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**

**Kristian Kersting**
Fraunhofer IAIS

**Scott Sanner**
NICTA

with an invited presentation by Saket Joshi, Tufts University

---

**Planning Languages**

- **Common languages:**
  - STRIPS
  - PDDL
    - more expressive than STRIPS
    - for example, universal and conditional effects:
      
      ```
      :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)))
      ```
  - General Game Playing (GGP)
    - one or more agents

---

**Benefits of Relational Languages**

- STRIPS, PDDL, GGP are relational languages...
  - Refer to relational fluents:
    - e.g., \(\text{Blue}(?,?)\), \(\text{OnTable}(?)\)
  - specify relations between objects
  - change over time
  
  - Use first-order logic to specify...
    - action preconditions
    - action effects
    - goals / rewards
      - e.g., \((\forall \text{b} \in \text{b} ) (\text{Destination} \text{b} \text{c} ) \Rightarrow (\text{Bin} \text{b} \text{c} ))!

  - Are domain-independent and often compact!

---

**How to Solve?**

- **Relational planning problem:**
  
  ```
  \text{BoxWorld:} \quad \text{London} \rightarrow \text{Rome} \rightarrow \text{Berlin} \rightarrow \text{Moscow}
  ```

  ```
  \(\text{action load-box-on-truck-in-city}\\)
  \(\text{parameters: (}\text{b} - \text{box} , \text{t} - \text{truck} , \text{c} - \text{city})\\)
  \(\text{precondition: (}\text{BIn} \text{b} \text{c} , \text{TIn} \text{t} \text{c})\\)
  \(\text{effect: (}\text{On} \text{b} \text{t} , \text{not (}\text{BIn} \text{b} \text{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:** \(\text{BoxIn}(\text{Box}, \text{City})\), \(\text{TruckIn}(\text{Truck}, \text{City})\), \(\text{BoxOn}(\text{Box}, \text{Truck})\)

- **Goal:** \(\text{[Blue} : \text{b}, \text{BIn}(\text{b}, \text{paris})]\)

- **Actions**
  
  - `load(\text{Box} : \text{a}, \text{Truck} : \text{t})`: 
    - Effects:
      - \(\text{when (\forall \text{c} : \text{c}, \text{BIn}(\text{b}, \text{c}) \Rightarrow \text{TruckIn}(\text{t}, \text{c})) then (\text{BoxOn}(\text{b}, \text{t}))}\)
  
  - `unload(\text{Box} : \text{a}, \text{Truck} : \text{t})`
    - Effects:
      - \(\text{when (\forall \text{c} : \text{c}, \text{BIn}(\text{b}, \text{c}) \Rightarrow \text{TruckIn}(\text{t}, \text{c})) then (\text{BoxOn}(\text{b}, \text{t}))}\)

  - `drive(\text{Truck} : \text{t}, \text{City} : \text{c})`
    - Effects:
      - \(\text{when (\forall \text{c} : \text{c}, \text{TruckIn}(\text{t}, \text{c}) \Rightarrow \text{TruckIn}(\text{t}, \text{c})) then (\text{TruckIn}(\text{t}, \text{c}))}\)
  
  - \(\text{when (\forall \text{c} : \text{c}, \text{TruckIn}(\text{t}, \text{c}) \Rightarrow \text{TruckIn}(\text{t}, \text{c})) then (\text{TruckIn}(\text{t}, \text{c}))}\)
Solving Ground BoxWorld

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

**Domain Object Instantiation:**
- Box = {box1, box2, box3}, Truck = {truck1, truck2}, City = {paris, berlin, rome}

**Ground Fluents (i.e., binary state variables):**
- BoxOn(b, t, s), TruckIn(t, c, s), BoxIn(t, c, s)

**Ground Actions**
- load(b, t, truck1, truck2, box1, truck1)
- unload(b, t, truck1, truck2, box1, truck1)
- drive(t, c, truck1, truck2)
- BoxOn(b, t, s), TruckIn(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)

A First-order Solution to BoxWorld

- Derive solution deductively at lifted PDDL level:

  - if (BoxOn(b, parix)) then do noop
  - else if (BoxOn(b, parix) ∧ TruckIn(t, parix)) then do unload(b, t, parix)
  - else if (BoxOn(b, t) ∧ TruckIn(t, c)) then do drive(t, c, t, c)
  - else if (BoxOn(b, c) ∧ TruckIn(c, t)) then do load(b, c, t)
  - else do noop

- 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', 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

- Recalling the BoxWorld PDDL specification...

\[ \text{load}(\text{Box}: b, \text{Truck}: t): \]

- Effects:
  - when \([\text{City}: c. \text{BoxOn}(b, c) \land \text{TruckIn}(t, c)]\) then \([\neg \text{BoxOn}(b, c)]\)
  - \(\forall c. \text{when} [\text{BoxOn}(b, c) \land \text{TruckIn}(t, c)] \text{ then } [\neg \text{BoxOn}(b, c)]\)

\[ \text{unload}(\text{Box}: b, \text{Truck}: t): \]

- Effects:
  - \(\forall c. \text{when} [\text{BoxOn}(b, c) \land \text{TruckIn}(t, c)] \text{ then } [\neg \text{BoxOn}(b, c)]\)
  - \(\forall c. \text{when} [\text{BoxOn}(b, c) \land \text{TruckIn}(t, c)] \text{ then } [\neg \text{BoxOn}(b, c)]\)

- \(\text{drive}(\text{Truck}: t, \text{City}: c):\)

- Effects:
  - when \([\text{City}: c, \text{TruckIn}(t, c)] \text{ then } [\neg \text{TruckIn}(t, c)]\)
  - \(\forall c. \text{when} [\text{TruckIn}(t, c)] \text{ then } [\neg \text{TruckIn}(t, c)]\)

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

\[ [(C_1 \supset F) \land (C_2 \supset F)] \equiv [(C_1 \lor C_2) \supset F] \]

Successor State Axioms (SSAs)

- Default solution to frame problem given as SSAs:

\[ \gamma_+^F(\vec{x}, a, s) \supset F(\vec{x}, do(a, s)) \]

\[ \gamma_-^F(\vec{x}, a, s) \supset \neg F(\vec{x}, do(a, s)) \]

\[ F(\vec{x}, do(a, s)) \equiv \gamma_+^F(\vec{x}, a, s) \lor \neg \gamma_-^F(\vec{x}, a, s) \]

Frame Problem

- Now we have positive and negative effects

\[ \gamma_+^F(\vec{x}, a, s) \supset \text{BoxOn}(\vec{x}, do(a, s)) \]

\[ \gamma_-^F(\vec{x}, a, s) \supset \neg \text{BoxOn}(\vec{x}, do(a, s)) \]

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...

SSAs

- Shorthand:

\[ F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s) \]

\[ \gamma_+^F(\vec{x}, a, s) \lor \Phi_F(\vec{x}, a, s) \]

\[ \neg \gamma_-^F(\vec{x}, a, s) \wedge \neg \Phi_F(\vec{x}, a, s) \]

- Reality check:
Regression

- Why have we defined SSAs?

- Regression:
  - $\phi$ held after action $a$ then regression is the $\phi'$ that held before action $a$

- Exploit following properties:
  - $\text{Regr}(\neg \psi) = \neg \text{Regr}(\psi)$
  - $\text{Regr}(\psi_1 \land \psi_2) = \text{Regr}(\psi_1) \land \text{Regr}(\psi_2)$
  - $\text{Regr}(\exists x \psi) = \exists x \text{Regr}(\psi)$
  - $\text{Regr}(F(\bar{x}, do(a, s))) = \Phi_{F}(\bar{x}, a, s)$

Regression Example

- But what action instantiation of $\text{unload}(b^*, t^*)$ leads to:
  $\exists b. \text{BoxIn}(b, \text{paris}, \text{do}(\text{unload}(b^*, t^*), s))$
- Just have to existentially quantify $b^*, t^*$
  - Can obtain instances via query extraction w.r.t. state KB

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

Regression Planning

- Define abstract goal state, e.g.,
  $\exists b. \text{BoxIn}(b, \text{paris}, s)$

- Check if regression through action sequence holds in initial state

Recap

- We translated PDDL to SitCalc theory
  - converted PDDL effects to SitCalc effect axioms
  - derived SSAs from effect axioms

- Introduced regression operator
  - extracted action instantiation to achieve goal

- Let the planning begin…

First-order Goal-regression

- We can now do goal regression planning!
  - Provide initial state and sequence of actions
  - Use regression, $\exists$ 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 Example
• Golog Program:
  \((\pi b \rightarrow \neg \text{OnTable}(b,s)\?$, \text{pickup}(b), \text{putOnTable}(b))\)*,
  \(\forall b. \text{OnTable}(b,s)\)?

  • Diagram of Golog Planning:
  - Initial state need not be fully known!
  - Program exploits first-order action abstraction!
  - Heavily restricted action sequences!

  Initial State: Captures goal state?
  $\exists b, t. \text{OnTable}(b,t)$ \land \text{TruckIn}(t, \text{paris}, s)$
  \(\lor \exists b. \text{BoxIn}(b, \text{paris}, s)\)$
  \(\lor \exists b. \text{BoxIn}(b,p, s)\)$
  \(\lor \exists b. \text{OnTable}(b,p, s)\)$

  Captures goal state?
  $\text{unload}(b, t, s)$

Golog: Restricted Plan Search
• AlGO in LOGic
  – Search the space of sequential action plans
  – Regress actions to initial state to test reachability
  – Restrict action space with program:

  \begin{tabular}{|c|c|}
  \hline
  Symbol & Meaning \hline
  $\pi$ & primitive action \hline
  $\phi$ & condition test \hline
  \(\delta_1, \delta_2\) & sequence \hline
  \(\delta\) & conditional loop \hline
  \(\delta^*\) & nondeterministic iteration \hline
  \(\delta^*\) & nondeterministic choice of action \hline
  \(\delta\) & nondeterministic choice of arguments \hline
  \text{proc} & procedure call \hline
  \end{tabular}

  \(\beta\(\vec{x}\))\endProc \) procedure call definition

  \(\beta(\vec{t})\) procedure call

  \(\delta\) | \(\delta\) \nondeterministic choice of action

  \(\pi\vec{x}\) \[\delta\] nondeterministic choice of arguments

  \(\delta\) \nondeterministic iteration

  \(\delta\) \nondeterministic iteration

  \text{proc} \(\beta(\vec{x})\) \endProc \) procedure call definition

  \(\beta\) (de Giacomo, Lesperance, Levesque, AIJ-00)

  \(\text{DT-Golog: decision-theoretic, covered next (Soutchanski, Boutilier, Reiter, Thrun, AAAI-20)\)}

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)

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!
**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) \cdot 0.9
- Discount γ

**What's the best Policy?**
- Immediate vs. long-term gain?

**MDP Policy, Value, & Solution**
- Define value of a policy π:
  \[ V_π(s) = E_π \left[ \sum_{t=0}^{\infty} \gamma^t \cdot r_t | s = s_0 \right] \]
- 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: ∃π*. ∀s, π. V_{π^*}(s) ≥ V_π(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* 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?
    \[ V^t(s) := \max_{a \in A} \{ Q^t(s,a) \} \]
  - At ∞ horizon, get same value (ωV*)
    \[ \lim_{t \to \infty} \max_s | V^t(s) - V^{t-1}(s) | = 0 \]
- * acts same at each decision stage for = horizon!
**Value Function → Policy**

- Can derive policy from value function $V$
- Given arbitrary value $V$ (optimal or not)...
  - A greedy policy $\pi_V$ 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 $s$, $\pi_V$ guarantees $V(s)$ in expectation

**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…
    \[
    r_{Case(s)} = \begin{cases} 
    1 & \text{if } \text{Dest}(b,c) \Rightarrow BIn(b,c,s) \\
    0 & \text{otherwise} 
    \end{cases}
    \]
- **Operators**: Define unary, binary case operations
  - E.g., can take "cross-sum" $\odot$ (or $\ominus$, $\oslash$) of cases...

**Stochastic Actions & FODTR**

- Stochastic actions using deterministic SitCalc:
  - User’s stochastic action: $A(x) = \text{load}(b,t)$
  - Nature’s choice: $n(x) \in \{\text{loadS}(b,t), \text{loadF}(b,t)\}$
- Probability distribution over Nature’s choice:
  \[
  P(\text{loadS}(b,t) | \text{load}(b,t)) = \begin{cases} 
    1 & \text{if unknown}(x) \\
    0 & \text{otherwise} 
    \end{cases}
  \]
  \[
  P(\text{loadF}(b,t) | \text{load}(b,t)) = \begin{cases} 
    1 & \text{if known}(x) \\
    0 & \text{otherwise} 
    \end{cases}
  \]
- First-order decision-theoretic regression
  - FODTR = expectation of regression:
    \[
    \text{FODTR}(v_{Case(s)}, A(x)) = E_{P(n(x)|A(x))}[\text{Regr}(v_{Case(s)}, n(x))]
    \]
Q-functions and Backups

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

  \[ FODTR(v_{\text{Case}}(\text{unload}(b, t))) = \begin{cases} V(b, t, (b, t)) & \gamma \gamma \\ 0 & \text{otherwise} \end{cases} \]

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

- Specify a backup operator for this

  \[ B^\text{unload}(v_{\text{Case}}(s)) = r_{\text{Case}}(s) \]

  - Yields a first-order Q-function

Symbolic Dynamic Programming

- What value if 0-stages-to-go?
  - Obviously \( V(s) = r_{\text{Case}}(s) \)

- What value if 1-stage-to-go?
  - We know value for each action

\[ V(s) = \begin{cases} B^4[v_{\text{Case}}(s)] & \text{if } s \in \text{Goal} \\ B^4[r_{\text{Case}}(s)] & \text{otherwise} \end{cases} \]

  - Now just need max for every state

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

Results for SDP with FOADDs

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

  - Use FO(A)ADD ops for structured SDP (using \( \exists, 9 \) ...)

  \[ v_{\text{Case}}(s) = \begin{cases} 0 & \text{if } s \notin \text{Goal} \\ 3, B_{\text{In}}(b, Paris, s) & \text{otherwise} \end{cases} \]

  \[ 100 : \text{noop} \]

  \[ 89 : \text{unload}(b, t), \exists b, t. T_{\text{In}}(t, Paris, s) \]

  \[ 80 : \text{drive}(t, Paris), \exists b, c. B_{\text{In}}(b, c, s) \cup \exists b, t. T_{\text{In}}(t, c, s) \]

Correctness of SDP

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

  First-order (FO) MDP  \rightarrow  FOMDP Value Function  \rightarrow  Ground MDP Value Function

  Lifted FOMDP Solution  \rightarrow  Ground MDP Solution

Related Purely Deductive Approaches

- Value Iteration:
  - ReBel algorithm
  - FOVIA algorithm for fluent calculus
  - First-order decision diagrams (FODDs)

- Approximate Linear Programming (ALP)
  - First-order ALP (FOALP)

- Policy Iteration
  - Approximate policy iteration (FOAPI)
  - Modified policy iteration with FODDs

- Factored FOMDPs – FOMDP extension
  - Factored SDP and Factored FOALP

Kristian covers this.

Saket covers this.

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

ReBeL

A relational Bellman algorithm

- Instance of SDP
- Sacrifices expressivity/compactness for „simplicity“
- Abstract state = existentially 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.

Aside: Subsumption

- Recall that a clause like
  $C$: grandparent(X,Z):- parent(X,Y), parent(Y,Z)
  is $C$: $(\text{grandparent}(X,Z), \neg \text{parent}(X,Y), \neg \text{parent}(Y,Z))$

Clause $C_1$ subsume $C_2$ iff there exists a substitution $\theta$ s.t. $\beta C_1$ is a subset of $C_2$

- Thus for the following pair of clauses, $C_1$ subsumes $C_2$:
  $C_1$: mem(A,[B,C]):- mem(A,C).
  $C_2$: mem(0,[1,0]):- nat(0), nat(1), mem(0,[0]).
- Note that $C_1$ “looks” more general.
Aside: Subsumption

- Subsumption induces a "generality" lattice

ReBeL’s Abstract Actions

An action is an expression of the form

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

[Semantics]: If current state b subsumed by B, i.e., $b \preceq B$, then taking action A will result in $b \oplus B$. Otherwise, with probability $p$, and $b$ remains unchanged.

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:
  $$\text{on}(a, b) \quad 1:\text{absorbing} \quad \text{on}(a, b).$$

ReBeL’s Reward Model

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

$$V(1) = 0 \quad V(2) = 10$$

$V(c) = v(c) - B, A$

Integrity Constraints …

- are simply Horn clauses
  $$\text{false} \rightarrow \text{on}(X,Y), \text{cl}(Y)$$
  $$X \neq Y \rightarrow \text{on}(X,Y)$$

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

Summary of ReBeL’s MDPs

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

越来越好.
Step 1: Regression

- Abstract actions define how states change
- In order to do the update from state Z, we need to consider all states that can reach Z after the action
- Intuition: ‘Revers’ the action effects
- Compute the weakest precondition of Z given the action

Two cases:
1. move caused on(a,b): We have been in abstract state
   \[ S_1 = \langle \text{cl}(a), \text{cl}(b), \text{on}(a,Z), a \neq B, b \neq Z, a \neq Z, \rangle \]
2. move did not cause on(a,b): We moved X or Y but not on a or b. So, we have been in abstract state
   \[ T = \langle \text{cl}(X), \text{cl}(Y), \text{on}(X,Z), \text{on}(a,b), a \neq X \text{ or } Y, y \neq X, X \neq Z \rangle \]
   satisfying \( r = \{ \text{on}(X,Y) \neq \text{on}(a,b) \wedge \text{on}(X,Z) \neq \text{on}(a,b) \} \)

   which guarantees that applying move(X,Y) will not affect on(a,b).

   Simplifies to
   \[ S_2 = (T \wedge X \neq a), \quad S_3 = (T \wedge Y \neq b) \quad \text{and} \quad S_4 = (T \wedge Z \neq b) \]

Step 2&3: Valuation & Combination

... that means computing the Q rules

From regression, for each state \( S' \) in \( V_t \) we obtain \( S \cdot A \)-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“

\[
q(10) \quad \leftarrow \quad \text{on}(a, b), a \neq b, \text{absorb.} \quad (1) \\
q(10) \quad \leftarrow \quad \text{cl}(X), \text{cl}(Y), \text{on}(Z, Z), \text{on}(a, b), X \neq Y, \quad Y \neq Z, X \neq Z, a \neq b, \text{move}(X, Y). \quad (2) \\
q(10) \quad \leftarrow \quad \text{cl}(X), \text{cl}(Y), \text{on}(Z, Z), \text{on}(a, b), X \neq Y, \quad Y \neq Z, X \neq Z, a \neq b, \text{move}(X, Y). \quad (3) \\
q(10) \quad \leftarrow \quad \text{cl}(X), \text{cl}(Y), \text{on}(Z, Z), \text{on}(a, b), X \neq Y, \quad Y \neq Z, X \neq Z, a \neq b, \text{move}(X, Y). \quad (4) \\
q(9) \quad \leftarrow \quad \text{cl}(a), \text{cl}(b), \text{on}(Z, Z), \quad a \neq b, b \neq Z, a \neq Z, \text{move}(a, b). \quad (5) \\
q(9) \quad \leftarrow \quad \text{cl}(X), \text{cl}(Y), \text{on}(Z, Z), X \neq Y, Y \neq Z, X \neq Z, \text{move}(X, Y). \quad (6)
\]

Step 4: Maximizing …

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

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

1. Sort the Q-rules in descending order
2. Remove top \(q(x) \rightarrow B, a\) from the rules
3. If no other more general rule
4. then add \(q(x) \rightarrow B\) to \(V_{t+1}\) and remove all Q-rules that are subsumed
5. continue until no more rules

\[
\begin{align*}
10 & : \text{move}(X, Y) \n\quad & \text{cl}(X), \text{cl}(Y) \\
5 & : \text{move}(a, b) \n\quad & \text{cl}(a), \text{cl}(b)
\end{align*}
\]

Step 4: Maximizing …

\[
q(10) \quad \leftarrow \quad \text{on}(a, b), a \neq b, \text{absorb.} \quad (1) \\
q(10) \quad \leftarrow \quad \text{cl}(X), \text{cl}(Y), \text{on}(Z, Z), \text{on}(a, b), X \neq Y, \quad Y \neq Z, X \neq Z, a \neq b, \text{move}(X, Y). \quad (2) \\
q(10) \quad \leftarrow \quad \text{cl}(X), \text{cl}(Y), \text{on}(Z, Z), \text{on}(a, b), X \neq Y, \quad Y \neq Z, X \neq Z, a \neq b, \text{move}(X, Y). \quad (3) \\
q(10) \quad \leftarrow \quad \text{cl}(X), \text{cl}(Y), \text{on}(Z, Z), \text{on}(a, b), X \neq Y, \quad Y \neq Z, X \neq Z, a \neq b, \text{move}(X, Y). \quad (4) \\
q(9) \quad \leftarrow \quad \text{cl}(a), \text{cl}(b), \text{on}(Z, Z), \quad a \neq b, b \neq Z, a \neq Z, \text{move}(a, b). \quad (5) \\
q(9) \quad \leftarrow \quad \text{cl}(X), \text{cl}(Y), \text{on}(Z, Z), X \neq Y, Y \neq Z, X \neq Z, \text{move}(X, Y). \quad (6)
\]
Finally, we get …

$$V_{t+1}$$

- $v(10) \leftarrow \text{on}(a, b), a \neq b.$
- $v(9) \leftarrow \text{on}(a, Y), \text{cl}(b), \text{cl}(a), b \neq Y, a \neq Y, a \neq b.$
- $v(0) \leftarrow \text{on}(X, Y), \text{cl}(X), \text{cl}(Z), Z \neq Y, X \neq Y, X \neq Z.$

**Blocks World**

**Predicates:** on/2 and cl/1

- \( \text{on}(X, Y), \text{cl}(X), \text{cl}(Z), X \neq Y, Y \neq Z, X \neq Z \)

Note that the number of blocks is unspecified.

**Value function, \( V \), e.g.:**

- $v(10) \leftarrow \text{cl}(a)$
- $v(0) \leftarrow \text{true}$

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

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

**Blocks World: cl(a)**

**Blocks World: on(a,b)**

**ReBel: Logistics Domain**

**Reward model**

- $v(10) \leftarrow \text{bin}(b, p)$
- $v(0) \leftarrow \text{true}$

**IC and absorbing**

- $\text{false} \rightarrow \text{rain} \rightarrow \text{not.rain}$

- $\text{bin}(b, p) \rightarrow \text{absorbing} \rightarrow \text{bin}(b, p)$

- $\gamma = 0.9$

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

**First-order Decision Diagrams (FODDs)**

- Thanks to Chenggang Wang and Roni Khardon

**Techniques for First-order Planning**

- Situation Calculus
- Symbolic Dynamic Programming
- Relational Bellman Algorithm (ReBeL)
- First-order Decision Diagrams (FODDs)

**Motivation for FODDs**

- Propositional ADDs - Syntax
- Propositional ADDs - Semantics

**Propositional ADDs - Syntax**

- Variable valuation
  - (X = true, Y = false, Z = true)
- Function output: 1.5
Propositional ADDs – Normal Form

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

First-Order Decision Diagrams (FODDs) –

Idea: nodes are existentially quantified atoms (goals)

Semantics based on Single Paths

Semantics based on Multiple Paths

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

First Order Decision Diagrams (FODDs) –

Semantics defined in terms of variable valuations MAP(I, ζ)

- Domain: {1, 2, 3}
- Interpretation I: {p(1), q(2), h(3)}
- ζI = {x/2, y/3}
- MAP(I, ζI) = 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

\[
p(x) \quad p(y) \\
1 \quad 0 \\
1 \quad 0 \\
B1 \quad B2
\]

First Order Decision Diagrams (FODDs) – Reductions

Is this the most compact representation?

\[
\begin{cases}
B1 & p(x) \\
1 & p(y) \\
0 & 1
\end{cases}
\]

Strong Reduction

\[
p(x) \xrightarrow{R5} p(x)
\]

Weak Reduction

\[
p(x) \xrightarrow{R7} p(x)
\]

Value Iteration using FODDs

- Instance of SDP

\[
FODTR[v\text{Case},A(\mathcal{F})] = r\text{Case} \oplus \gamma [\exists \delta (p\text{Case}(\alpha,\mathcal{F})) \otimes \text{Reg}[v\text{Case},A(\mathcal{F})]]
\]

\[
q\text{Case}^{\delta}(A(\mathcal{F})) = \max \exists \delta. FODTR[v\text{Case}^{\delta-1},A(\mathcal{F})]
\]

\[
v\text{Case}^\delta = \max_{\alpha=v(x),v(y) \in (B_1,B_2)} q\text{Case}^\delta(\alpha)
\]

- Let’s go through an example

\[
MAP_{B_1}(I) = MAP_{B_2}(I) \text{ for any } I
\]
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)

- 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

Encoding Nature’s Choice and Reward Function

- Action choice probabilities
  - Probability of UnloadS being chosen given Unload is executed.
  - 0.7
  - 0.9

- Reward and value functions
  - Bin(b, Paris)
  - 10
  - 0

Value Iteration with FODDs

- The value iteration algorithm
  - $V_{n+1}(s) = \max_{a \in A} \{ r(s) + \gamma \sum_{s' \in S} P(s'|s,a)V_n(s') \}$
  - The first-order value iteration
  - $T^{A(s)}_{n+1}(V_n) = \bigoplus_{a \in A} (\text{prob}(A'(x)) \otimes \text{Regr}(V_n, A'(x)))$

- The first-order value iteration
  - $T^{A(s)}_{n+1}(V_n) = \bigoplus_{a \in A} (\text{prob}(A'(x)) \otimes \text{Regr}(V_n, A'(x)))$

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

- The value iteration algorithm

\[ V_{n+1}(s) = \max_{a} \left[ r(s) + \gamma \sum_{s', a} \Pr(s' | s, a) V_n(s') \right] \]

- The first-order value iteration

1. \( T_{n+1}^{A(\hat{x})}(V_n) = \bigoplus_j \left( \text{prob}(A_j(\hat{x})) \otimes \text{Regr}(V_n, A_j(\hat{x})) \right) \)
2. \( Q_{n+1}^{A} = R \otimes \gamma \otimes \text{obj} - \max(T_{n+1}^{A(\hat{x})}(V_n)) \)
3. \( V_{n+1} = \max_{a} Q_{n+1}^{A} \)

Finally, reduce the resulting FODD by block replacement

Each regression result multiplied with the corresponding choice probability

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

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

Value Iteration with FODDs - Adding regression results

Each regression result multiplied with the corresponding choice probability
Value Iteration with FODDs - Object maximization

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

\[ \text{max}_{\text{param}} \text{val} \]

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

Value Iteration with FODDs - Maximizing over actions

\[ Q_{\text{load}}^{1} = R \otimes \gamma \otimes \text{obj} \]

\[ Q_{\text{drive}}^{1} = R \otimes \gamma \otimes \text{obj} \]

\[ V_{n+1} = \max_{A} Q_{A}^{n+1} \]

Value Iteration with FODDs

\[ \Delta \left( V_{n+1} \right) = \gamma \left( \text{prob}(A_{n} \in \tilde{A}) \otimes \text{Regr}(V_{n}, A_{n}) \right) \]

\[ Q_{n+1}^{\ast} = R \otimes \gamma \otimes \text{obj} \]

ICAPS IPPC Results

Logistics:
- Without approx.
  - slower than ReBel
- with approx.
  - comparable

<table>
<thead>
<tr>
<th>GPT</th>
<th>Coverage</th>
<th>Time (ms)</th>
<th>Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy Iteration with policy language bin</td>
<td>96.66%</td>
<td>40.46</td>
<td>16</td>
</tr>
<tr>
<td>Re-Bel MDP</td>
<td>104%</td>
<td>20.63</td>
<td>16</td>
</tr>
<tr>
<td>FODD Planner</td>
<td>100%</td>
<td>85.00</td>
<td>47.5</td>
</tr>
</tbody>
</table>

FODDs for Relational MDPs - Summary

- Approximation à 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 …

Percepts

A

B

Actions

Pickup(a)?
deterministic

perfect model + goals

Now: Reinforcement Learning

Percepts

A

B

Actions

Pickup(a)?
stochastic

Transition model + Utilities

… and relational MDPs

Percepts

A

B

Actions

Pickup(a)?
deterministic

fully observable

stochastic

Transition model + Utilities

perfect model + goals

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
- Learn a policy mapping states to optimal actions

Value Function

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

- 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)

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
  \[
  Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') Q(s')
  \]

- After taking action a in state s and reaching \(s'\):
  \[
  Q(s,a) \leftarrow Q(s,a) + \alpha (R(s,a) + \gamma \max_{s''} Q(s'',a) - Q(s,a))
  \]
  (noisy) sample of Q-value based on next state
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: \[ Q(s,a) = r + \max_b Q(s',b) \]
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.
    - \([\text{on}(a,b) = \text{true}, \text{on}(b,a) = \text{false}, \text{ontable}(a) = \text{false}, \text{ontable}(b) = \text{true}]\)
- 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 \(f_1(s,a), ..., f_n(s,a)\)
  - 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
  \[
  \hat{Q}_\theta(s,a) = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + ... + \theta_n f_n(s,a)
  \]

- 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 \( \langle s_j, a_j, q(s_j, a_j) \rangle, \langle s_j, a_j, q(s_j, a_j) \rangle, \ldots \)
- **Goal**: minimize the sum of squared errors between our estimated function and each target value:
  \[ E_j = \frac{1}{2} \left( \hat{Q}_\theta(s_j, a_j) - q(s_j, a_j) \right)^2 \]
- After seeing \( j \)th state the stochastic gradient descent rule tells us to update all parameters by:
  \[ \theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j}{\partial \theta_i} \]

Aside: continued

\[ \theta_i \leftarrow \theta_i + \alpha \frac{\partial E_j}{\partial \theta_i} \]

- For a linear approximation function:
  \[ \hat{Q}_\theta(s, a) = \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \ldots + \theta_n f_n(s, a) \]
  \[ \frac{\partial \hat{Q}_\theta(s_j)}{\partial \theta_i} = f_i(s_j, a_j) \]
- Thus the update becomes:
  \[ \theta_i \leftarrow \theta_i + \alpha (q(s_j, a_j) - \hat{Q}_\theta(s_j, a_j)) f_i(s_j, a_j) \]

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
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](image1)
  ![10 vs. 10](image2)

- **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

Example: Tactical Battles in Wargus

- **Relational States**: contain information about the locations, health, and current activity of all friendly and enemy agents

- **Relational Actions**: \( \text{Attack}(F,E) \)
  - 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 \( Q(s,\text{Attack}(F,E)) \)
  - \( Q(s,\text{Attack}(F,E)) \) 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

- Engineered a set of relational features

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

- 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

![Damage Differential vs. Episodes](image3)

- 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

Example: Tactical Battles in Wargus

- Linear Q-learning in 5 vs. 5 battle

![Damage Differential vs. Episodes](image4)

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

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

• Problems with RRL approach
  – Example set increases with every episode
  – No value-update of old examples
  – Trees are rebuilt 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

![TG Algorithm Diagram]

Experiments. Blocks World

- Timings

<table>
<thead>
<tr>
<th></th>
<th>3 blocks</th>
<th>4 blocks</th>
<th>5 blocks</th>
<th>6 blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch RRL Stack (30 epochs)</td>
<td>6.16 min</td>
<td>62.4 min</td>
<td>306 min</td>
<td></td>
</tr>
<tr>
<td>Unstack (30 epochs)</td>
<td>8.75 min</td>
<td>Not Stated</td>
<td>Not Stated</td>
<td></td>
</tr>
<tr>
<td>On(a,b) (30 epochs)</td>
<td>20 min</td>
<td>Not Stated</td>
<td>Not Stated</td>
<td></td>
</tr>
<tr>
<td>RRL-TG Stack (200 epochs)</td>
<td>19.2 sec</td>
<td>26.5 sec</td>
<td>39.3 sec</td>
<td></td>
</tr>
<tr>
<td>Unstack (500 epochs)</td>
<td>1.10 min</td>
<td>1.92 min</td>
<td>2.75 min</td>
<td></td>
</tr>
<tr>
<td>On(a,b) (5000 epochs)</td>
<td>25.0 min</td>
<td>57 min</td>
<td>102 min</td>
<td></td>
</tr>
</tbody>
</table>

Instance Based Regression

![Instance Based Regression Diagram]

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

**Instance Based Regression**

- Q-value

\[
\hat{q}_i = \sum_j \frac{q_{ij}}{\text{dist}_{ij}}
\]

**Timings**

<table>
<thead>
<tr>
<th></th>
<th>3 blocks</th>
<th>4 blocks</th>
<th>5 blocks</th>
<th>6 blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch RRL Stack (30 epochs)</td>
<td>6.16 min</td>
<td>62.4 min</td>
<td>306 min</td>
<td></td>
</tr>
<tr>
<td>Unstack (30 epochs)</td>
<td>8.75 min</td>
<td>Not Stated</td>
<td>Not Stated</td>
<td></td>
</tr>
<tr>
<td>On(a,b) (30 epochs)</td>
<td>20 min</td>
<td>Not Stated</td>
<td>Not Stated</td>
<td></td>
</tr>
<tr>
<td>RRL-TG Stack (200 epochs)</td>
<td>19.2 sec</td>
<td>26.5 sec</td>
<td>39.3 sec</td>
<td></td>
</tr>
<tr>
<td>Unstack (500 epochs)</td>
<td>1.10 min</td>
<td>1.92 min</td>
<td>2.75 min</td>
<td></td>
</tr>
<tr>
<td>On(a,b) (5000 epochs)</td>
<td>25.0 min</td>
<td>57 min</td>
<td>102 min</td>
<td></td>
</tr>
</tbody>
</table>
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

Graph Kernels

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

\[ k(\cdot;\cdot) \]

Some Experimental Results

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

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
- 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
Challenge Problem

Consider the following class of stochastic blocks world problems:

\[ \text{Goal: clear off blocks in the goal} \]

• 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

Reactive Policy

Planner Domain (problem distribution)

Reactive Policy Learner

Approximate Policy Iteration

Control Policy

current policy \( \pi \)

Policy Improvement

Improved policy \( \pi' \)

Planning Domain (problem distribution)

Approximate Policy Iteration

Draw trajectories of improved policy \( \pi' \)

(1) Draw some states \( E \) from \( I \)
(2) Compute \( \pi' \) trajectories

Planning Domain (problem distribution I)

Drawing Trajectories from improved policy \( \pi' \)

\( \pi' \) Learn approximation of \( \pi' \)

Control Policy

Planning Domain (problem distribution I)

Approximate Policy Iteration

(1) Draw some states \( E \) from \( I \)
(2) Compute \( \pi' \) trajectories
Computing $\pi'$ Trajectories from $\pi$

**Given:** current policy $\pi$ and problem

**Output:** a trajectory under improved policy $\pi'$

Use heuristic at these states

This way, we generate examples of state-action pairs that are (approx.) sampled w.r.t. $\pi'$ because they are (approx.) evaluated taking next "value iteration" into account
Policy Learning Algorithm

- Training data: generate $\pi'$ 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)

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Percent Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
</tr>
</tbody>
</table>

Typically our solution lengths are comparable to FF’s.

Domains with Good Policies

<table>
<thead>
<tr>
<th></th>
<th>Blocks</th>
<th>Elevator</th>
<th>Schedule</th>
<th>Briefcase</th>
<th>Gripper</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>FF-Plan</td>
<td>28</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

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

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

$$= \sum_{s,a} d^\pi(s)Q^\pi(s,a) \frac{\partial \pi(s,a,\theta)}{\partial \theta}$$
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)
  \[ \Delta \Psi = \alpha \frac{\partial \rho}{\partial \Psi} \]

Functional Gradient Boosting

- (inspired by Friedman et al. and Dietterich et al.)
- Regular Parameterized Gradients
  \[ F(\theta) \rightarrow \theta_m = \theta_0 + \delta_1 + \delta_2 + \delta_3 + ... + \delta_m \]
- Functional Gradients
  \[ \Psi \rightarrow \Psi_m = \Psi_0 + \Delta_1 + \Delta_2 + \Delta_3 + ... + \Delta_m \]

In Practice

- Following Sutton et al.
  \[ \frac{\partial \rho}{\partial \Psi} = \frac{\partial}{\partial \Psi} \sum_{s,a} d^\pi(s) \pi(s,a) Q^\pi(s,a) \]
  \[ = \sum_{s,a} d^\pi(s) Q^\pi(s,a) \frac{\partial \pi(s,a)}{\partial \Psi} \]

Gradient Tree Boosting

1. Generate behavior traces following \( s, a, \Psi, Q^\pi(s,a) \)
2. For each encountered state \( s \)
   - For each available action in that state
     Generate a learning example:
     \[ \langle s, a, Q(s,a)\pi(s,a)(1-\pi(s,a)) \rangle \]
     \[ \langle s,b,-Q(s,a)\pi(s,a)\pi(s,b) \rangle \]
3. Learn a tree:
   \[ \Psi_{n+1} = \Psi_n + \Delta_{n+1} \]

Local Evaluation

\[ Q^\pi(s,a) = \sum_{s,a} \pi(s,a) \]

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

\[ \frac{\partial \pi(s,a)}{\partial s} = \pi(s,a)(1-\pi(s,a)) \]
\[ \frac{\partial \pi(s,a)}{\partial a} = -\pi(s,a)\pi(s,b) \]
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.