Pruning Techniques
Metagaming

Metagaming is match-independent game processing, i.e. game processing that is done independent of any particular opponent or any particular state.

Objective of metagaming - to optimize performance in playing specific matches of the game.

Usually done offline, i.e. during the startclock or between moves or in parallel with regular game play.
Game Reformulation

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:
- Disjunctive Factoring
- Conjunctive Factoring
- Sequential Factoring - Bottleneck - Maze
- Symmetry - Tic-Tac-Toe
- Hierarchical Abstraction
Metagaming Ideal

Trade-off - cost of processing vs savings
Find cases where cost varies
with size of small description (e.g. propnet)
rather than with size of expanded game graph

Techniques:
Game Reformulation
Pruning Techniques
Evaluation Functions
Dead State Removal
Latches

A *latch* is a proposition that persists indefinitely, i.e. once it becomes true, it stays true forever.

Tictactoe:

\[
\text{next} \left( \text{cell}(M,N,x) \right) \text{ :- } \text{true} \left( \text{cell}(M,N,x) \right) \\
\text{next} \left( \text{cell}(M,N,o) \right) \text{ :- } \text{true} \left( \text{cell}(M,N,o) \right)
\]

Latches:

\[
\text{cell}(1,1,x) \\
\text{cell}(1,2,x) \\
\ldots \\
\text{cell}(1,1,o) \\
\text{cell}(1,1,o) \\
\ldots
\]
Finding Latches Using Propnets

Naïve algorithm for detecting latches - Try all combinations of values for base and input propositions in which \( p \) is true and check that it is true in the next state.

NB: Effectively searches entire state space.

Improved version - As above but restrict attention to those propositions that determine \( p \).

NB: Improved version frequently much better than naive.
Requirements and Antirequirements

A proposition $p$ is a *requirement* / *antirequirement* for a proposition $q$ if and only if $p$ is true / false in every state in which $q$ is true.

In Tictactoe, $\text{cell}(1,1,x)$ is a requirement for $\text{row}(1,x)$. $\text{cell}(1,1,x)$ is an antirequirement for $\text{cell}(1,1,o)$. 
Inhibition

A proposition $p$ inhibits a proposition $q$ if and only if $p$ must be false to achieve $q$ or retain $q$. In other words, whenever $p$ is true in a state, $q$ is false in the next state.

Example in a minute.
Dead State Removal

If a latch $p$ inhibits a requirement for the goal in a game, then it is a good idea to avoid states in which $p$ is true.

Example (assuming $p$ is a latch and inhibits $q$):

```prolog
goal(white, 100) :-
    true(q) &
    true(r)
```

Pruning game tree below any state in which $p$ is true can save lots of computation.
Untwisty Corridor

Roles: Robot
Actions: a, b, c, d
Bases: p, q1, ..., q8, 1, ..., 9
Goal: q8
Actions a, b, c make p true.
Once true, p remains true.
Action d successively sets q1, ..., q8 iff p is false.

Plan: d, d, d, d, d, d, d, d, d
Game tree size: 349,525
Performance

? (time (genplan untwistycorridor))
    349,521 states
    1,133,293 milliseconds.
597,682,424 bytes of memory allocated.
(PROG D D D D D D D D)

? (time (strplan untwistycorridor))
    9 states
    183 milliseconds
    54,568 bytes of memory allocated.
(PROG D D D D D D D D D)
And So Forth

Hillclimbing

Goal Proximity

Etc.