- Interpretation I: {p(1), q(2), h(3)}
- ζ_1 = { x/1, y/1}
- $MAP_B(I, \zeta_1) = 0$

First Order Decision Diagrams (FODDs) – Semantics based on Multiple Paths

Semantics defined in terms of variable valuations $MAP_B(I, \zeta)$



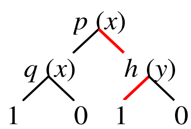
- Domain: {1, 2, 3}
- Interpretation I: {p(1), q(2), h(3)}
- ζ_1 = { x/2, y/3}
- $MAP_B(I, \zeta_1) = 1$

First Order Decision Diagrams (FODDs) – Semantics based on Multiple Paths

Semantics defined in terms of variable valuations $MAP_B(I, \zeta)$

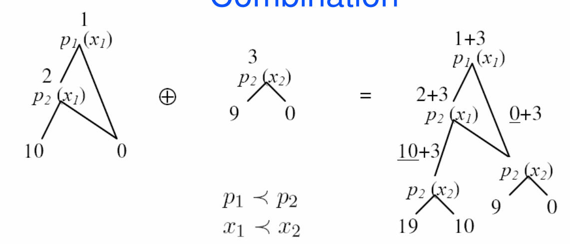
$MAP_B(I) = \max_{\xi}\{MAP_B(I, \xi)\}$

- take the maximum over all paths -



- Domain: {1, 2, 3}
- Interpretation I: {p(1), q(2), h(3)}
- $MAP_B(I) = 1$

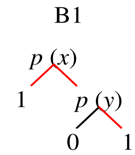
First Order Decision Diagrams (FODDs) – Combination



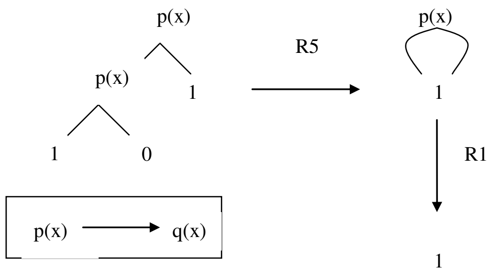
- Assume a fixed order among predicates and parameters
- Choose lower label as new root and combine sub-diagrams recursively.
- Stop when combining two leaves; perform numerical operation

First Order Decision Diagrams (FODDs) – Reductions

Is this the most compact representation?

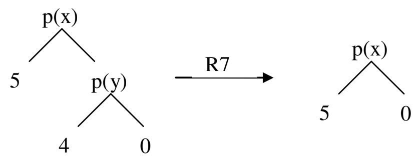


Strong Reduction



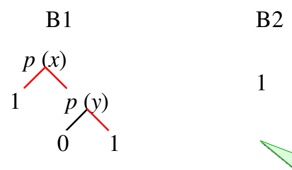
Background Knowledge

Weak Reduction



Under any interpretation, for every valuation reaching value 4, there is another valuation reaching 5

First Order Decision Diagrams (FODDs) – Reductions



No!!! Several reduction operators have been developed

Reductions are key to practical FODDs. The rest is SDP

$MAP_{B1}(I) = MAP_{B2}(I)$ for any I

Value Iteration using FODDs

- Instance of SDP

$$FODTR[vCase, A(\vec{x})] = rCase \oplus \gamma\, [\oplus_j \{pCase(n_j(\vec{x})) \otimes Regr[vCase, A(\vec{x})]\}]$$

$$qCase^t(A(\vec{x})) = \max \exists \vec{x}.\, FODTR[vCase^{t-1}, A(\vec{x})]$$

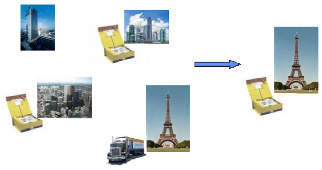
$$vCase^t = \max_{a \in \{A_1(\vec{x}_1), \ldots, A_m(\vec{x}_m)\}} qCase^t(a)$$

- Let's go through an example

FODDs for MDPs - Logistics Domain Revisited

- Predicates

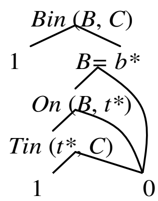
Bin (Box, City),
Tin (Truck, City),
On (Box, Truck)



- Actions

Load(box,truck) {LoadS, LoadF}
Unload(box,truck) {UnloadS, UnloadF}
Drive(truck,city)

Encoding the Domain Dynamics - Truth value diagrams (TVDs)

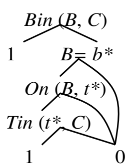


The TVD for Bin(B, C) under unloadS(b^*, t^*)

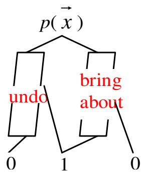
- For every action A schema and predicate schema P, a TVD is a FODD with {0,1} leaves
- Gives the truth value of the predicate P in the next state when A is executed in the current state.
- It captures the truth values for all instances of P
- Only the variables in A and P can appear in the corresponding TVD
- Can express universal effects

Truth value diagrams (TVDs)

predicate is true
if it **was true before** and is not **undone** by the action
or **was false before** and is **brought about** by the action



The TVD for Bin(B, C) under unloadS(b^*, t^*)

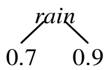


TVD template

Multi-path semantics is beneficial. With single path semantics, a TVD would have to specify all possible ways a predicate can become true

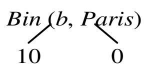
Encoding Nature's Choice and Reward Function

- Action choice probabilities



Probability of UnloadS being chosen given Unload is executed.

- Reward and value functions



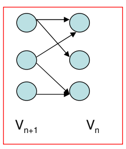
Value Iteration with FODDs

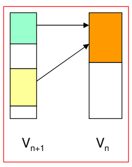
- The value iteration algorithm

$$V_{n+1}(s) = \max_{a \in A}[r(s) + \gamma \sum_{s' \in S} \Pr(s'|s,a)V_n(s')]$$

- The first-order value iteration

1. $$T_{n+1}^{A(\vec{x})}(V_n) = \oplus_j (prob(A_j(\vec{x})) \otimes \mathrm{Regr}(V_n, A_j(\vec{x})))$$

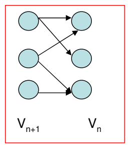




Value Iteration with FODDs

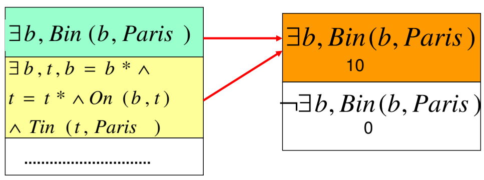
- The value iteration algorithm

$$V_{n+1}(s) = \max_{a \in A}[r(s) + \gamma \sum_{s' \in S} \Pr(s'|s,a)V_n(s')]$$



- The first-order value iteration

1. $$T_{n+1}^{A(\vec{x})}(V_n) = \oplus_j (prob(A_j(\vec{x})) \otimes \mathrm{Regr}(V_n, A_j(\vec{x})))$$



UnloadS

Value Iteration with FODDs

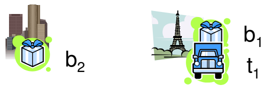
- The value iteration algorithm

$$V_{n+1}(s) = \max_{a \in A}[r(s) + \gamma \sum_{s' \in S} \Pr(s' | s, a) V_n(s')]$$

- The first-order value iteration

1. $T_{n+1}^{A(\vec{x})}(V_n) = \oplus_j (prob(A_j(\vec{x})) \otimes \mathrm{Regr}(V_n, A_j(\vec{x})))$

2. $Q_{n+1}^{A} = R \oplus \gamma \otimes \mathrm{obj-max}(T_{n+1}^{A(\vec{x})}(V_n))$

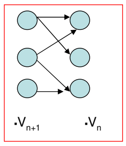


- Unload(b_1,t_1)
- Unload(b_2,t_1)

Value Iteration with FODDs

- The value iteration algorithm

$$V_{n+1}(s) = \max_{a \in A}[r(s) + \gamma \sum_{s' \in S} \Pr(s' | s, a) V_n(s')]$$

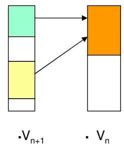


- The first-order value iteration

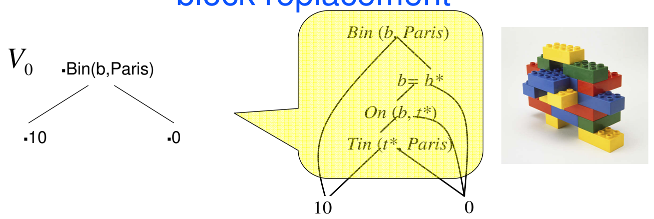
1. $T_{n+1}^{A(\vec{x})}(V_n) = \oplus_j (prob(A_j(\vec{x})) \otimes \mathrm{Regr}(V_n, A_j(\vec{x})))$

2. $Q_{n+1}^{A} = R \oplus \gamma \otimes \mathrm{obj-max}(T_{n+1}^{A(\vec{x})}(V_n))$

3. $V_{n+1} = \max_A Q_{n+1}^{A}$



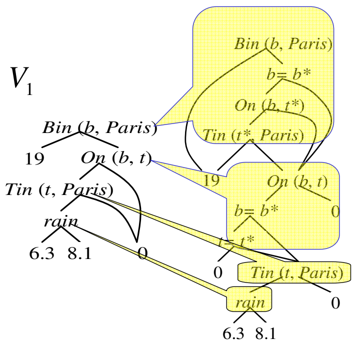
Value Iteration with FODDs - Regression by block replacement



$T_{n+1}^{A(\vec{x})}(V_n) = \oplus_j (prob(A_j(\vec{x})) \otimes \mathrm{Regr}(V_n, A_j(\vec{x})))$

Replace each node with the corresponding TVD, with the outgoing edges connected to 0 and 1 leaves of the TVD

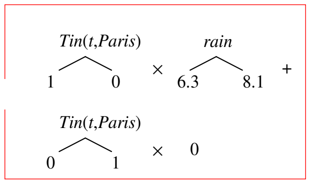
Value Iteration with FODDs - Regression by block replacement



Block combination:

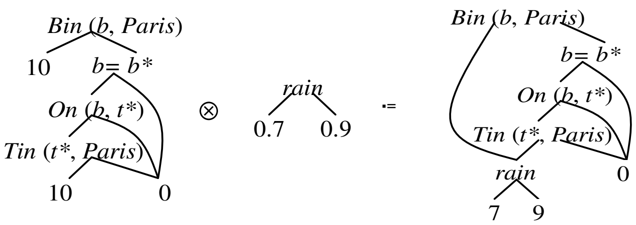
Parent B, the true branch child B_t, the false branch child B_f

$[B_n \otimes B_t] \oplus [(1 \ominus B_n) \otimes B_f]$



Value Iteration with FODDs - Adding regression results

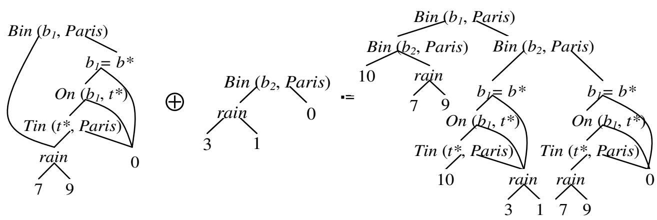
Each regression result multiplied with the corresponding choice probability



$T_{n+1}^{A(\vec{x})}(V_n) = \oplus_j (prob(A_j(\vec{x})) \otimes \mathrm{Regr}(V_n, A_j(\vec{x})))$

Value Iteration with FODDs - Adding regression results

Standardize apart different regression results before adding

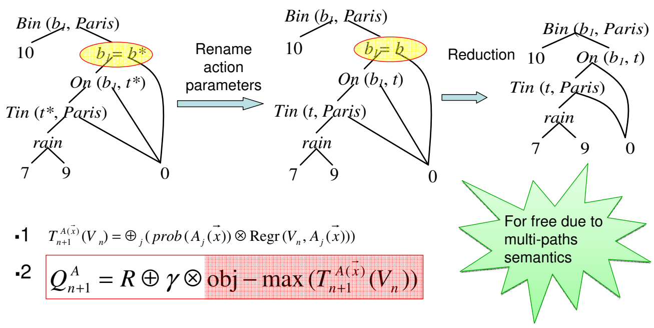


$T_{n+1}^{A(\vec{x})}(V_n) = \oplus_j (prob(A_j(\vec{x})) \otimes \mathrm{Regr}(V_n, A_j(\vec{x})))$

- Finally, reduce the resulting FODD

Value Iteration with FODDs - Object maximization

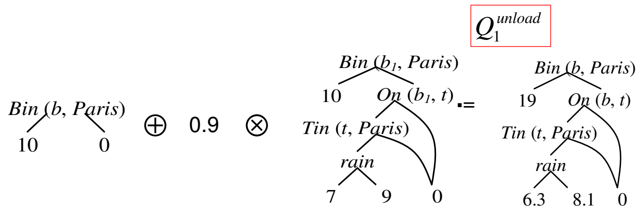
Maximizing over the action parameters to get the maximum value achievable by an instance of this action



•1 $T_{n+1}^{A(\vec{x})}(V_n) = \oplus_j (prob(A_j(\vec{x})) \otimes \mathrm{Regr}(V_n, A_j(\vec{x})))$

•2 $\boxed{Q_{n+1}^A = R \oplus \gamma \otimes \text{obj} - \max (T_{n+1}^{A(\vec{x})}(V_n))}$

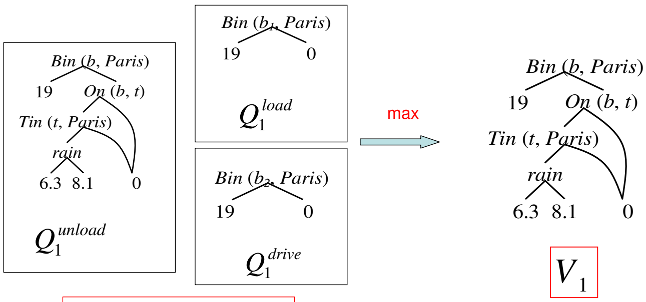
Value Iteration with FODDs



•1 $T_{n+1}^{A(\vec{x})}(V_n) = \oplus_j (prob(A_j(\vec{x})) \otimes \mathrm{Regr}(V_n, A_j(\vec{x})))$

•2 $\boxed{Q_{n+1}^A = R \oplus \gamma \otimes \text{obj} - \max(T_{n+1}^{A(\vec{x})}(V_n))}$

Value Iteration with FODDs - Maximizing over actions



3 $\boxed{V_{n+1} = \max_A Q_{n+1}^A}$

ICAPS IPPC Results

Logistis:

– Without approx.

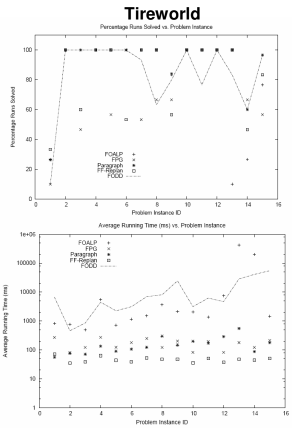
  • slower than ReBel

– with approx.

  • comparable

	Coverage	Time (ms)	Reward
GPT	100%	2220	57.66
Policy Iteration with policy language bias	46.66%	60466	36
Re-Engg NMRDPP	10%	290830	-387.7
FODD-Planner	100%	65000	47.3

Fileworld main results



FODDs for Relational MDPs - Summary

- FODDs compactly represent functions such as truth values, Q-values etc. over logical spaces
- Complete set of operators to reduce, multiply, add, etc. FODDs
  - direct implementation of SDP

FODDs for Relational MDPs - Summary

- Approximation a la SPUDD possible: merge …
  – substructures with similar values
  – Leaves, which are within a certain distance,
  – ….
- Policy iteration approach exists
  – Does not implement the same algorithm as original PI; instead it incorporates an element of policy improvement
  – Theorem: the sequence of value functions obtained from relational modified policy iteration converges monotonically to the optimal value function.
- Initial approach on partially observed, relational MDPs

ICAPS 2008 Tutorial

Techniques for First-order Planning

Inductive First-Order Planning

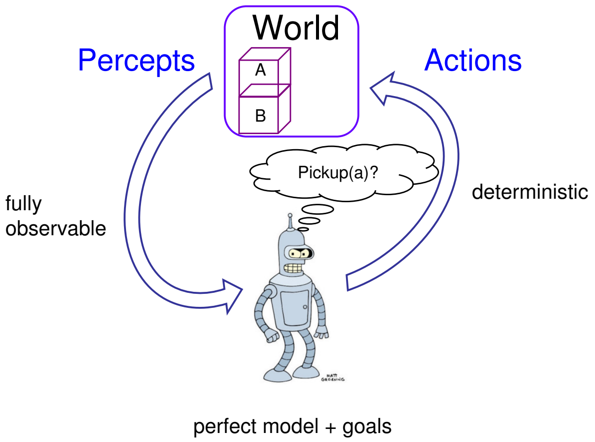
Kristian Kersting
Fraunhofer IAIS

• Thanks to Prasad Tadepalli, Alan Fern, Kurt Driessens, Martijn Van Otterlo,

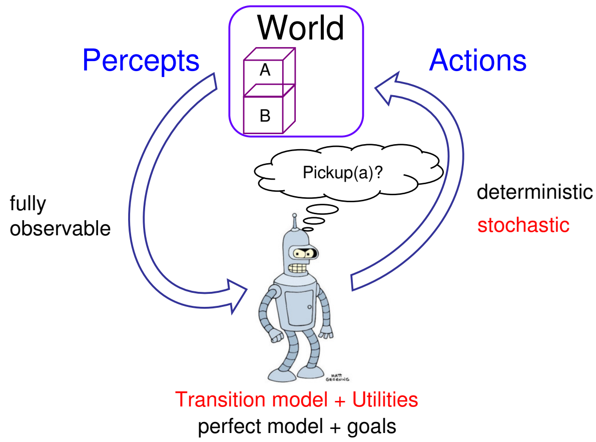
Tutorial Outline

- **Motivation**
- Deductive First-order Planning
  - Situation Calculus
  - Symbolic Dynamic Programming
  - Relational Bellman Algorithm (ReBeL)
  - First-order Decision Diagrams (FODDs)
- **Inductive First-order Planning**
- Conclusion

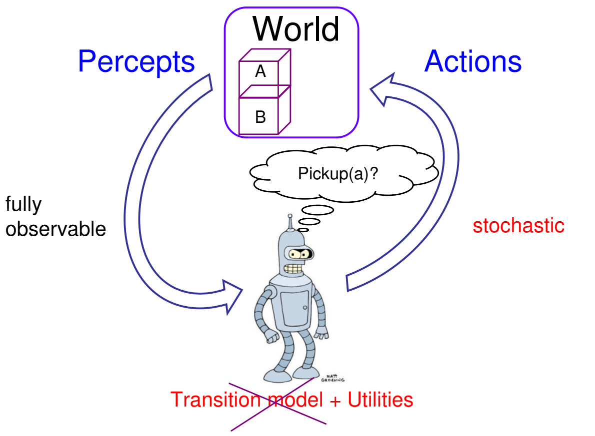
So far: Classical Planning …



… and relational MDPs



Now: Reinforcement Learning



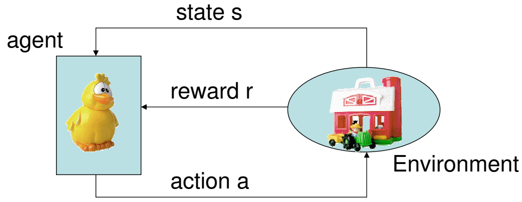
Reinforcement Learning

- **No knowledge of environment**
  - Can only act in the world and observe states and reward
- Situated agent: learner **must decide** what actions to take at each step
- Must solve **credit**/blame **assignment** to actions
  - What actions are responsible for success?
- **Tradeoff** between **exploiting** what is known vs. **exploring** something new
- Must act with limited amount of reasoning

Model-Based vs. Model-Free RL

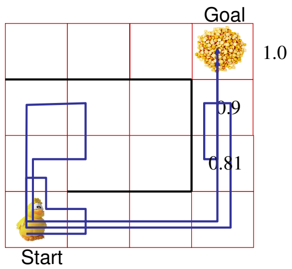
- **Model-based approach to RL:**
  - learn the MDP model, or an approximation of it
  - use it for policy evaluation or to find the optimal policy
  - can use value iteration or policy iteration treating the learned model as if it is correct
  - can do sample-based update of the value function
- **Model-free approach to RL:**
  - derive the optimal policy without explicitly learning the model
  - learn an action-based value function or Q-function
  - we will focus on this in this tutorial !!!

Reinforcement Learning



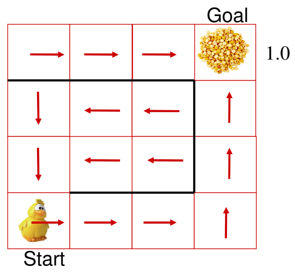
Agent's goal: Choose actions to maximize total reward

Reinforcement Learning



… and so on …

Reinforcement Learning



Learn a **policy mapping states** to optimal **actions**

Value Function

Compute

$$V_{i+1}(s) = \sum_a \pi(s,a) \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V_i(s'))$$

- Prefer actions such that the maximal expected one-step ahead value.
- Could also be done using Q-Values, i.e, values assigned to state action pairs (e.g. Q-Learning)

Goal

0.729	0.81	0.9	1.0
0.431	0.387	0.349	0.9
0.478	0.431	0.387	0.81
0.531	0.591	0.656	0.729

Start

Q-Learning: Model-Free RL

(1) Learn the optimal Q function.

(2) Act greedily with respect to Q(s,a).

- Q(s,a) is the expected value of taking action a in state s and then following the optimal policy thereafter.

- Optimal Q-function satisfies $V(s) = \max_{a'} Q(s,a')$
  s.t. $Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s')$
  $$= R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$
- After taking action a in state s and reaching s':
  $$Q(s,a) \leftarrow Q(s,a) + \alpha(\underbrace{R(s,a) + \gamma \max_{a'} Q(s',a')}_{\text{(noisy) sample of Q-value based on next state}} - Q(s,a))$$

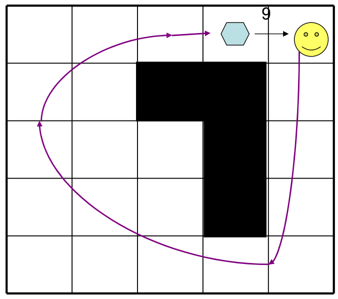
Q-Learning

1. Start with initial Q-function (e.g. all zeros)
2. Take action according to an explore/exploit policy (should converge to greedy policy)
3. Perform update
   $$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$
   Q(s,a) is current estimate of optimal Q-function.
4. Goto 2

- Does not require model since we learn Q directly!
- Uses **explicit |S|x|A| table** to represent Q
- Explore/exploit policy directly uses Q-values
  - E.g. use ε-greedy or Boltzmann exploration.

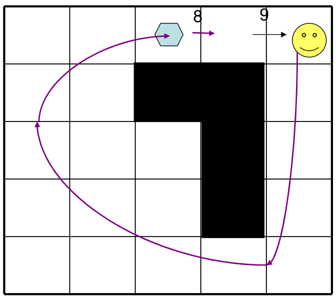
A Grid Example

Rewards:
10 for reaching the goal state
-1 for every action.
α is set to 1 for simplicity.
Update*: Q(s,a) = r +Max$_b$ (Q(s',b))*



A Grid Example

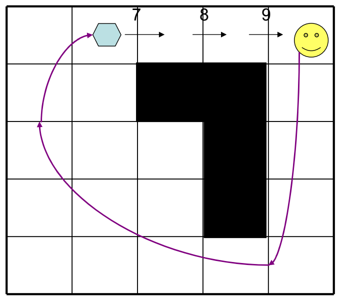
Rewards:
10 for reaching the goal state
-1 for every action.
α is set to 1 for simplicity.
Update*: Q(s,a) = r +Max$_b$ (Q(s',b))*



A Grid Example

Choose an action *a = argmax$_a$ Q(s,a)* reaching *s'*

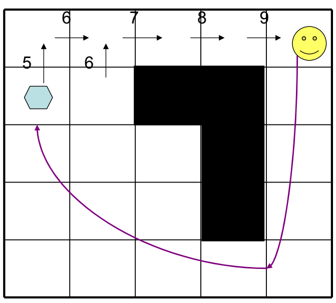
Update *Q(s,a) = r + Max$_b$ Q(s',b)*



A Grid Example

Choose an action *a = argmax$_a$ Q(s,a)* reaching *s'*

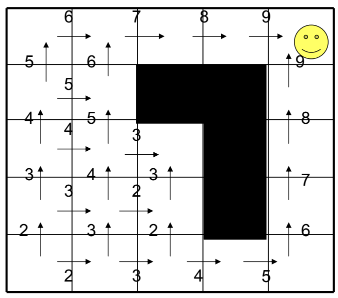
Update *Q(s,a) = r + Max$_b$ Q(s',b)*



A Grid Example

Choose an action *a = argmax$_a$ Q(s,a)* reaching *s'*

Update *Q(s,a) = r + Max$_b$ Q(s',b)*



The values converge to the optimal Q-values under GLIE policy

Large-Scale Problems

- Typical state spaces in AI domains are exponentially large
  - Bellman's curse of dimensionality
- Learning a **model** and **utility** function
  - Can be difficult to learn good models for large complex environments
  - But if we can learn a model then learning utility function is simpler than learning Q(s,a)
  - Also can reuse the model for "related problems"
- Learning **Q-function**
  - Simpler to implement since we don't need to worry about representing and learning a model
  - But Q-functions can be substantially more complex than utility functions (**they must somehow make up for not having the model**)

Large Relational State Spaces

- When a problem has a large state space we can **not** longer represent the V or Q functions as explicit tables
  - **Generally the case for RMDPs** with a non-trivial numbers of objects
- Even if we had enough memory
  - Never enough training data!
  - Learning takes too long
- What to do??

Relational RL

- RMDPs with fixed set of objects can be "propositionalized"
  - Describe states via traditional feature vectors that list values of all properties and relations.
    - [on(a,b)=true, on(b,a)=false, ontable(a)=false, ontable(b)=true]
- Can then directly apply feature-based RL
  - Loses the relational structure provided by objects
  - Policies can't be applied directly to new object domains
  - **Can be difficult to learn from such large feature vectors**
- **Relational RL attempts to learn value functions resp. policies that directly exploit relational structure:**
  - Faster learning w.r.t. propositionalization even with fixed # of objects
  - Learn policies that generalize across object domains

Roadmap for RRL

- Function Approximation
- Relational Value Function Learning
  - Propositionalization
  - Relational Regression
- Relational Policy Learning
  - Approximate Policy Iteration
  - Nonparametric Policy Gradient

Function Approximation

- Never enough training data!
  - Must generalize what is learned from one situation to other "similar" new situations

> Basic Idea:
> 1. Represent Q-function using a compact representation
>   - Function encoding size much smaller than table
> 2. Learn function from experience instead of table

- Q-function updates arising from experience in one state can influence Q-estimate in other similar states
  - Facilitates generalization of experience
- We will first consider feature-based approximation

Feature Based Function Approx.

- Define a set of n state-action features f1(s,a), …, fn(s,a)
  - The features are used as our representation of state-action pairs
  - State-action pairs with similar features will be considered similar
  - In RRL s and a are relational states and actions
- **Example Representation**: linear approximator

$$\hat{Q}_\theta(s,a) = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

- More generally one can use any form of function approximator in terms of these features
  - Regression trees, Kernel regression, Neural networks, etc.

Q-Learning for Linear Approximators

1. Start with initial parameter values
2. Take action according to an explore/exploit policy
3. Perform Q-update for each parameter

$$\theta_i \leftarrow ?$$

4. Goto 2

Aside: Gradient Descent for Squared Error

- **Given**: sequence of states-action pairs with target Q-values $\langle s_1, a_1, q(s_1,a_1)\rangle, \langle s_2, a_2, q(s_2,a_2)\rangle, \ldots$
- **Goal**: minimize the sum of squared errors between our estimated function and each target value:

$$\underbrace{E_j}_{\text{squared error of example j}} = \frac{1}{2}\left(\underbrace{\hat{Q}_\theta(s_j,a_j)}_{\text{our estimated value for j'th state}} - \underbrace{q(s_j,a_j)}_{\text{target value for j'th state}}\right)^2$$

- After seeing j'th state the **stochastic gradient descent rule** tells us to update all parameters by:

$$\theta_i \leftarrow \theta_i - \underbrace{\alpha}_{\text{learning rate}} \frac{\partial E_j}{\partial \theta_i}, \qquad \frac{\partial E_j}{\partial \theta_i} = \frac{\partial E_j}{\partial \hat{Q}_\theta(s_j,a_j)} \frac{\partial \hat{Q}_\theta(s_j,a_j)}{\partial \theta_i}$$

Aside: continued

$$\theta_i \leftarrow \theta_i + \alpha \frac{\partial E_j}{\partial \theta_i} = \theta_i + \alpha \underbrace{\left(q(s_j,a_j) - \hat{Q}_\theta(s_j,a_j)\right)}_{\frac{\partial E_j}{\partial \hat{Q}_\theta(s_j,a_j)}} \underbrace{\frac{\partial \hat{Q}_\theta(s_j,a_j)}{\partial \theta_i}}_{\text{depends on form of approximator}}$$

- For a linear approximation function:

$$\hat{Q}_\theta(s,a) = \theta_1 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

$$\frac{\partial \hat{Q}_\theta(s_j)}{\partial \theta_i} = f_i(s_j,a_j)$$

- Thus the update becomes:

$$\theta_i \leftarrow \theta_i + \alpha\left(q(s_j,a_j) - \hat{Q}_\theta(s_j,a_j)\right) f_i(s_j,a_j)$$

Q-Learning for Linear Approximators

1. Start with initial parameter values
2. Take action according to an explore/exploit policy
3. Perform Q-update for each parameter

$$\theta_i \leftarrow \theta_i + \alpha\left(\underbrace{R(s,a) + \beta \max_{a'} \hat{Q}_\theta(s',a')}_{\text{Predicted Target q(s,a)}} - \underbrace{\hat{Q}_\theta(s,a)}_{\text{Current estimate}}\right) f_i(s,a)$$

4. Goto 2

Roadmap for RRL

- Function Approximation
- Relational Value Function Learning
  - Propositionalization
  - Relational Regression
- Relational Policy Learning
  - Approximate Policy Iteration
  - Nonparametric Policy Gradient

Relational RL via Feature Engineering

- **What if our states and actions are relational?**
- One approach: **engineer a fixed set of "relational features"**
  - Each feature returns a value for any relational state-action pair
  - Features should be well defined regardless of the number of objects
    - Unlike naïve propositionalization approach
  - Ideally feature values should be similar for similar relational states
- With such a feature representation, can use **feature-based Q-learning** to learn **a relational Q-function**.
  - Success relies critically on ability to define appropriate features
  - Often requires significant effort and insight into problem

Example: Tactical Battles in Wargus

- Wargus is real-time strategy (RTS) game
  - Tactical battles are a key aspect of the game

5 vs. 5

10 vs. 10

- **RL Task**: learn a policy to control *n* friendly agents in a battle against *m* enemy agents
  - Policy should be applicable to tasks with different sets and numbers of agents
  - That is, policy should be relational

[Fern et al.]

Example: Tactical Battles in Wargus

- **Relational States**: contain information about the locations, health, and current activity of all friendly and enemy agent
- **Relational Actions**: Attack(F,E)
  - causes friendly agent F to attack enemy E
- **Policy Structure**: each decision cycle loop through each friendly agent F and use a learned Q-function to select enemy to attack
  - I.e. select enemy E for F that maximizes Q(s,Attack(F,E))
- Q(s,Attack(F,E)) is relational since any agents can be substituted for F and E
  - We used a linear function approximator with Q-learning

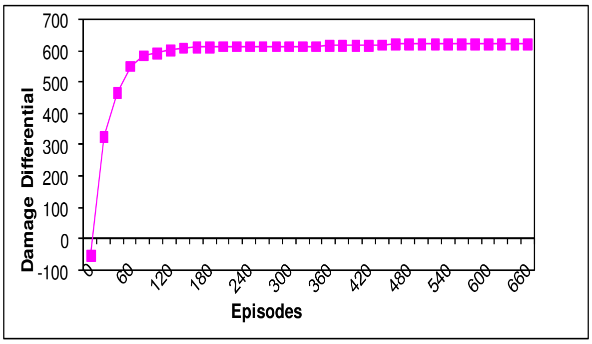
Example: Tactical Battles in Wargus

$$\hat{Q}_\theta(s,a) = \theta_1 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

- Engineered a set of relational features
  {f1(s,Attack(F,E), …., fn(s,Attack(F,E)}
- **Example Features**:
  - # of other friendly agents that are currently attacking E
  - Health of friendly agent F
  - Health of enemy agent E
  - Difference in health values
  - Walking distance between F and E
  - Is E the enemy agent that F is currently attacking?
  - Is F the closest friendly agent to E?
  - Is E the closest enemy agent to E?
  - …
- Features are well defined for any number of agents

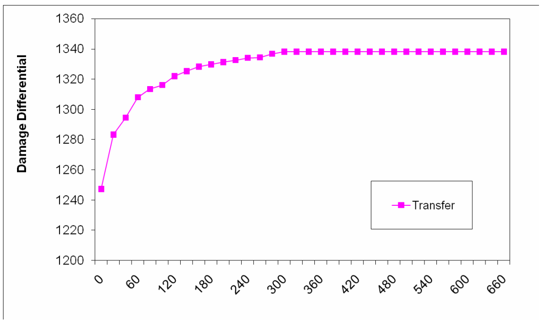
Example: Tactical Battles in Wargus

- Linear Q-learning in 5 vs. 5 battle



Example: Tactical Battles in Wargus

- Initialize Q-function for 10 vs. 10 to one learned for 5 vs. 5
  - Initial performance is very good which demonstrates relational generalization from 5 vs. 5 to 10 vs. 10



Roadmap for RRL

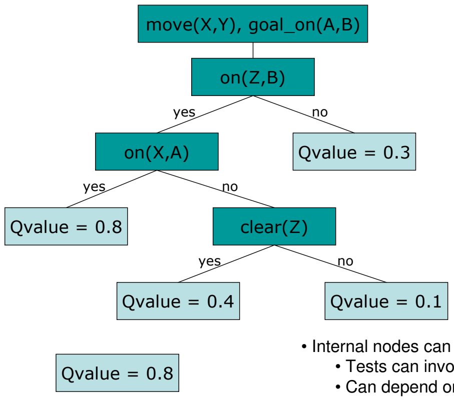
- Function Approximation
- Relational Value Function Learning
  - Propositionalization
  - Relational Regression
- Relational Policy Learning
  - Approximate Policy Iteration
  - Nonparametric Policy Gradient

Relational Regression

- The previous approach relies on feature engineering to reduce a relational problem to a propositional one
  - Requires significant effort and trial-error
- **Can we learn a relational value function automatically without propositionalization?**
- Several relational regression algorithms for batch supervised learning
  - Relational regression trees, Gaussian processes, Nearest neighbors

[Blockeel, De Raedt AIJ 101]

Relational Regression Trees



on(yellow,grey).
on(green,floor).
on(grey,floor).
on(brown,green).
clear(yellow).
clear(brown).

move(brown,floor).

goal_on(green,grey).

- Internal nodes can have relational tests
  - Tests can involve variables
  - Can depend on input objects (X,Y,A,B)

[Dzeroski et al. MLJ 43(1/2)]

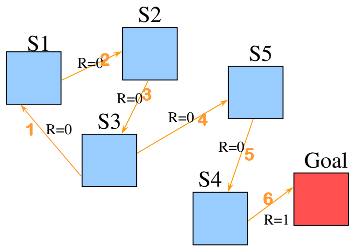
Original RRL algorithm

- TILDE is a batch learning approach, but RL is an incremental process
- RRL will accumulate training data and call TILDE periodically

> initialize an empty example-set
> **while** (true)
> - generate episode through the use of a standard Q-learning algorithm using the current tree as Q-function
> - generate example (s_i, a_i, q_i) for each state-action pair encountered
> - add the examples to the example-set
> - run tilde on the knowledge-base

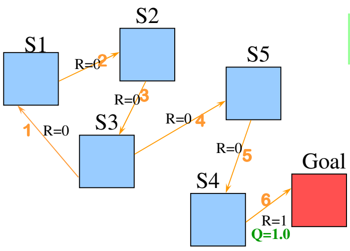
RRL Example

- Use current policy until a goal-state is reached



RRL Example

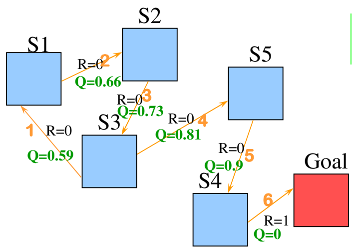
- Back propagate Q-value estimates



$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

RRL Example

- Store (state,action,qvalue) triples as batch training set
- Give to TILDE to learn a tree



$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

state(s3). action(a). qvalue(0.56).	state(s3). action(d). qvalue(0.81).
state(s1). action(b). qvalue(0.66).	state(s5). action(e). qvalue(0.90).
state(s2). action(c). qvalue(0.73).	state(s4). action(f). qvalue(1.00).

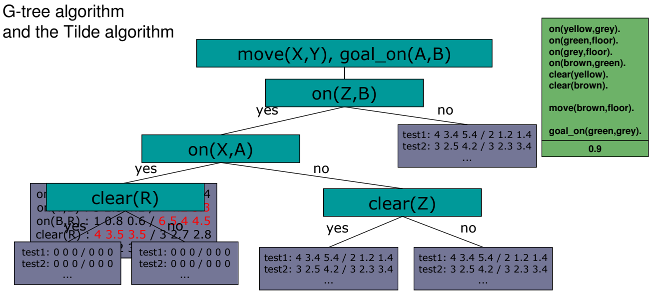
Extension RRL-TD

- Problems with RRL approach
  - Example set increases with every episode
  - No value-update of old examples
  - Trees are rebuild from scratch each episode

- **Build trees incrementally**
  - G-tree algorithm [Chapman & Kaelbling, IJCAI 91] is an online, incremental regression tree learner for propositional data
  - Straightforward to extend to relational setting

The TG algorithm

Based on the G-tree algorithm and the Tilde algorithm



$$\frac{n_p}{n}\sigma_p^2 + \frac{n_n}{n}\sigma_n^2 \quad \text{vs.} \quad \sigma_{total}^2$$

Per leaf and per test:
- Number of positive and negative examples
- Sum of q-values
- Sum of squared q-values
  - Allows to compute the significance of tests

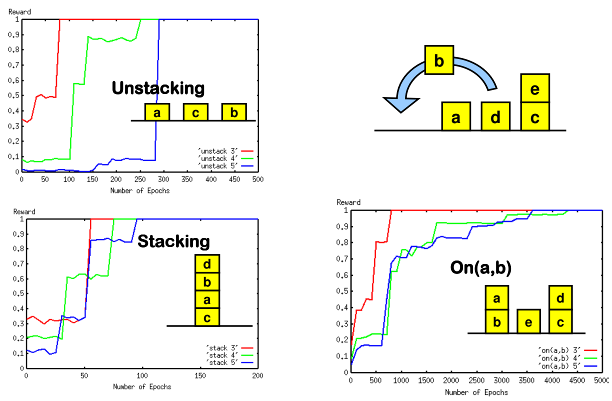
[Driessens et al. ECML01]

RRL-TG algorithm

- No need to accumulate examples generated during RRL
- Simply update tree incrementally

initialize tree to a single root node
while (true)
generate episode through the use of a standard Q-learning algorithm using the current tree as Q-function
generate example (s_i, a_i, q_i) for each state-action pair encountered
update tree using the TG-algorithm and the generated examples

Experiments. Blocks World

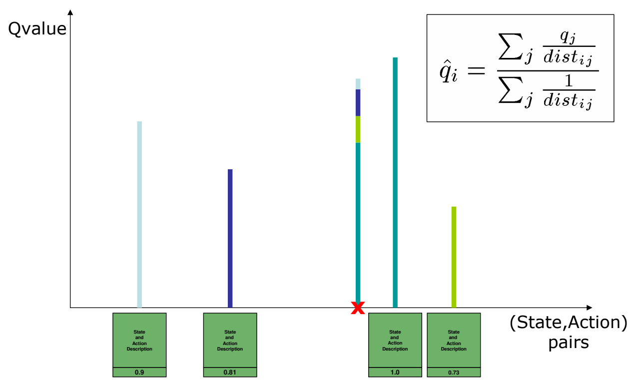


Timings

		3 blocks	4 blocks	5 blocks
Batch RRL	Stack (30 epochs)	6.16 min	62.4 min	306 min
	Unstack (30 epochs)	8.75 min	Not Stated	Not Stated
	On(a,b) (30 epochs)	20 min	Not Stated	Not Stated
RRL-TG	Stack (200 epochs)	19.2 sec	26.5 sec	39.3 sec
	Unstack (500 epochs)	1.10 min	1.92 min	2.75 min
	On(a,b) (5000 epochs)	25.0 min	57 min	102 min

[Driessens, Ramon ICML03]

Instance Based Regression



$$\hat{q}_i = \frac{\sum_j \frac{q_j}{dist_{ij}}}{\sum_j \frac{1}{dist_{ij}}}$$

- Requires a distance metric or kernel between relational state-action pairs

Instance-Based Regression

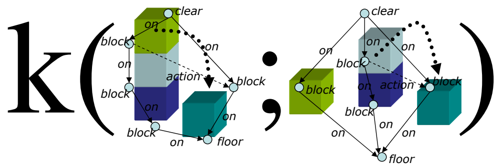
- Instance based regression typically
  - Stores past examples (s,a,q)
  - Given a new example (s',a') interpolate wrt example set to estimate q'
- Require a "kernel" K(x,x') that measures distances between any examples x and x'
- A variety of algorithms exist that turn a kernel function into a regression method
  - Gaussian Processes, Support Vector Regression, K-NN methods

Kernel for relational data?

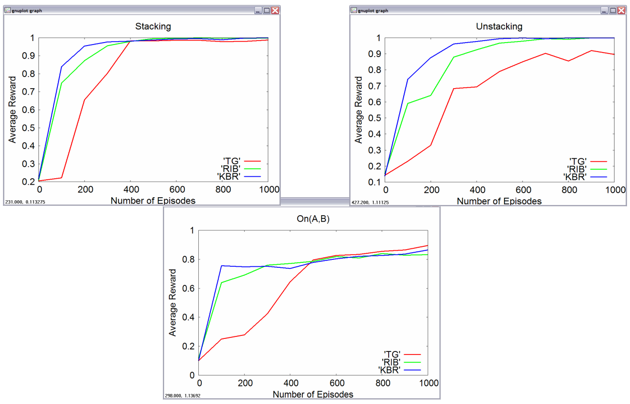
[Driessens et al. MLJ 64(1-3)]

Graph Kernels

- Relational can be viewed as graphs
- A number of kernels exist for graphs



Some Experimental Results



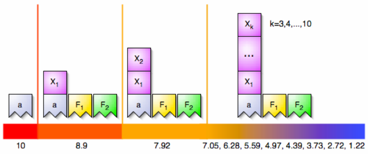
Roadmap for RRL

- Function Approximation
- Relational Value Function Learning
  - Propositionalization
  - Relational Regression
- Relational Policy Learning
  - Approximate Policy Iteration
  - Nonparametric Policy Gradient

Direct Policy Learning

- Value functions can often be much more complex to represent than the corresponding policy

Goal: cl(a)



- When policies have much simpler representations than the corresponding value functions, **direct search in policy space can be a good idea**

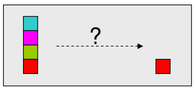
Policy: put each block on top of a on the floor

Roadmap for RRL

- Function Approximation
- Relational Value Function Learning
  - Propositionalization
  - Relational Regression
- Relational Policy Learning
  - Approximate Policy Iteration
  - Nonparametric Policy Gradient

Challenge Problem

Consider the following class of stochastic blocks world problems:



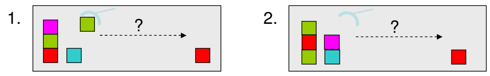
Goal: clear off blocks in the goal

- Optimal policy is:
  - obvious to us
  - simple and compact
  - independent of number of blocks

Policy for Simple Domains

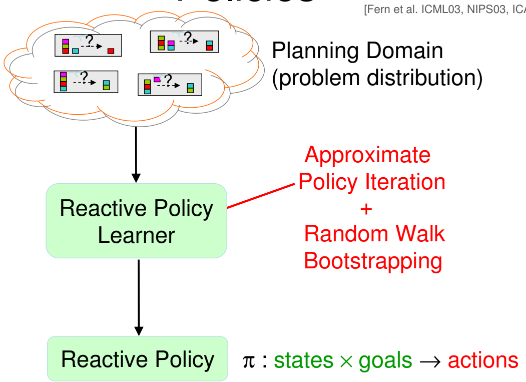
A compact policy for this domain:

1. If holding a block, put it down on the table, else…
2. Pick up a clear block above a block that is clear in the goal.

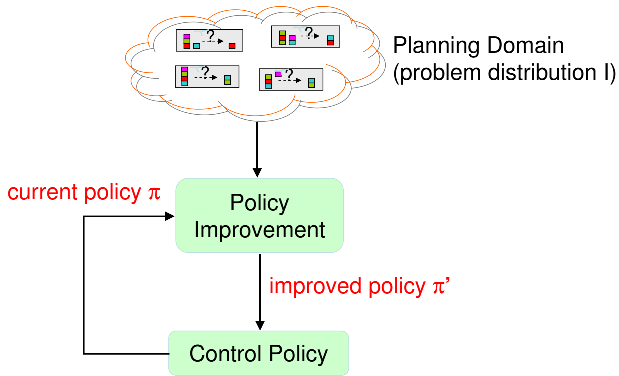


Learning Domain-Specific Policies

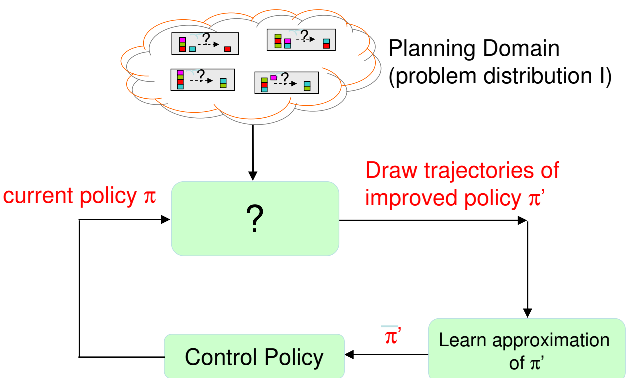
[Fern et al. ICML03, NIPS03, ICAPS04]



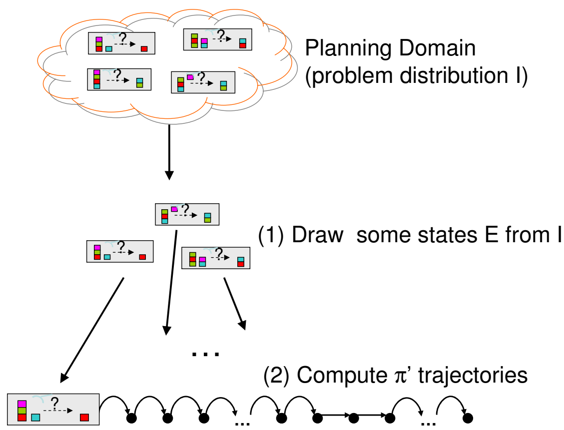
Policy Iteration



Approximate Policy Iteration



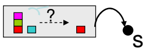
Drawing Trajectories from improved policy π'



Computing π’ Trajectories from π

Given: current policy π and problem

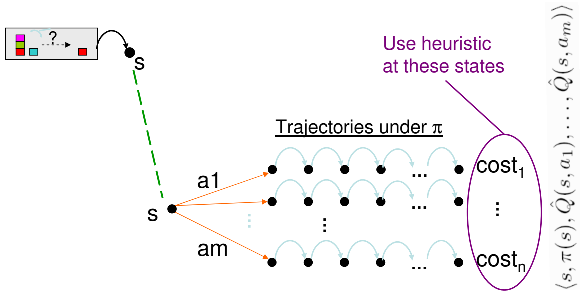
Output: a trajectory under improved policy π’



Computing π’ Trajectories from π

Given: current policy π and problem

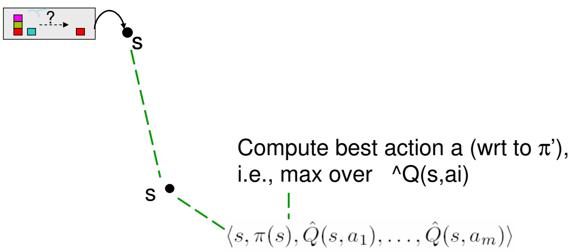
Output: a trajectory under improved policy π’



Computing π’ Trajectories from π

Given: current policy π and problem

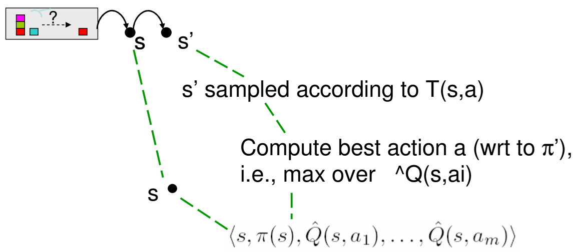
Output: a trajectory under improved policy π’



Computing π’ Trajectories from π

Given: current policy π and problem

Output: a trajectory under improved policy π’



Computing π’ Trajectories from π

Given: current policy π and problem

Output: a trajectory under improved policy π’

… and so on …

Computing π’ Trajectories from π

Given: current policy π and problem

Output: a trajectory under improved policy π’

This way, we generate examples of state-action pairs that are (approx.) sampled w.r.t. π’ because they are (approx.) evaluated taking next “value iteration” into account

Policy Learning Algorithm

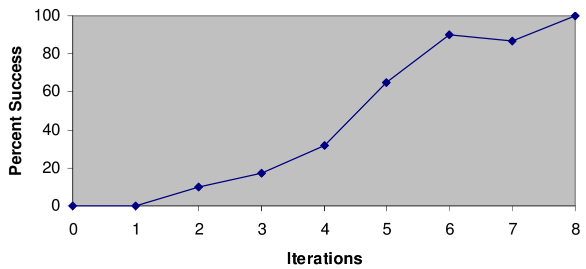
- Training data: generate π' trajectories and save state-action pairs <s_1,A_1>, <s_2,A_2>, <s_3,A_3>, etc.
- Use a Rivest-style decision-list learning approach (induce one rule at a time, in order).
  - While there are training instances remaining,
    - Find a good rule
    - Remove instances covered by the rule
  - Similar to ReBeL's maximization step
- Rules found by heuristically guided beam search
  - heuristic combines consistency and coverage
  - rules searched from small to large

Experiments

- Evaluate on the seven domains from AIPS-2000 + TL-PLAN planning competition.
  - 5 domains : can represent good policies
  - 2 domains : can not represent good policies
- Compared against state-of-the-art planner FF
  - FF's heuristic is very good for most of these domains.
- API equal or better FF's performance when we can represent policies.

Experimental Results

Blocks World (20 blocks)



Domains with Good Policies

Success Percentage

	Blocks	Elevator	Schedule	Briefcase	Gripper
API	100	100	100	100	100
FF-Plan	28	100	100	0	100

Typically our solution lengths are comparable to FF's.

Roadmap for RRL

- Function Approximation
- Relational Value Function Learning
  - Propositionalization
  - Relational Regression
- Relational Policy Learning
  - Approximate Policy Iteration
  - Nonparametric Policy Gradient

Policy Gradients with Function Approximation (Sutton et al.)

- Parameterized policy: $\pi(s,a,\theta)$
- Gradient search w.r.t. world-value

$$\frac{\partial \rho(\theta)}{\partial \theta} = \frac{\partial}{\partial \theta} \sum_{s,a} d^{\pi}(s) \pi(s,a,\theta) Q^{\pi}(s,a)$$

$$= \sum_{s,a} d^{\pi}(s) Q^{\pi}(s,a) \frac{\partial \pi(s,a,\theta)}{\partial \theta}$$

[Kersting, Driessens ICML08]

Non-Parametric Policy Gradient

- Express policy using an arbitrary potential function

$$\pi(s,a,\Psi) = \frac{e^{\Psi(s,a)}}{\sum_b e^{\Psi(s,b)}}$$

- Use a Functional Gradient (Friedmann)

$$\Delta\Psi \approx \alpha \frac{\partial\rho}{\partial\Psi}$$

Functional Gradient Boosting

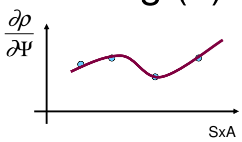
- (inspired by Friedman et al. and Dietterich et al.)
- Regular Parameterized Gradients

$$F(\theta) \rightarrow \theta_m = \theta_0 + \delta_1 + \delta_2 + \delta_3 + \ldots + \delta_m$$

- Functional Gradients

$$\Psi \rightarrow \Psi_m = \Psi_0 + \Delta_1 + \Delta_2 + \Delta_3 + \ldots + \Delta_m$$

Functional Gradient Boosting (2)



- Evaluate gradient locally
  - Wherever there is data
- Use your favorite algorithm to generalize
  - Propositional, relational, continuous

$$\Psi \rightarrow \Psi_m = \Psi_0 + \Delta_1 + \Delta_2 + \Delta_3 + \ldots + \Delta_m$$

Infinitely many parameters

In Practice

- Following Sutton et al.

$$\frac{\partial\rho}{\partial\Psi} = \frac{\partial}{\partial\Psi}\sum_{s,a} d^{\pi}(s)\pi(s,a)Q^{\pi}(s,a)$$

$$= \sum_{s,a} d^{\pi}(s)Q^{\pi}(s,a)\frac{\partial\pi(s,a)}{\partial\Psi}$$

sample

compute locally

Local Evaluation

$$Q^{\pi}(s,a)$$

Simple: Monte-Carlo estimate
Future Work: actor-critic

$$\pi(s,a) = \frac{e^{\Psi(s,a)}}{\sum_b e^{\Psi(s,b)}}$$

$$\frac{\partial\pi(s,a)}{\partial\Psi(s,a)} = \pi(s,a)(1-\pi(s,a))$$

$$\frac{\partial\pi(s,a)}{\partial\Psi(s,b)} = -\pi(s,a)\pi(s,b)$$

Gradient Tree Boosting

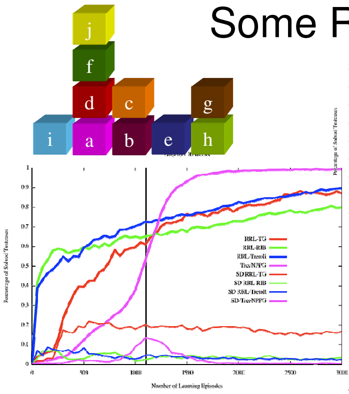
1. Generate behavior traces following $\pi(s,a,\Psi) = \frac{e^{\Psi(s,a)}}{\sum_b e^{\Psi(s,b)}}$
2. For each encountered state s
   For each available action in that state
   Generate a learning example:

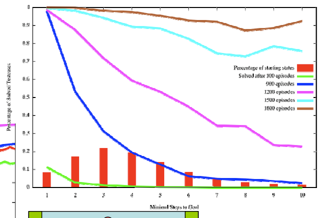
for chosen action a $\quad < s,a,Q(s,a)\pi(s,a)(1-\pi(s,a)) >$

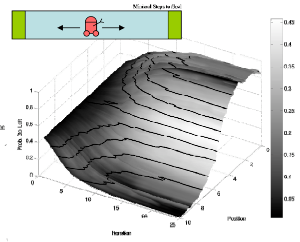
for other actions b $\quad < s,b,-Q(s,a)\pi(s,a)\pi(s,b) >$

3. Learn a tree: Δ_{n+1}
4. $\Psi_{n+1} = \Psi_n + \Delta_{n+1}$

Some Results







Conclusions: Inductive FO Planning

- Model-free RL algorithms solve MDPs without knowing the transition model
- Relational RL algorithms attempt to exploit the relational structure
  - Faster learning, policies generalize across object domains
- Can exploit relational regression and classification methods for model-free RRL
  - Feature engineering
  - Relational regression trees (RRL-TG, NPPG)
  - Relational decision lists (API)
  - Nonparametric Policy Gradient
    - Unifies finite, continuous, and relational RL