Rook Jumping Maze Generation

Todd W. Neller
Gettysburg College
**Rook Jumping Maze**

- **Specification:** grid size, start state (square), goal state, jump numbers for each non-goal state.

- **Jump number:** Move exactly that many squares up, down, left, right. (*Not diagonally.*)

- **Objectives:**
  - Find a path from start to goal.
  - Find the shortest of these paths.
The Assignments

- [http://modelai.gettysburg.edu/2010/rjmaze](http://modelai.gettysburg.edu/2010/rjmaze)
- Preliminaries:
  - Maze Representation
- Uninformed Search:
  - Maze Evaluation
  - Maze Evaluation II
- Stochastic Local Search:
  - Hill Descent
  - Hill Descent with Random Restarts
  - Hill Descent with Random Uphill Steps
  - Simulated Annealing
- Machine Learning
  - Restart Bandit
  - Restart SARSA
Problem: Generate and print a random \( n \)-by-\( n \) Rook Jumping Maze (5 \( \leq \) \( n \) \( \leq \) 10) where there is a legal move (jump) from each non-goal state.
Maze Evaluation

Problem: [Maze Representation step] Then, for each cell, compute and print the minimum number of moves needed to reach that cell from the start cell, or "--" if no path exists from the start cell, i.e. the cell is unreachable.

- Breadth-first search
- Print objective function: negative goal distance, or a large positive number if goal is unreachable.
Stochastic Local Search

- Maze design as search
  - Search space of possible maze designs for one that minimizes objective function.

- Stochastic Local Search (SLS):
  - Hill Descent
  - Hill Descent with Random Restarts
  - Hill Descent with Random Uphill Steps
  - Simulated Annealing

- Additional resources for teaching SLS:
  [http://cs.gettysburg.edu/~tneller/resources/sls/index.html](http://cs.gettysburg.edu/~tneller/resources/sls/index.html)
As You Wish...

“I really hate this damned machine; I wish that they would sell it. / It never does quite what I want but only what I tell it.” – Anonymous

Our maze designs are only as good as our obj. function.
- While maximizing shortest path is a simple starting point for maze design, we can do better.

Maze Evaluation II
- **Problem**: Define an *better* maze objective function and argue why it leads to improved maze quality.
- **Features**: Black/white holes, start/goal positions, shortest solution uniqueness, forward/backward branching, same-jump clusters, etc.
- Forthcoming ICCG’10 paper: *Rook Jumping Maze Design Considerations*
Machine Learning

- SLS is an anytime algorithm
  - More search iterations → same/better maze design
  - When generating many mazes, how does one balance utility of computational time versus utility of maze quality?

- **Restart Bandit**
  - $n$-armed bandit MDP with # iterations as arms
  - $\epsilon$-greedy/softmax strategy for action selection

- **Restart SARSA**
  - Use SARSA to map # iterations since restart and best maze evaluation to actions {GO, RESTART}
Variations

Many variations are possible to avoid plagiarism:

- Use different regular tilings, e.g. triangular or hexagonal.
- **Topological constraints** may be added (e.g. impassable walls/tiles) or removed (e.g. toroidal wrap-around).
- **Movement constraints** may be varied as well.
  - Add diagonal moves → Queen Jumping Maze
  - Abbott's "no-U-turn" rule increases state complexity
The best puzzle assignments have a high fun to source-lines-of-code (SLOC) ratio:
- RJMs are fun, novel, interesting mazes with simple representation and rules.
- RJMs are particularly well suited to application of graph algorithms (evaluation) and stochastic local search (design).

Everything you’d ever want to know about RJMs:
- [http://tinyurl.com/rjmaze](http://tinyurl.com/rjmaze)

Questions?