Metagaming

Overview

Metagaming is the process of analyzing and/or modifying a game during the startclock so as to improve play.

Classes of Metagaming Techniques:
- Creation of an Evaluation Function
- Game Optimization
- Materialization and Relational Optimization
- Game Reformulation
- Traditional Software Engineering
Evaluation Functions

Heuristic Search

Techniques so far are for blind search. In traditional approaches to game-playing, it is common to use evaluation functions to assess the quality of intermediate game states and, presumably, the likelihood of achieving the goal.

Example: piece count in chess.

In general game playing, the rules are not known in advance; and it is not possible to devise a guaranteed evaluation function without such rules.
Ideas for Evaluation Functions

Clune: maximize number of options

Novelty with reversibility

Distance to goal

Arguing for Mobility

Initial State: \( s_1 \)
Legal action \( a \) leads to state \( s_a \)
  Legal action \( a \) leads to state \( s_{aa} \)
  Legal action \( b \) leads to state \( s_{ab} \)
Legal action \( b \) leads to state \( s_b \)
  Legal action \( a \) leads to state \( s_{ba} \)

Terminal states are \( s_{aa}, s_{ab}, s_{ba} \)
One of these is a goal state.
Assume checking goalhood is very expensive.
Assume terminal states are equally probable as goals.
Game Optimization

Rule Ordering

Example:

\[
\text{anc}(X,Z) :- \text{parent}(X,Y) & \text{ancestor}(Y,Z) \\
\text{anc}(X,Y) :- \text{parent}(X,Y)
\]

Better Version:

\[
\text{anc}(X,Y) :- \text{parent}(X,Y) \\
\text{anc}(X,Z) :- \text{parent}(X,Y) & \text{ancestor}(Y,Z)
\]
Conjunct Ordering

Example:

\[ \text{goal}(Y) :\text{-} \text{parent}(X,Y), \text{carpenter}(X), \text{senator}(Y) \]

Solution Set Sizes:

\[
\begin{align*}
|\text{senator}(Y)| &= 100 \\
|\text{carpenter}(X)| &= 100,000 \\
|\text{parent}(X,Y)|_x &= 2.3 \\
|\text{parent}(X,Y)|_y &= 2 \\
|\text{parent}(X,Y)| &= 500,000,000
\end{align*}
\]

Better Version:

\[ \text{goal}(Y) :\text{-} \text{senator}(Y), \text{parent}(X,Y), \text{carpenter}(X) \]

Data Extraction

Original:

\[
\begin{align*}
\text{p}(10) :\text{-} &\text{ q}(a) \\
\text{p}(20) :\text{-} &\text{ q}(b) \\
\text{p}(30) :\text{-} &\text{ q}(c) \\
\text{q}(x) :\text{-} &\text{ ...}
\end{align*}
\]

Assumptions:

q is expensive to compute  
as easy to generate answers as to check answers

New, Improved Version:

\[
\begin{align*}
\text{p}(x) :\text{-} &\text{ q}(x) \land \text{ r}(x,y) \\
\text{r}(a,10) \\
\text{r}(b,20) \\
\text{r}(c,30) \\
\text{q}(x) :\text{-} &\text{ ...}
\end{align*}
\]
Materialization and Relational Reformulation

Database Views

The ancestor relation $a$ is the transitive closure of the parent relation $p$

\[ a(x,y) \Leftarrow p(x,y) \]
\[ a(x,y) \Leftarrow a(x,z) \land a(z,y) \]

The samefamily relation $sf$ is true of all pairs of people that are relatives, i.e., that have a common ancestor.

\[ sf(x,y) \Leftarrow a(z,x) \land a(z,y) \]
Using Materialized Views

\[ a(x,y) \leftarrow p(x,y) \]
\[ a(x,y) \leftarrow a(x,z) \land a(z,y) \]
\[ sf(x,y) \leftarrow a(z,x) \land a(z,y) \]

If we materialize the views \( a \) or \( sf \) then we increase the computational efficiency of answering the query \( sf \).

If we do not materialize the views \( a \) or \( sf \) then we decrease the amount of database storage space (space economy).

What are the optimal views to materialize? Database reformulation gives answers.

Querying Data Faster: Ideas

How about precomputing all elementary queries?
- not always a great idea (materializing \textit{samefamily})
Querying Data Faster: Ideas

How about precomputing predefined views?
- not too good, either (materializing ancestor)

- materializing new views that are not already defined
  on the database: relational reformulation
May Need to Invent New Relations!

Ancestor $a$: $a(x,y) \iff p(x,y)$

$$a(x,y) \iff a(x,z) \land a(z,y)$$

Same family $sf$:

$$sf(x,y) \iff a(z,x) \land a(z,y)$$

New: $has\ parent$: $hp(x) \iff p(z,x)$

New: $founding\ father$: $ff(x,y) \iff a(x,y) \land \neg hp(x)$

New: a rewriting of $sf$ in terms of $ff$:

$$sf(x,y) \iff ff(z,x) \land ff(z,y)$$

Reformulating Samefamily

Has family: $hp(x) \iff p(z,x)$

Founding father: $ff(x,y) \iff a(x,y) \land hp(x)$

A rewriting of $sf$ in terms of $ff$: $sf(x,y) \iff ff(z,x) \land ff(z,y)$
Game Reformulation

Overview

Game Reformulation is transformation of a game into one or more different games that can be played more efficiently yet yield the same result.

Sample Methods:
- Sequential Independence - Bottleneck - Maze
- Parallel Independence - Hodgepodge
- Single player abstraction
- End game book to enlarge goal set - Checkers
- Symmetry - Tic-Tac-Toe
- Hierarchical Abstraction
Buttons and Lights

Pressing button \( a \) toggles \( p \).
Pressing button \( b \) interchanges \( p \) and \( q \).

Double Buttons and Lights

Pressing button \( aa \) toggles \( p \) and toggles \( s \).
Pressing button \( ac \) toggles \( p \), interchanges \( s \) and \( t \).
Pressing button \( bc \) interchanges \( p \) and \( q \), toggles \( s \).
Pressing button \( bd \) interchanges \( p \) and \( q \), \( s \) and \( t \).
Original Version

\[
\begin{align*}
p' &\leq ac \land -p \\
qu' &\leq ac \land q \\
s' &\leq ac \land -s \\
t' &\leq ac \land t \\
p' &\leq ad \land -p \\
qu' &\leq ad \land q \\
s' &\leq ad \land t \\
t' &\leq ad \land s
\end{align*}
\]

Factored Version

\[
\begin{align*}
p' &\leftarrow a \land -p \\
qu' &\leftarrow a \land q \\
p' &\leftarrow b \land q \\
qu' &\leftarrow b \land p \\
s' &\leftarrow c \land -s \\
t' &\leftarrow c \land t \\
s' &\leftarrow d \land t \\
t' &\leftarrow d \land s
\end{align*}
\]
Import and Export

Import:
- a :- ac
- a :- ad
- b :- bc
- b :- bd
- c :- ac
- c :- bc
- d :- ad
- d :- bd

Export:
- ac :- a & c
- ad :- a & d
- bc :- b & c
- bd :- b & d

Startclock Analysis

Find disconnected components of game graph by looking for rule dependencies.

p' <= ac & -p
q' <= ac & q
p' <= ad & -p
q' <= ad & q
p' <= bc & q
q' <= bc & p

Actions can be grouped according to behavior.

p' <= ac & -p
q' <= ac & q
p' <= bc & q
q' <= bc & p
p' <= ad & -p
q' <= ad & q
p' <= bd & q
q' <= bd & p
Traditional Software Engineering

Examples

Compile the Game Description into machine code
pre-indexed rules
variable bindings on the stack
specialized unification procedures

Specialized Data Structures
e.g. Make propositional net explicit
   1 bit per proposition
   Convert rules to bit operations
Celebrating AAAI's 25th Anniversary