Game factoring (aka *game decomposition*) is the process of discovering independent subgames inside larger games.

Techniques so far provide polynomial improvement. Factoring can provide exponential improvement.

Trade-off - cost of factoring vs savings.

- Sometimes cost proportional to size of description.
- Sometimes savings proportional to size of game tree.
Hodgepodge = Chess + Othello

Analysis of joint game:
   Branching factor as given to players: $a \times b$
   Fringe of tree at depth $n$ as given: $(a \times b)^n$
   Fringe of tree at depth $n$ factored: $a^n + b^n$
Double Tic Tac Toe

Analysis of joint game:
Branching factor: 81, 64, 49, 36, 25, 16, 9, 4, 1
Branching factor: 9, 8, 7, 6, 5, 4, 3, 2, 1
Overall terminal and goals defined as disjunctions of individual terminals and goals.

Terminating one group terminates entire game.
Playing Factored Best Games

Technique:
(1) Play each factor to get a move and a score.
(2) Select subgame/move with best score.

Cost:
\[ cost(game_1) + \ldots + cost(game_k) \ll cost(game) \]

Nodes searched by Minimax on 9 board BB&L
with factoring: 3276
without factoring: 14348907
Grounding
next(p(X)) :- does(robot,a(X)) & ~true(p(X))
next(p(X)) :- does(robot,b(X)) & true(q(X))
next(p(X)) :- true(p(X)) & ~does(robot,a(X)) & ~does(robot,b(X))

next(p(1)) :- does(robot,a(1)) & ~true(p(1))
next(p(1)) :- does(robot,b(1)) & true(q(1))
next(p(1)) :- true(p(1)) & ~does(robot,a(1)) & ~does(robot,b(1))

next(p(2)) :- does(robot,a(2)) & ~true(p(2))
next(p(2)) :- does(robot,b(2)) & true(q(2))
next(p(2)) :- true(p(2)) & ~does(robot,a(2)) & ~does(robot,b(2))

next(p(3)) :- does(robot,a(3)) & ~true(p(3))
next(p(3)) :- does(robot,b(3)) & true(q(3))
next(p(3)) :- true(p(3)) & ~does(robot,a(3)) & ~does(robot,b(3))
General Game Playing
Game Grounder

Game Description:

; Buttons and Lights
; Components

{role robot}
(base p)
(base q)
(base r)
(base 1)
(base 2)
(base 3)
(base 4)
(base 5)
(base 6)

Output: 3 milliseconds

role(robot)
base(p)
base(q)
base(r)
base(1)
base(2)
base(3)
base(4)
base(5)
base(6)
base(7)
input(robot,a)
input(robot,b)
input(robot,c)
init(1)
legal(robot,a)
legal(robot,b)
legal(robot,c)
Propnet for Best Buttons and Lights
Factoring
Propnet for Best Buttons and Lights
Base and Input Propositions

**Bases:**

- `base(p(X)) :- index(X)`
- `base(q(X)) :- index(X)`
- `base(r(X)) :- index(X)`

**Inputs:**

- `input(robot,a(X)) :- index(X)`
- `input(robot,b(X)) :- index(X)`
- `input(robot,c(X)) :- index(X)`

**Results:**

- `true(p(1))`
- `true(p(2))`
- `true(p(3))`
- `true(q(1))`
- `true(q(2))`
- `true(q(3))`
- `true(r(1))`
- `true(r(2))`
- `true(r(3))`
Compute all \( \text{true}(p) \) and \( \text{does}(role,a) \) factoids needed to compute \( \text{next}(q) \) and \( \text{legal}(role,a) \) for each base proposition \( q \) and each input \( a \). These sets are called residues.

**Rules for \( \text{next}(p(1)) \):**

\[
\begin{align*}
\text{next}(p(1)) & \leftarrow \text{does}(\text{robot},a(1)) \land \neg \text{true}(p(1)) \\
\text{next}(p(1)) & \leftarrow \text{does}(\text{robot},b(1)) \land \text{true}(q(1)) \\
\text{next}(p(1)) & \leftarrow \text{true}(p(1)) \land \neg \text{does}(\text{robot},a(1)) \land \neg \text{does}(\text{robot},b(1))
\end{align*}
\]

**Residue for \( \text{next}(p(1)) \):**

\{\text{true}(p(1)), \text{true}(q(1)), \text{does}(\text{robot},a(1))), \text{does}(\text{robot},b(1))\}
{true(p(1)), true(q(1)),
  does(robot,a(1))), does(robot,b(1))}
{true(q(1)), true(p(1)), true(r(1)),
  does(robot,b(1))), does(robot,c(1))}
{true(r(1)), true(q(1)), does(robot,c(1))}

{true(p(2)), true(q(2)),
  does(robot,a(2))), does(robot,b(2))}
{true(q(2)), true(p(2)), true(r(2)),
  does(robot,b(2))), does(robot,c(2))}
{true(r(2)), true(q(2)), does(robot,c(2))}

{step(1), step(2),..., step(7)}

{true(p(1)), true(q(1)), true(r(1))
  does(robot,a(1))), does(robot,b(1), does(robot,c(1))))
{true(p(2)), true(q(2)), true(r(2))
  does(robot,a(2))), does(robot,b(2), does(robot,c(2))))
{step(1), step(2),..., step(7)}
Disjoint Action Sets

\{\text{does(robot,a(1))}, \text{does(robot,b(1), does(robot,c(1)))}\}\}

\{\text{does(robot,a(2))}, \text{does(robot,b(2), does(robot,c(2)))}\}\}

\{\text{does(robot,a(3))}, \text{does(robot,b(3), does(robot,c(3)))}\}\}
legal(robot,a(1)) :- index(1)
legal(robot,b(1)) :- index(1)
legal(robot,c(1)) :- index(1)
legal(robot,a(2)) :- index(2)
legal(robot,b(2)) :- index(2)
legal(robot,c(2)) :- index(2)
legal(robot,a(3)) :- index(3)
legal(robot,b(3)) :- index(3)
legal(robot,c(3)) :- index(3)
Partition into Different Subgames

legal(robot,a(1)) :- index(1)
legal(robot,b(1)) :- index(1)
legal(robot,c(1)) :- index(1)

legal(robot,a(2)) :- index(2)
legal(robot,b(2)) :- index(2)
legal(robot,c(2)) :- index(2)

legal(robot,a(3)) :- index(3)
legal(robot,b(3)) :- index(3)
legal(robot,c(3)) :- index(3)
Playing Factored Best Games

Technique:
(1) Play each factor to get a move and a score.
(2) Select subgame/move with best score.

Cost:
\[ \text{cost}(\text{game}_1) + \ldots + \text{cost}(\text{game}_k) << \text{cost}(\text{game}) \]

Nodes searched by Minimax on 9 board BB&L

\[ \text{with factoring: } 3276 \]
\[ \text{without factoring: } 14348907 \]
<table>
<thead>
<tr>
<th>Depth</th>
<th>Unfactored</th>
<th>Factored</th>
<th>Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28</td>
<td>36</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>757</td>
<td>117</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>20,440</td>
<td>360</td>
<td>140</td>
</tr>
<tr>
<td>4</td>
<td>551,881</td>
<td>1089</td>
<td>420</td>
</tr>
<tr>
<td>5</td>
<td>14,369,347</td>
<td>3276</td>
<td>1300</td>
</tr>
<tr>
<td>6</td>
<td>3437</td>
<td>1350</td>
<td></td>
</tr>
</tbody>
</table>

Unfactored: $1 + 27 + 27^2 + 27^3 + 27^4 + 27^5$

Factored: $(1 + 3 + 3^2 + 3^3 + 3^4 + 3^5) \times 9$
Conditions
Inertiality

Overall terminal and goals defined as disjunctions of individual terminals and goals.

Suppose one group changes on action in another group.
A subgame is *inertial* iff the state of the subgame does not change when no action in the subgame is performed.
Technique

Inertiality is computed by checking $\text{next}(p)$ for every base proposition $p$.

Reduce all rules for $\text{next}(p)$ to rules defining $\text{next}(p)$ in terms of $\text{true}(p)$ and $\text{does}(r,p)$, resolve with each other, and filter out subsumed rules.

There must be a rule $\text{next}(p) :- \text{true}(p) \& \text{junk}$ where junk does not depend on base propositions.

There must not be any rule of the form $\text{next}(p) :- \text{junk}$ where $\text{junk}$ does not include $\text{true}(p)$. 
No Termination

Overall terminal and goals defined as disjunctions of individual terminals and goals.

Is it possible for a group to have no terminal state?
Termination

Possible for a subgame not to have a termination condition at all?

Not possible. All games must terminate. If game is inertial and some subgame does not have a termination condition, then it would be possible to play repeatedly in the game forever, contradicting the requirement for termination of all games.

Upshot: No check necessary.
Is it possible for one group to terminate with low goal while another is non-terminal with higher goal?
In general, it is possible for a subgame to have a high goal and no termination while another has termination and low goal.

Problem resolved if all non-zero goal states are also terminal states.

Can be checked in a manner similar to checking for inertiality.
Validation Problem

Check that the game satisfies all of these conditions.

Almost as difficult or even more difficult than factoring.
Brian’s Method

(1) Factor

(2) Get a solution

(3) See if it works in the unfactored game.
Other Types of Game Analysis
Overall terminal and goals defined as disjunctions of individual terminals and goals.

Terminating one group terminates entire game.
Multiple Buttons and Lights

Only this group matters
Joint Buttons and Lights

Terminates if and only if *all* groups terminate.
Parallel Buttons and Lights

Terminal if and only if *all* groups terminal. Each group acted upon by different player.
Simultaneous Buttons and Lights

```
p1  q1  r1
p2  q2  r2
p3  q3  r3

aaa  aba  aca  baa  bba  bca  caa  cba  cca
aab  abb  acb  bab  bbb  bcb  cab  cbb  ccb
aac  abc  acc  bac  bbc  bcc  cac  cbc  ccc
```
Conditional Factoring

\[\begin{array}{ccc}
  & X & O \\
  X & O & O
\end{array}\]
Conditional Factoring

[Diagram of a game board with 'X' and 'O' placement]
Bottlenecks
Series of games
each of which must terminate before next begins

Dead State Elimination
Find states that cannot lead to acceptable outcomes
Prune whole subtrees

Goal Monotonicity
Detect monotonicity in states
e.g. higher goal value in non-terminal states
correlated with progress toward goal