Automatic Design of Chess Variants

There are well-known heuristics for designing good games of various kinds, most of which revolve around making sure that players do not have dominant options at most major decision points (in fact, the lack of dominant options is kind of what makes a decision point "major" in the first place). I would like to develop these relatively vague heuristics for human designers into conditions on the game tree for chess-like games instantiated in GDL. I will define generalized classes for chess pieces, based on how they move, and work out heuristics for their effect on the game tree. The ultimate goal will be a program that takes in a board size and some conditions on the armies of each player, and returns the setup and rules for a fun chess variant, possibly with simple fairy pieces.

It would be nice to be able to test the heuristics, but I'm not sure it's feasible, since the audience for chess variants is rather small to begin with.

Notes on Generalizing Chess

Chess is hard to generalize when we're dealing with the problem of analyzing the game tree for play, but it's relatively easy to generalize from a game design standpoint.

The most important part of a chess piece's movement is the number of squares it can reach, both in general and on any given turn. The other important piece is whether it can be blocked.

Knights can be treated as pieces which are allowed to move one space at a 30 or 60 degree angle from the row/column directions. I will probably *not* deal with pieces that are allowed true jumps, but might.

Pawns are the hardest pieces to incorporate into a general scheme for chess; I will almost certainly leave off the initial double move (and therefore *en passant*, and I may or may not just replace them with the forward-capturing pawns seen in other descendants of the original game.