Crafty Command Documentation (version 18)

Crafty is nothing more than a long-time hobby of mine, dating back to Blitz and later Cray Blitz. People ask me how I keep doing this, and that is the one question that generally leaves me at a loss for words.

Perhaps the most common question I'm asked is "is this version of Crafty some dumbed-down version of what you play on ICC or what you use at a computer chess event?" The answer is a resounding *NO*. The current version is *exactly* what is running on ICC under this version number. Note that a new version can, on occasion, introduce weaknesses or outright bugs that were not present in previous "gold" versions. As a result, you should be careful to back up your "favorite" before trying the latest and greatest. If you aren't satisfied with the new version, you can then go back to what you believe is a better version.

If you are looking for the strongest playing computer chess program available, you should likely look to Fritz, Rebel, Tiger, and the other commercial entries. There you will find strong opponents with polished interfaces that have been tested in a systematic and careful way. If you are looking for a program that plays good chess, has a reasonable set of features for you to use, is available in source form, and one where the author welcomes feedback, code or suggestions, then you are at the right place. I welcome comments and suggestions, and also feedback from ideas you try yourself that seem to work.

Crafty is a state-of-the-art computer chess program, and uses all of the search algorithms you have probably read about, negascout search, killer/history move ordering, SEE (Static Exchange Evaluation) quiescence move ordering and pruning, hash (transposition/refutation) tables as well as evaluation caches, selective extensions, recursive null-move search, and a host of other features that have been used and are still being used in most computer chess programs. If it's not in Crafty, either it is on the "to do" list, or it has been tried, found wanting, and discarded.

Chess Knowledge is growing, and suggestions (or even better, real code) are welcome. This is the best place to contribute your ideas, because knowledge can be used to supplant search and make it play better. The evaluation is probably the easiest place to start studying Crafty because of the comments and simplicity of using bitmaps, *once* you get "into" them.

My purpose for doing this is an exercise in computer chess efficiency. I can't begin to count the number of people I know that started from scratch to write a chess program. Even larger is the group that started from scratch, and gave up before finishing, because of the basic size of the project.

Crafty offers everyone a very clean starting point, if you are fascinated by the bitmap chess board implementation (as I am). The search and quiescence code is reasonably straightforward, as is the evaluation,

It offers a great starting point, so that if you are interested in trying a new search extension, you can be testing tomorrow, rather than next year, because you start with a fully functional chess engine that is not a "toy" application, but is a functional and "dangerous" chess player. It offers a rapid start, although you can certainly replace it piece by piece until it is "yours" if you want. It also offers a fairly complete set of commands and an interface for a GUI as well as support for chess server play, so that testing and debugging your new ideas is greatly simplified.

If you'd like more information, please check out the read.me document and the crafty.FAQ that are distributed with Crafty. These contain recent news and specific instructions for commonly asked questions, like "where can I obtain tablebase files and how do I use them?"

How to play a game.

When you execute Crafty, you will immediately be greeted by the prompt string "white(1): " and Crafty will wait for commands. This prompt means it is white on move, and we are at move #1 for white. You can first use any of the commands from the alphabetic command listing below to tailor the game to your liking (time control, hash table size, book randomness, etc.) and then you have two choices. If you want to play white, just enter your move, and Crafty will take it from there and make a move in response. You will then be prompted by "white(2):" and it is your move again. If you would prefer to play black, just enter either "move" or "go" at the prompt and crafty will move for that side rather than accepting a move from you. After it makes its move for white, you will then see the prompt "black(1): " indicating it is now time for blacks first move. You can enter a move, or you can once again enter "move" or "go" and Crafty will again move for the current side, change sides, and prompt you for what to do next.

If you find yourself continually using a set of commands to configure crafty to play as you want, you can put these commands in a startup file called .craftyrc (Unix) or crafty.rc (DOS/Windows). The format for this file is just like you would type the commands at the keyboard, with the requirement that the last line of the file must be "exit" on a line by itself. Using this, each time you start Crafty, it will first execute the commands from this file before it prompts you for input.

While Crafty is running, you can control what it displays, but here's a couple of samples to explain what it is saying and why:

```
         depth   time   score    variation (1)
          book moves {d4, c3, Nc3, d3, b3, c4, g3, b4, Be2, Bb5}
          book   0.0s     70%    d4

White(3): d4
          time used:   0.01
```

This is the normal output for those cases where Crafty is in book. The book moves line gives the set of book moves that made the first selection cut (see the book selection explanation given later), followed by the move actually played, in this case d4.

If Crafty is out of book, then the output looks somewhat different as given below:

```
  depth   time   score    variation (1)
   4->   0.81    2.09    6. dxe4 Bxe4 7. Rad8 Qf2 8. Qb5
   5     1.37    2.41    6. dxe4 Bxe4 7. Ne5 Qf4 8. Bxe4+ Qxe4 9. f5
   5->   1.88    2.41    6. dxe4 Bxe4 7. Ne5 Qf4 8. Bxe4+ Qxe4 9. f5
   6     7.38     --    6. dxe4
   6    11.90    1.97    6. dxe4 Bxe4 7. Rab8 Qf2 8. Qc7 Nc5 9. Qe5
   6    12.92     ++    6. Ne5
   6    13.71    2.23    6. Ne5 Qg2 7. Ng6 h5 8. Nh4 Qg4
   6->  15.59    2.23    6. Ne5 Qg2 7. Ng6 h5 8. Nh4 Qg4
  time: 15.60  cpu:99%  mat:1  n:246565  nps:15927
  ext-> checks:4706 recaps:1336 pawns:0 1rep:301
  nodes  full:45951  quiescence:200614  evals:104657
  endgame tablebase-> probes done: 0  successful: 0
```

Let's take this stuff one line at a time. Lines that have something like 4-> in the depth column are printed when that iteration (depth) is completely finished. The time and score columns should be obvious as to their meaning as is the PV, the sequence of moves that led to this score. One note about the "score" column. As of version 18, Crafty displays the score with + values good for white, - values good for black, no matter which side it is playing in the game. All output now follows this convention, from playing, to analysis mode, to annotating your games, to whispering/kibitzing on the chess servers, and so forth. This is unlike other engines, but once you get used to it, it is much less confusing when you remember that negative scores are good for black and bad for white, and vice-versa.

the line that has -- in the score column means that when we started depth 6, dxe4 turned out to be worse than we thought (notice score dropped from 2.411 last search to 1.972 for this move this search.) To resolve this, Crafty lowers the lower search bound (alpha) and re-searches the move to find the score. The line with ++ means that this move (Ne5) is better than the best move so far, so Crafty raises the upper search bound (beta) and re-searches this move to find the new score.

the first line of statistics gives the total time taken for this search, the cpu percentage which should stay at 98-100% unless your machine is heavily loaded or unless Crafty is in an endgame that is having lots of contact with endgame databases. If this drops below 98%, it means that Crafty is not getting full CPU usage and will be playing weaker than normal. The mat:1 is simply the true material score, since Crafty's positional scores are often larger than a pawn.

Alphabetic Listing of Commands

----

1. alarm on|off This command is used to control Crafty's "beep" after it makes a move. Turning this off will make Crafty "quiet" when it plays, but also makes it easy to miss a move if you are using crafty to play in a tournament. This is primarily designed to make Crafty tolerable during late night matches.

2. analyze This command puts crafty into analyze mode. In this mode, Crafty starts computing for whichever side is on move, and it continues computing and showing its analysis until a move is entered. This move is made, Crafty changes sides, and starts thinking and printing analysis all over, but for the other side now.

This command is useful to play through a game, because you get instant feedback when you try a move. If you want to try a different move from the one you just entered, use the "back" command to back up one move, or use "back <n>" to back up <n> moves. Note that one move is a single move for the last player, not a move for both sides. To unmake the most recent 2 moves (one for black, one for white) use "back 2".

3. annotate|annotateh <filename> <colors|name> <moves> <margin> <time> This command is used to annotate (make comments in) a game that has already been played.

The annotate command produces a file with the .can extension added to the original name. This file will contain pure ascii information from the annotation pass. "annotateh" produces an HTML file instead (with the .html extension). This includes the normal output, plus a nice bitmapped graphical board display for every position where crafty had 'something to say'.

<filename> is the name of the file that has the game moves stored in it. This should be a PGN-compatible file, although Crafty can read nearly any file with chess moves and convert it to pgn using the "read" and "savegame" commands to perform the conversion.

<colors|name> indicates which side Crafty will annotate. The valid choices are w, b, and wb/bw for white only, black only, and both, respectively. Crafty will search and produce results for the indicated color only, making moves for the other side silently as they are read in.

Alternatively, you can specify the player's name (useful if you want to annotate several of your own games in one large pgn file, for example, and you alternated colors so that you can't pick the right one easily). Crafty will then figure out which side to annotate for in each game. Note that the name is case-sensitive, but that you only have to enter a string that is unique in the name field. IE if one name is "Anatoly Karpov" and the other is "unknown" then specifying Karpov as the name would be sufficient. If the same 'string' appears in both names, Crafty will complain.

<moves> indicates the moves that should be annotated. If this is a single integer, annotation starts at this

move number (for the color given above) and proceeds for the rest of the game. If a range is given, as (20-33), then only moves 20-33 inclusive are annotated. To annotate the complete game, you can use 1-999.

<margin> gives a score "window" that controls whether Crafty will produce comments (see below). The larger this number this number, the fewer annotations Crafty will produce. A negative number will result in an annotation for every move selected.

<time> indicates the time limit for each search. Since each move selected requires two searches, you can take the number of moves, double this number and multiply by <time> to determine how long the annotation process will take. This time is in seconds.

How it works. Suppose you use the command "annotate game1 w 1-999 1.000 30" This asks Crafty to read the file "game1", and annotate the white moves for the entire game. The margin is 1 pawn and the search time limit is 30 seconds. The output for the annotate command is found in <filename>.can, in this case this is game1.can.

Crafty first searches the move actually played in the game to determine the score for it. Crafty then searches the same position, but tries all legal moves. If the score for the best move found in this search is greater than the score for the move actually played plus the margin, then a comment is added to the output file. This output file is quite short, with all the game moves (plus any PGN tags in the original, for identification purposes) plus the brief comments. An annotation looks like this:

{real_value (depth:best_value PV moves)}

real_value is the score for the move actually played. depth is the depth Crafty searched to produce the best_value and PV for what it thinks is the best sequence of moves for both sides. If you set <margin> to 1.000, you are asking Crafty to only annotate moves that either lost a pawn or more, or moves that failed to win a pawn or more. If you set <margin> to .300, you are asking for annotations for any move that makes the score drop about 1/3 of a pawn below the value for the best move Crafty found.

If you have other moves you would like to see analyzed during this annotate process, at the point where the move can be played, insert it into the PGN file as an analysis comment, surrounded by () or {} characters. Crafty will produce analysis for this move as well. If more than one move appears inside a single set of delimiters, only the first will be analyzed. To force Crafty to analyze more than one move, enter them like this: (move1) (move2) as though they were two separate comments.

4. ANSI on|off This command is used to control whether or not Crafty attempts to display its move in reverse video or not. For PC's, Linux, and most Unix boxes, this works fine. Should you find yourself playing crafty via a dumb terminal, this might hose the terminal and interfere with your ability to see or input moves. If moves are not displayed in reverse video, it's probably wise to turn this off to avoid hanging the terminal you are using.

5. black|white This command simply toggles the side on move. if it is white to move, and you enter white, nothing happens. If it is white to move and you enter black, then it becomes blacks turn to move immediately from the same position. Used only infrequently.

6. book (see the book explanation near the end of this document for a full explanation of this command and its many options.) Note that there are special commands available (*only* on the command line, *not* in the crafty.rc/.craftyrc files) to direct crafty to specific directories for the book files (bookpath=/a/b/c), the tablebase files (tbpath=/i/j/k) and the log files (logpath=/x/y/z). Note that these commands can *only* be used on the command line, because they must be executed before the engine is initialized. Putting them in the crafty.rc/.craftyrc file will produce error messages without affecting how the files are opened.

If you need to specify multiple directories (tbpath only) you may do so by using "tbpath=path1:path2:path3:etc" or else use the more Unix- like "tbpath=(path1:path2:path3:etc)" instead. The paths are separated by ":" (colon) characters and everything is case-sensitive as usual. For dos/windows users, the separator can be a semi-color (;) or a comma(,) to avoid the drive designation ambiguity.

7. cache=N This command is used to alter the cache size used for endgame database probes. N can be a simple integer, representing the number of bytes to use or it can be specified as nK or nM representing n * 1024 bytes or n * 1024 * 1024 bytes. This should be in multiples of the database "chunk" size, which might vary. Using the nM form guarantees that you will have a reasonable number.

8. clock <ctime> <otime> This command is primarily intended for use when Crafty is playing in a tournament, such as the WMCCC or WCCC events. If the operator is somewhat slow in entering moves, or forgets to stop the clock after making a move for Crafty, the chess clock for the game will drift from the values that Crafty maintains internally. <ctime> is the time (in minutes or hh:mm format) crafty has left until the next time control, and <otime> is the opponent's remaining clock time. This command can be used at any time, but will only affect the time per move *after* crafty makes the current move and starts to think about what the opponent might do next.

9. computer This command usually comes from xboard/winboard, but can be used at any time to tell Crafty it is playing a computer. This will prevent some things from happening, such as a draw score that varies, as well as adjusting the book selection code to be more selective in what it plays.

10. display this command is used to display the game board. This board is displayed using the ICS style #1 type of ASCII display, with white always at the bottom of the screen, black at the top. Very unusable to play a game with, but good to verify a position after it has been set up.

This command is also used to display the various piece/square tables, by typing "display <piece>" where <piece> is replaced by pawn, knight, bishop, rook, queen or king. The board is oriented in the same way as the display board with a one-to-one correspondence between the squares. Perhaps useful for curiosity, but not for anything else. These values can not be modified by the user.

The final version of this command is used to control what kind of output you will see when crafty runs. Currently the following options are available.

> display time: this will make Crafty display the amount of time each side takes after making a move.
>
> display changes: this will make Crafty display the PV each time it changes during the search, including when a move fails high or becomes a new best move.
>
> display variation: this will make Crafty display the PV at the end of each iteration, but it will only show the best PV for the entire iteration, not all of the changes.
>
> display stats: this enables basic search statistics output including time, nodes and so forth.
>
> display extstats: this enables extended search stats including the hashing statistics, search extension statistics and so forth.
>
> display movenum: causes all PV output to have move numbers embedded in them to make the PV possibly easier to read. This causes the PV to look like this: 12. ... Nxe4 13. Bxe4 h6 rather than simply Nxe4 Bxe4 h6. This is very helpful when playing on a server and whispering or kibitzing analysis. It will also be useful when crafty is run from within a database program as the move numbers will sync up with the actual game.

display moves: will display each root move as it is searched, along with updating the search time at the bottom of the screen, so you can see what move is currently being analyzed.

display general: will display general information messages whenever Crafty wants to tell you something (ie "clearing hash tables" or other such things like "Mate in n moves."

If you put a "no" in front of any of these options, that will disable that particular type of output.

11. draw offers Crafty a draw. It generally will look at the value returned by the last search, and compare it with the value returned by an internal function DrawScore(). If the search value is not above this result, then Crafty will accept the draw. If the search value is above the theoretical value for a draw, Crafty will decline the draw. Note that crafty will offer draws based on internal analysis. When it offers a draw, you can respond with "draw" although the game does not really end until you exit Crafty.

12. drawscore N sets the draw score (or contempt factor) to N. If you want crafty to avoid draws, set this number to something that is negative. IE -50 says that a repetition (draw) is the same as being 1/2 pawn down. Setting it to +100 will make it try very hard to draw because that looks like it is winning a pawn when it does so. Note that this is dangerous (either way) as a -50 in a king and pawn ending is very likely dead lost... and a repetition is better.

13. echo <text> This command is normally used inside a command file that you are going to use to "feed" crafty some positions for analysis or whatever. Since crafty depends on the operating system to echo commands as they are typed, commands read in from a file are "invisible." This gives you the ability to insert commands into such a file so that crafty displays a message on the screen to give you an idea of where it is in processing the file.

14. edit This command has been "cloned" from GnuChess to provide an interface with Xboard. After entering the "edit" command, you are in "edit.white" mode, where any piece/square combination you enter will add the indicated white piece on the given square. Piece/squares are entered as "qa3", or "bc4" for example. This puts a white queen on a3 and a white bishop on c4. Once all white pieces are entered, typing a "c" changes to "edit.black" mode where piece/square combinations now place black pieces. Typing a "." character exits edit mode. To clear the board initially, you use the "#" character.

Here's a sample to set up the original starting position, after white has played 1. e4, but no other moves have been played.

```
edit
#
ra1 nb1 bc1 qd1 ke1 bf1 ng1 rh1
pa2 pb2 pc2 pd2 pe4 pf2 pg2 ph2
c
ra8 nb8 bc8 qd8 ke8 bf8 ng8 rh8
pa7 pb7 pc7 pd7 pe7 pf7 pg7 ph7
.
```

Note that input is free form, so piece/squares can be entered one per line or all on one line. Ditto for the #, c, and . special characters. Note also that there is no way to enter castling status here. It is far better to use the "setboard" command which uses a FEN-like syntax and allows you to set both castling and enpassant status.

15. egtb This command enables the endgame databases. Crafty will use the "tbpath" directory (if provided) to locate and register all of the databases you have downloaded. It will report the largest class it found, as in "5 piece tablebase files found" if you downloaded at least one 5-piece file. If you use this

command to enable databases, you should also consider using the "cache" command to specify the egtb cache size.

16. end|quit These commands are used to terminate crafty. Note that you can resume a game later without having to replay the moves, by starting Crafty using the "crafty c" command. It will immediately read in the moves for the last game, although you will have to set the time controls and clock time remaining yourself.

17. evaluation option <value> This command is used to modify the evaluation scores.

The option "asymmetry" is used to make crafty evaluate king safety differently for each side. "evaluation asymmetry 25" will increase the king safety scores for the opponent only, meaning it will pay less attention to its own king safety than to that of its opponent. This will make it play more aggressively. "evaluation asymmetry -25" will reduce the king safety scores for for the opponent by 25%, making it care more about its own king safety than that of its opponent. This will make it play more defensively.

The "bscale" option will adjust the scores for blocked pawns. The default value is 100. Increasing this will tend to make Crafty dislike blocked pawn positions more, which will lead to more open positions. Note that this only affects moves _after_ the opening book has been followed, which means that the position might be blocked before the evaluation term has a chance to affect the game.

The "kscale" option will adjust all king safety scores based on the 'value' entered. For example, "evaluation kscale 50" will reduce all king safety scores to 50% of their normal value. "evaluation kscale 133" will increase all king safety scores to 133% of their normal values.

The option "tropism" is used to scale king tropism scores. This will attract pieces toward kings. A value of 100 means no change. other values are treated as a percentage (like scale) to increase (> 100) or decrease (<100) the king tropism scores.

When you use this command, you will see something like this:

```
modified king-safety values:
white:   0   4  16  26  39  45  58  77  87  90  93  96 100 103 106 109
       112 116 119 122 125 128 128 128 128 128 128 128 128 128 128 128

black:   0   5  20  32  48  56  72  96 108 112 116 120 124 128 132 136
       140 144 148 152 156 160 160 160 160 160 160 160 160 160 160 160
```

Those values represent the king-safety evaluation as the king gets more and more exposed. This is always based on the fast that "crafty" will be the side on move when the search starts. In the above, it was white's move when this was typed, meaning that it appears that crafty will be playing black. Notice that white's king safety numbers are scaled by 20% to make it slightly more cautious about its own king. If you type "go" in this position, the scores get reversed as Crafty's scores are always left alone (with the asymmetry option) and the opponent's scores are scaled down as indicated.

You will see similar numbers (but not black and white sets) that represent the actual scores produced for king tropism. Note that pieces interact to choose which element of this vector is used, but in general, the more pieces are close to the king, the larger the element from this array.

The "pscale" option is used to scale normal pawn structure scoring in the same way as the other scaling options. 100 is the default. Values less than 100 reduce this term, values over 100 inflate it.

The "ppscale" option is used to scale some passed pawn scoring in the same way as the other scaling options. 100 is the default. Values less than 100 reduce this term, values over 100 inflate it. This mainly effects outside passed pawns/protected passed pawns. The normal pawn scoring computes the value of a passed pawn. This term is then used to scale those terms that modify this value further, such as two connected passed pawns on the 6th, or a passed pawn with the king supporting it in an endgame.

18. extensions type value

This command is used to control the extension depth for the various extensions done in Crafty's search. The extensions are set as decimel numbers that represent plies (or fractions of plies) to extend for each particular reason. Most default to 1.0 and .75, but any value can be used. Note that value > 1.0 are _very_ dangerous as they can cause the search to become non-terminating (crafty will stop when it runs out of time for the move, but it might not be able to get anything useful from such a search).

These extensions are presently limited to a maximum of one ply of extensions at any point in the tree. IE no matter what you set these values to, they can not exceed one ply at present.

incheck This is the amount to extend when the side on move makes a move that leaves the opponent in check. Note that Crafty extends on the ply where the check is played, not on the next ply where the opposite side is in check.

onerep This is the one-reply-to-check extensions, and is done at the point where one side is in check and has exactly one legal move to escape from the check.

pushpp This is the extension used for certain passed pawn pushes in the endgame.

recapture This is the recapture extension, and is applied when the current move recaptures an equal-valued piece that made a capture at the previous ply. IE BxN, PxB. Note that this can only be applied once for every two plies so that BxN, BxB, NxB, NxN won't look like three recaptures.

mate This is the mate threat extensions and is applied when a null move search returns -MATED, which means that doing nothing gets the side on move mated. The opponent must have some sort of serious mate threat in such a position.

19. flag on|off This command is used to force crafty to end a game where the opponent runs out of time with winboard/xboard (on) or to ignore this (off) if desired.

20. force [move] This command is used to force Crafty to play a move that is different from the one chosen and played by the tree search. If [move] is given, and it is a legal move, Crafty will retract its last move and make this move instead. It does not change the side on move, but does change the position of course. If [move] is not given, Crafty will prompt you for a move to make.

21. help This command displays multiple pages of one-line help, one command per line. If a line ends with [help], then you can use help followed by the specific command to get detailed help.

22. history This command displays the history in a vertical column with one move for white and one per black per line. There are other ways to display the current game moves and also to save them in files that are explained later.

23. hash=x and hashp=x These commands are used to adjust the size of the hash tables in Crafty. hash modifies the size of the transposition/refutation table, while hashp modifies the size of the pawn structure/king safety hash table. The sizes may be entered as one of the following two types of values: nnnK where nnn is an integer indicating how many Kbytes Crafty should use for this hash table; nnnM where nnn is an integer indicating how many Mbytes Crafty should use.

The transposition/Refutation table is the most critical of the two, because it directly affects search efficiency, particularly in the endgame. For this reason this should be maximized. The most effective size for this hash table is 3/4 of your available memory. If you don't know how to figure this out, but know that you have 16 megs for example, they you can say hash=16M and crafty will round that down to 12M, which is 3/4 of a power of two size. If you study the sizes that are possible, you will find 3M, 6M, 12M, 24M, 48M, and so forth. Anything up to, but not including, the next size will be rounded down to the next lower size. hashp should be set to approximately 1/2 of what is left. For example, the P6 Crafty runs on when playing on ICC often uses hash=48M and hashp=8M. The only thing to watch for is that if you make this too large, particularly under windows, performance will suffer badly because of paging I/O overhead. When Crafty is searching in a normal (non-book, non-endgame database) position, the disk light should *not* be on, indicating lots of I/O.

There is no danger in making this table too large, although you have to watch out because if Crafty barely fits in memory, doing something else on the machine can cause Crafty to be swapped out completely or partially, depending on the operating system you are using. If you are going to use the machine for anything else while Crafty is running, it is better to "pretend" that the machine only has 1/2 of the memory it actually does when computing the size of the hash tables you want to use.

24. import <filename> [clear] This command is used to import any sort of learning data that Crafty supports, which currently includes book learning data and position learning data. This command reads the appropriate <filename> and imports that learned data, just as though Crafty had learned it by playing the games. The [clear] option, if specified, caused all old learned results to be cleared before the import operation, otherwise the imported data is simply added to what is already present.

25. input <filename> This command is used to redirect the console input I/O stream from the keyboard to a file. Crafty will then read commands from this file, rather than from the keyboard, and execute them just as though they were typed in. Such a command file *must* be terminated by an "exit" command (no quotes) as the last command in the file. This reverts the input stream back to the keyboard, and prompts you for another command or move.

This command might be used to configure crafty for a specific time control, by putting the appropriate time control commands in the file, or to customize the hash table sizes as needed.

26. info This command is used to display information about Crafty and the current game. Such things as the time control, the time left on the clocks and other information is shown.

27. learn n controls the learning facilities in crafty. Currently this is a 3-bit boolean switch, bit 1 (001) controls book learning, bit 2 (010) controls position learning, and bit 3 (100) controls result learning. learn=0 disables all learning, learn=1 enables book learning only, learn=2 enables position learning only, and learn=4 enables result learning. Add the values together to turn on more than one type of learning (default=7 to enable everything).

28. level <m> <t> <inc> This command was taken directly from GnuChess so that the Xboard/WinBoard interface would interface with Crafty. There are other better ways to set the time, but this one is well-known. The three parameters are <m> (number of moves in the game) <t> initial time on the clock. After <m> moves are made, this amount of time is added to the clock again. <inc> is the Fischer-Clock increment that is added back to each clock after a move is made. It may be zero for a non-increment game.

Examples:

level 0 5 0 (ICS 5 0 game)
level 0 5 3 (ICS 5 3 game)
level 0 15 30 (ICS 15 30 game)

29. list GM|IM|C|AK|S +name [+name ...] -name [-name ...] This command is used to maintain the internal "lists" Crafty uses to auto-tune itself when playing on a chess server. There are three lists, GM, IM and C. If Crafty's opponent is in any of these lists, Crafty adjusts internal controls that affect how/when it resigns or offers draws, and how randomly it will choose moves from the opening book. For example, Crafty resigns much sooner against a GM, because it assumes he knows how to win a rook-up ending, regardless of how much time is left. By the same token, when playing against computers, Crafty will always assume that a draw is 0.000, so that it doesn't wreck its position trying to avoid repeating a position.

The AK list will automatically kibitz scores/PV's if the opponent is in this list. The S list will turn on special scoring for opponents in this list. The only current member is "mercilous".

The syntax +name1 +name2 simply adds these players to the specified list. To remove a name, use -name1 -name2. You can use one command per name to remove or add, or you can use one command to add and remove multiple names. Note that all names must be entered in lowercase characters, using any uppercase characters will break the matching algorithm.

30. log off|on|<n> This command is used to disable logging. The default is log on, which causes crafty to produce a new log.nnn file for each game played. If you are running Crafty on a server, you might use log off, which disables creating these files as well as the game.nnn files used to restart a game after you exit crafty and come back later. If you use the form "log n" crafty will simply display the last n lines of the log on the screen. If you use "log n file" crafty will copy the last n lines of the log to "file" which could be your hard drive, or a floppy.

Note that if you run with log off, you will be unable to find out what Crafty was thinking about since there is no other record of the game. You will always see a game.001 because as crafty plays a game, this contains all the real moves played so far so that you can back up if needed. you will also see a log.001 file, but it will be empty.

31. ls <filename> will list all the files that match the filename wildcard (the wildcards depend on the system you are using, but generally *, ? will work fine. you can also supply path information in the filename if you want to list the contents of a different directory. Just use the same syntax you would if you were using "ls" under unix or "dir" under windows.

32. mode tournament|normal This command is primarily used to put Crafty into "tournament" mode, which is intended for use when Crafty is playing in computer chess events. It accomplishes two things: (1) makes all draws return a score of 0.000, and (2) makes crafty issue a message after each move showing the internal chess clock time, and requesting that that operator check and adjust as needed using the "clock" command. This primarily makes Crafty comply with computer chess rules that say the operator can't do anything not specifically requested by the program.

33. name <name> This command is an ICS-play specific command. Xboard/WinBoard uses this to inform Crafty of the opponent's name. Crafty uses the name, and looks it up in its GM/IM/C lists, and if found, adjusts itself accordingly. This is not used by the PGN code and this will not cause the players <name> to show up in the PGN tag section.

34. new This command wipes everything out and starts a brand new game. It closes the old log-file and game-file, and opens the next sequential numbered file. It also resets the game to the beginning and prepares to start a brand new game. This was added for Xboard, but it turns out that Xboard does not use this, rather it starts Crafty fresh for each new game by first terminating the old copy then starting a new one. Not nearly as efficient as using "new" but likely safer it a program can't be sure of resetting everything back to the initial state.

35. noise <n> This command sets the "noise" level in Crafty. Basically, until <n> nodes have been

searched, crafty will be silent and not display analysis.

This is useful in two ways. First, in end-games, 20+ ply searches are not uncommon, and the search analysis for the first few plies arrives so quickly that it is distracting. Second, when observing games (new interface only) on ICS servers, this can be used to avoid having Crafty generate too many analysis kibitzes. A value of 100000 will basically shut off any output for the first second or so (on a P6/200). Similarly, 1000000 will eliminate any output for about the first 10 seconds. When watching and kibitzing games like the World Championship games on ICC, I generally use 5000000, which is almost one minute of silence so that the first PV it kibitzes is a pretty deep search.

noise 0 will cause *all* analysis to be displayed, which on a fast machine causes no problems. On a slower machine, or over a slow phone connection, this might cause a big communication backlog. The default is roughly one second on a P6/200 (100000) but can be modified by this command.

36. operator <n> Another command intended for use when Crafty is playing in a tournament, operated by a human. This tells crafty to "hide" <n> minutes of time and not use them. This time is basically allocated to the operator to make up for the time it takes to type in moves and/or correct mistakes. At the WMCCC events, the normal value we use is 5. Playing on a server, this is not needed, as it is not needed if you are playing Crafty yourself.

37. perf This command is primarily used in optimizing Crafty, or to test the speed of the move generator and MakeMove()/UnMakeMove() on different platforms. It produces two results, the moves it can generate per second, and the moves is can generate and make/unmake per second. While this is not a perfect performance indicator, it does give an "approximation" for how fast Crafty might run. In general, the higher the numbers, the better the program will play, although machines are certainly different. It's not uncommon to find a machine that searches slower than another, but has a higher "perf" value.

38. perft <depth> This command is generally used to confirm that the move generator and bitmap operators are working properly. It simply takes the current position, and generates/makes/unmakes moves and counts them. Many programs use this from a "standard" position to make sure that their move generator does not miss generating odd moves like enpassant/promotions and also to confirm that the make/unmake code correctly updates the board so that the totals remain constant across different machines and programs, since there is no alpha/beta or evaluation things done. if <depth> is greater than 5 or 6, it will take a *long* time, since this is basically a minimax tree traversal that will visit *every* node within the <depth> search horizon.

39. pgn <tag> <value> This command is used to set the usual PGN tags to meaningful values. The recognized tags are Event, Site, Round, Date, White, WhiteElo, Black, BlackElo, and Result, and the tags *are* case sensitive. The <value> can be any valid input and blanks and special characters are allowed. Note that the date is clearly specified in the PGN standard and must be yyyy.mm.dd with no variance. Valid results are 1-0 (white won), 0-1 (black won), 1/2-1/2 (drawn) and * (unknown). Some examples:

pgn Event 14th World MicroComputer Chess Championship
pgn Date 1996.10.8
pgn Site Jakarta, Indonesia
pgn Round 1
pgn White Crafty
pgn WhiteElo 2400
pgn Black assassin
pgn BlackElo 2400
pgn Result 1-0

Setting these values will result in a proper PGN file when using the savegame command. Note that if you use the "read" command to input a PGN game, these values will be extracted from that game if they are given.

40. ponder off|on|<move> This command serves two purposes. First, it can be used to disable (off) or enable (on) thinking on the opponent's time (or pondering as it is called in many programs including Crafty.) Turning it off will weaken Crafty since it will not use any machine time while waiting on the opponent to move. It is sometimes useful, however, when playing Crafty against another computer and the machines are not equal. If crafty is on a faster machine, and you attempt to adjust for this by giving the opponent more time than Crafty, it doesn't work quite as expected, because while the opponent is thinking, so is Crafty, which lets it use the extra opponent time in an unexpected way. In such a case, it's best to stop pondering in both programs.

If <move> is given, it directs Crafty to use that <move> to ponder, rather than the one from the previous search. Most commonly this is used to set the right move to ponder after Crafty has been stopped and then restarted, assuming that it is the opponent's turn to move when this happens. Otherwise, it is probably better to not try to influence things, although if you are watching and suddenly wonder "what would Crafty do if the opponent plays move 'X'?", you can answer this by simply typing "ponder X" and then watching the analysis. You should reset the correct ponder move after you do this of course.

41. reset <n> This command lets you back up in the current game to any move of your choice. reset <n> backs up the game to move <n> with the same side on move. If you want to first change the side to move, use the white/black command, then use the reset command to back up to the right move. Note that you can also go forward as well, just so there are moves in the current game history.

42. resign <n> This command sets the resign threshold. When running on ICC I typically use "resign 9" which will make crafty resign roughly five moves after the score drops below -9.000. For IM's I change this to 6, and for GM's I often use 3, so that it will resign quicker and not drag a lost game out unnecessarily.

43. read/reada [<filename>] This command will read input, and extract the chess moves and make them to set up the position at the end of the game. It first resets the chess board to the initial position (read command only) and then extracts the PGN tags (if present) from the front of the input. The rest of the input is parsed for chess moves (comments and similar things are culled automatically) and the moves are made and added to the game history. Once this is done, you can back up, go forward, or play from any point in the game. If you specify a <filename> everything is read from the file, otherwise it is read from the console keyboard.

The reada command reads moves, but appends them to the current game history/ position rather than resetting to the initial chess position. This lets you read in a game, then use reada to manually add some more moves to see the resulting position.

44. savegame <filename> This command is used to save the current game in a PGN-compliant file with the PGN tags included. Note that the default TAG values might not be what you want if you do not either use the pgn command to set them or else input a valid PGN file with the tags already filled in.

Be aware that this command doesn't check the filename for legality since anything goes in UNIX. In DOS, you might produce a bad filename with either too many characters, too many periods, or whatever, so be careful with the name you choose. Note also that this file will be overwritten if it already exists, so be sure to choose a name that is not the name of a file that has something you consider important in it.

45. savepos <filename> This command writes a single line into <filename> in FEN-like notation. This lets you save a position, and then come back later to re-examine it. You would use the "in <filename>" command to input this file and set the position up.

46. search <move> This command allows you to specify one particular move for the side on move, and then when you tell Crafty to search this position, this is the only move that will be searched. This is used internally by the annotate command, but can be used to investigate one specific move. If the move is not the best move, a normal search won't show you why it is bad, but this will. It is also quite a bit faster since the other moves in the position are not searched at all.

47. settc <moves> <ctime> <otime> This command is primarily used in tournaments, and is an error-recovery command. If the machine crashes and corrupts the game history file, frequently the operator will have to simply set the position using the setboard command, and then use the settc command to restore the time control values. <moves> is moves until the next time control (from Crafty's perspective, be careful and don't look at the opponent's moves to time control by accident.) <ctime> is minutes left on Crafty's clock, <otime> is minutes left on the opponent's clock.

48. setboard <FEN input> This command is used to set a chess position up for analysis and is the preferred way to do this, rather than using the gnu EDIT interface. It uses a classic Forsythe-like notation to encode the position and also has provisions for castling status and enpassant capture status.

the standard piece codes p,n,b,r,q,k are used to denote the type of piece on a square, upper/lower case are used to indicate the color of the piece (uppercase=white pieces, lowercase=black pieces).

the pieces are entered from the classic chess diagram's orientation of a8 being the upper-left-hand corner of the board, and this square is entered first, followed by the remainder of the 8th rank left to right. To indicate empty squares, use a number between 1 and 8 to indicate how many adjacent squares are empty. use a / to terminate each rank after all of the pieces for that rank have been entered. Note that you do not have to account for all 8 squares on a given rank, although many test suites do this for clarity.

the following input will setup the board position that given below:

    k2r/ppp////Q/5PPP/7K/ B

this assumes that k represents a white king and -q represents a black queen.

```
-k  *  * -r  *  *  *  *
-p -p -p  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *
 q  *  *  *  *  *  *  *
 *  *  *  *  *  p  p  p
 *  *  *  *  *  *  *  k
```

* the field after the final "/" should be either b or w to indicate which side is "on move." after this side-to-move field any of the following characters can appear to indicate the following: KQ: white can castle king-side/queen-side/both; kq: same for black; a1-h8: indicates the square occupied by a pawn that can be captured enpassant.

49. score This command simply gives the positional score for the current position. This score is from whites perspective, so a + score is good for white, a - score is good for black. Crafty also breaks the score down into major categories (from Evaluate()) to indicate things like pawn structure, piece evaluation, passed pawns, development, and so forth. Note that some of Crafty's evaluation is asymmetric, so that if you simply change sides with the white/black command and then enter "score" again, you may get a different value. This is *not* a bug. :)

50. sd <n> This command lets you specify a specific search depth limit that Crafty can not exceed. It still

pays attention to the clock, however, so often you will use the st <n> command (below) in conjunction with this if the search is going to take an extended amount of time. <n> is the depth (in plies or 1/2 moves) that the search must reach. Note that if Crafty is pondering, it still honors this limit and will stop a ponder search after this depth has been completed as well. This is *not* the way to make Crafty play weaker, although this will be covered in a later section of this document.

51. show <category> This command forces Crafty to display additional information about certain actions it takes. Currently the only <category> is "book" which will make crafty display information about all the book moves it found in the database. More is given about this information in the BOOK section later in this file.

52. smpmt=n This command is used to set the number of threads to use on a machine with more than one processor. For optimal performance, "n" should be set to the number of processors you have, although using fewer will reduce the load on your machine. For this command to work, Crafty must have been compiled with SMP defined. When compiled with SMP enabled, mt=0 effectively disables the SMP code completely.

This command also has two that are closely related. smpmin and smpmax. Both accept single numerical arguments. smpmin is used to control the minimum tree depth required at a node for it to be eligible for parallel searching. IE smpmin 2 says don't split unless at least two more plies are left to search below this node. smpmax sets the maximum for the same idea, is smpmax 10 says don't split if more than 10 plies are remaining below this node.

53. sn <n> This command is similar to the sd command, but instead of setting a specific search depth, it sets a number of nodes to search. Once the search has searched this number of nodes (+ maybe one more second of searching to check the time) the search will exit.

54. st <n> This command lets you specify a specific search time limit for Crafty. Again, this is not the preferred way to set a time per move, because this limit is absolute and Crafty will never go over this limit, even if it sees that it is losing or getting mated. Setting the time control with the usual "time" or "level" command is *much* better. <time> is given in seconds, although it may also be entered as mm:ss if preferred.

55. swindle on|off This command gives you control over "swindle mode." When on, and playing a game, Crafty will try to win drawn endings (according to the tablebases) if it has winning chances (like KR vs KB, for example). This will put up very stiff "resistance" to accepting the draw, while with this mode off, it may be very easy to draw a position once the tablebases say "drawn." This mode is automatically turned "off" during analysis or when annotating a game, and is only used when actually playing a game against an opponent. If there are no tablebases then this has no effect on the game at all.

56. tags This command will simply display the current PGN tags (you can edit them with the various PGN commands).

57. test <filename> [n] This command will run a suite of positions (the file must be in "Crafty" format as explained below) and produce a summary of how many it got right, wrong, etc. It uses the time per move you set with the (typically) st <n> command. The optional parameter [n] is the "early exit" counter. If Crafty finds, and holds the solution move for n iterations, it will terminate the search. I use this to make a win at chess run take < 15 minutes, even though the time per position is set to 1 minute, by setting n to 2. After two correct iterations, Crafty goes on to the next problem. For absolutely correct results, this is not advisable as it could obviously change its mind later on, but for performance analysis, this saves a lot of time.

The test suite contains the following lines: (this is a sample from my test suite for Win At Chess.)

title wac299
setboard 1n2rr2/1pk3pp/pNn2p2/2N1p3/8/6P1/PP2PPKP/2RR4 w
solution Nca4

title wac300
setboard b2b1r1k/3R1ppp/4qP2/4p1PQ/4P3/5B2/4N1K1/8 w
solution g6

end

The title command simply displays this message in the log file so you can look at the output and figure out which position it goes with. This is optional, but useful.

The setboard command sets the position as explained before.

The solution command gives the set of solution moves (one or more moves that are separated by blanks) and/or a set of "anti-solution" moves (moves that must not be played to count the position as correct.) "anti-solution" moves are simply followed by a "?" character, for example:

solution Bxa7?

The solution command supplies a set of key moves, and then starts the search. If, after the search terminates, one of the key solution moves was chosen (or none of the anti-solution moves were chosen) the position is counted as correct.

The final line should be "end" although end-of-file or EOF will also be detected in this case.

57. time CPU|elapsed|<values> This command controls whether the program uses CPU time or wall-clock time for timing. for tournament play, it is safer to use wall-clock timing, for testing it may be more consistent to use CPU timing if the machine is used for other things concurrently with the tests being run.

time is also used to set the basic search timing controls. the general form of the command is as follows:

    time nmoves/ntime/[nmoves/ntime]/[increment]

nmoves/ntime represents a traditional first time control when nmoves is an integer representing the number of moves and ntime is the total time allowed for these moves. the [optional] nmoves/ntime is a traditional secondary time control. increment is a feature related to ICS play and emulates the Fischer clock where <increment> is added to the time left after each move is made.

as an alternative, nmoves can be "sd" which represents a sudden death time control of the remainder of the game played in ntime. the optional secondary time control can be a sudden-death time control, as in the following example:

    time 60/30/sd/30

this sets 60 moves in 30 minutes, then game in 30 additional minutes. an increment can be added if desired.

One final example is the following:

    time sd/15

which is a simple game/15 setting. This command can also be used to perform the same function as the "level" command. For example, to set up a classic ICS 2 12 game, the following would would work:

```
time sd/2/12
```

59. trace <n> This command is used to make crafty display the tree as it searches a position. Due to the incredible speed at which this program can search, however, this is less than useful since it can swamp any known display driver and make things scroll impossibly fast.

Also note that this command usually is disabled, because Crafty is generally compiled with the -DFAST flag, which removes the trace output code from the search to make things slightly faster. You will have to recompile, without the -DFAST, if you want to use this. It's utility is limited, except for debugging, anyway.

60. usage <n> is simply a way to modify Crafty's time usage to fit your tastes. You can "suggest" a time limit with any of the options discussed previously, but if you use anything other than the "st" command, Crafty will do its best to use time as you suggest, but it also anticipates that it will save some time by pondering, etc., and will therefore be more aggressive at trying to use time. if <n> is a positive integer, it is taken as a percentage and crafty will compute the time limit it thinks is appropriate for the current clock settings, then increase this limit by this percentage (50 would make it take 1.5 times longer than normal.) -50 would make it take 1/2 the time it would normally take.

Crafty adjusts the usage internally based on time left, opponent's time left, how quickly or slowly the opponent is moving, etc. Further modifying things with this is dangerous, but possible.

61. whisper/kibitz <n> These commands are used to control what Crafty will whisper or kibitz on a chess server. The options are (1) only whispers or kibitzes mate announcements; (2) adds time, score, depth to the previous option, but no PV or moves. (3) adds the PV. (4) adds book move information to the output. The remaining two options generate a lot of output and should be used with caution. (5) displays the PV after each iteration completes. I use this when using my custom interface to let Crafty observe/comment on games in progress on ICC. Noise can be used to prevent shallow searches from generating output and keeping "noise" down on the games being watched. (6) basically will whisper/kibitz nearly everything you see on the console from a search, each PV when it changes, fail highs and fail lows, etc. A significant amount of output that should be carefully weighed before turning it "loose."

62. xboard This command turns on Xboard/WinBoard compatibility mode, and makes Crafty behave somewhat like GnuChess. This is designed to be used *only* when Crafty is interfacing with Xboard/WinBoard. Crafty will not work with these two GUIs without this option, and really won't work very well with this option if it is not connected to one of them.

63. There are other commands that are not documented. They are part of the xboard protocol and really should not be used by the normal user. You can find all the commands in option.c should you be interested.

Opening Book Setup and Usage

Crafty uses two pre-compiled opening books, called "book.bin" and "books.bin".

The file book.bin is usually build from a large text file containing PGN games, often taken from collections of GM games. Building book.bin is a simple exercise and requires nothing other than the raw input file you are going to use. Generally this will be either medium.zip or the set of four files large[1-4].zip, all of which are stored on the ftp machine ftp.cis.uab.edu/pub/hyatt/.

To create the file book.bin, you need a PGN game collection that is a *real* PGN-compliant file.

Supposing this file is called "large.pgn" you would use the following command:

book create large.pgn <m> [n] [wpct]

The only thing you have to supply is <m>, a number indicating how many moves from each game are to be stored in the book.bin database. I typically use 60, which stores the first 60 moves from each game. Increasing this number slightly increases the probability that Crafty will stay in book longer, but it also increases the probability that it will follow a game too far, so that it begins to reach positions where the move actually played might not be the best move, letting it fall into a bad hole. Increasing this also increases the size of the database as well.

You can decrease the size of the book, and also reduce the number of ugly moves by specifying <n>, which says that unless a move is played in at least N games, the move is discarded. This will substantially decrease the size of the book.bin file, and also eliminate single game moves that often have significant errors or blunders.

You can increase the quality of book lines by specifying <wpct> which is the "winning percentage". This is specified as a percentage of lost games, and is used to discard moves that led to mostly losses. A safe value is 50, which says that if a particular opening move didn't win at least 50% as many games as it lost, the move is culled. A value of 100 would mean that moves are culled if they produced more losses than wins, and is really a strict criterion.

After creating book.bin, you need to create books.bin. This is a small version of book.bin, which is intended to give you more control over the moves/openings Crafty will play. This is usually built from the file start.pgn on the ftp machine, but you can modify this file to suit your taste easily. To build books.bin, you use the following command:

books create start.pgn 60

Again, 60 is what I use, but none of my start.pgn lines go anywhere near that many moves. The main point here is that in start.pgn, you can append a "!" to any move you want, and when it is Crafty's turn to move for that color, it will play from the set of moves with "!" if there are any, ignoring the rest of the book moves. If you only want it to play 1. e4 as white, you would just enter the short game:

[Crafty only plays 1. e4] 1. e4!

and you are finished!. You can enter as many as you want. If on the other hand there is a move you don't want Crafty to play, then follow that move with a "?" and it will never play it. Moves in books.bin that are not flagged with ! or ? don't have any influence on Crafty's choice at all.

Here's how the files are used. When searching a position, Crafty first enumerates the set of moves it can find in the opening database. It then does the same for the books.bin database, and performs a "merge" operation to combine the ? and ! flags... The purpose of the books.bin file is to give you a small database that you can easily modify, rebuild, and repeat this process over and over. Since it takes a fraction of a second to completely rebuild this file, it is very easy to modify this to control what Crafty will play, without having to rebuild the large database.

One important characteristic of the PGN input is the Result tag must be specified in most of the lines, because Crafty counts wins, losses and draws for each book move and uses these counts with some of the book selection options given below.

How the flags are used.

The ! and ? flags should only appear in the small books.bin file, although there is no reason why they can not appear in the large file as well. For this discussion, it doesn't matter since Crafty takes the moves from both files and "merges" the flags/etc into one entry for each move.

Quite simply, if any book legal book move has a ! flag, then Crafty will only play moves from the set of moves which all have the ! flag. If a move has a ? flag, then that move is immediately removed from the set of possible book moves. If the only legal book move has a ? flag, it will not be played as a book move and Crafty will simply pretend that it found no book moves and will execute a normal tree search. Pretty simple.

How to control the frequency of opening move selection.

A new feature in version 15.15 and beyond allows you to append a PGN comment to any move in a text file used to create books.bin, of the form {play nn%}. This will force the move it follows to be played that percentage of the time, regardless of the normal book-ordering values based on frequency and so forth.

Note that {play 0%} will not prevent a move from being played at all, as this will look just like a move with no percent specified. To avoid playing a move, use the ? flag.

How does Crafty choose book moves?

Crafty's book selection algorithm depends on two user-defined values that can be set at any time during a game:

book random <n>

book width <w>

The selection algorithm first finds the set of legal book moves as above. This set will either be all ! moves, or will have no ! moves in it. This set is then sorted based on the setting of book random. Here's the options:

book random 0. This is a special case for book selection. Crafty simply takes the set of book moves, and searches only these moves using a normal alpha/beta search, but with a shorter than usual time limit. It then plays the move that produces the best search value. This has one serious disadvantage in that there is no randomness to this at all. It will always play the same move in the same position, unless the evaluation is modified, or the time per move is different enough to let the search find a different move from the book move set.

book random 1. This enables a random form of book move selection, but you have a lot of control over how moves are randomly chosen. The moves are ordered, based on 4 parameters: frequency of play, win/lose ratio, static evaluation and learned results. Normally these are factored into the value used to sort the moves, based on default settings that you can modify by using the command "bookw option N" "option" should be "freq", "ratio", "eval" and "learn". N should be a number between 0 and 1.

Crafty finds the min and max values for each of the 4 parameters, and then maps this into the range 0-1000 for each parameter. Each parameter is multiplied by the corresponding "weight" you have assigned, and this is used as a value to sort the moves from low to high. Note that the first sort value is always the "play percent" to move them to the top of the list. For moves with equal "play percent" values, the normal sort-value is used as the second-level sort variable (if no moves have a play-percent, then this second-level variable is the only one used, of course.)

Once Crafty has sorted the moves as given above, it next excludes any book moves which have 0 wins.

This culls the odd lines where a player chose a bad line and lost quickly. With zero wins, it will never be chosen, although Crafty will happily follow it from the other side. :) This is not anywhere near perfect, however, because an opening could have 1 win and 19 losses and that still would stay in the list.

If a move has a learned value of > 100, this move is elevated in priority to that of a ! move, since it appears to win material instantly. If a value is < -100, it is given a ? since it appears to be a lemon.

After this, the setting for "book width <w>" is used to keep the first <w> moves in the list, after the above culling has been completed. The smaller you make <w> the less randomness you get, but the chance of Crafty popping out a really bizarre book move gets smaller as well.

After sorting, the final step is to fold in any mandatory "play percent" values. What this step does is that it finds all the moves in the "playable move set" just computed, which have no percent-to-play value set. It sums the sort-values for these moves, then adjusts the sort-values for the other moves so that their percentages will be honored.

Once this has been done, crafty simply uses the "sort value" for each move to compute a total for all moves. It then generates a random number between 1 and this limit, and chooses the move that this probability distribution matches. This will certainly evolve over time as new ideas are developed.

For my play on ICC, I use book random 1, and book width 5 normally, although for titled players this is reduced to book width 3. For computers, I reduce this further to 2, to try to play reasonable openings and cull the gambits and things that don't work out.

How does book learning work and how can I share learned results?

1. *all* of crafty's "learned knowledge" is in the book.bin file. It keeps the learned value and learned count right in the book file for speed. You can't modify it, although "show book" will make crafty display all the book stuff before it makes a move.

2. the book.lrn file has two purposes: (a) to serve as a log for your prying eyes, so you can see what it's learned, against whom, and what the score was that got its attention in the first place. The values on the end of each book line, inside the {} characters are as follows:
{value, depth, rating_difference} value is the value of the "key" search that comes from the first 10 moves out of book. it's in centipawns, and + is good - is bad. depth is the depth the search reached at this "key" position, the deeper the search, the more the value is "trusted." rating_difference is crafty's rating - opponent's rating a negative value means pay more attention to the score since the opponent is better than crafty, a positive value means to take the score with a grain of salt, because the opponent was weaker than Crafty.

You can delete this file at any time, and it has no effect on learning. As I mentioned, the learning takes place in book.bin... this is mainly for you to peek at if you are interested. However, this is the "portable learning data file" also, and can be given to others to import into their crafty, where it will affect the opening book just like their crafty had played the openings and got the same scores. There are two ways to use such "lrn" files:

1. "import <filename>" will read <filename> and import the knowledge therein into your book.bin. Since I use the same learning code as is used when playing games, this information also gets appended to *your* book.lrn file as well, so that your book.lrn always reflects *everything* your program has learned, so long as you don't ever remove this file. It would be a mistake to use this command on your own book.lrn file, because the things would get counted twice, which might or might not be a good thing.

2. "import <filename> clear" will read <filename) and import the knowledge as above, but first clears *all*

learned results from book.bin. you will want to do this if you import my book.lrn, *and*, you have contributed to my book.lrn data by sending me yours. I'll take care of eliminating duplicates if you screw up in what you send me, but once you send me something, you run the risk of getting it "back again" later. This is going to be a problem until everyone gets used to sharing something that is rather "vapid" like this "learned info" is...

Other than that, we are now "open for business"... when the urge strikes you, email me your .lrn file, I'll keep a large master here and update it on occasion. Probably the best thing to do is to send me your .lrn and at the same *instant* delete yours. This will capture anything learned *after* you send me the file, but you'll get it all right back with the next version of book.lrn that I distribute. after getting this new book.lrn back, here's what you should do:

1. rename your old book.lrn to something else. I'll call it "book.lrn.old" here.

2. copy my blearn.dat to your machine, but *do not* put it in the directory with your book.bin and books.bin file because it will get confusing very quickly if you do. put it somewhere else, because you are going to remove it quickly anyway. I'll call it new.lrn for this example.

3. import new.lrn clear
   import book.lrn.old

and you are ready to rumble again. The first command clears the learned values, sucks in my new learn file and updates everything. the second command re-learns everything you've learned since you sent me the last book.lrn file. After doing this your book.lrn will have my .lrn stuff, plus your old.lrn stuff, just waiting to be sent to me again...

If this is confusing, I can probably add an automatic command to do all of this by renaming book.lrn, etc. Hopefully this is not too error-prone for the time being anyway...

What is this new Position Learning I've heard about?

Crafty now has a "permanent" hash table that is kept from game to game. A position gets into this "hash file" when Crafty executes a search and the search value is significantly lower than the last search value.

When this happens, Crafty stores the current information for this position in the permanent hash file, which can hold up to 65536 positions. Once it fills up, the positions are replaced on a FIFO basic always keeping the most recent 64K entries.

Each time crafty starts a search, the positions/scores from this file are stuffed into the normal transposition table, and used during the search just like any other table entry. Here's how it helps: In a game that was played, the following moves and scores were found by crafty (playing white):

1. Ng5 (+.277) h6 2. Nh7 (+.321) Kg8 3. Qh5 (+.133) Qg7 4. Ng5 (-2.122) hxg5

So, the knight got trapped at h7, and at move 4 crafty discovered that this is gross and "learns" this result/position.

We play the exact same game again: except that two things can happen here. It might be that Ng7 is the *only* square the knight can move to here, which means this whole thing is forced. the first search would find:

1. Ng5 (-2.122) if the search can reach 8 plies deep, which happens even in 5 second games. It's learned that Ng5 is bad. It stores *this* position in the permanent hash file also, and the next time you try this same

trap, it will discover 4-5 moves earlier that if the knight gets to g5 it is in trouble. Each game will diverge from the first game 3-4 moves earlier. Simple and effective.

2. Ng5 might not be forced, and if not, it knows Ng5 loses a piece for a pawn, so it will promptly play something else, which is exactly what is desired.

This is implemented with two (count 'em, two) files. One file "position.bin" is a binary file that contains the hash table entries, and it right at one megabyte in size, *max*. (16 bytes per hash entry X 65536 entries = exactly one meg, but I have 8 extra bytes for the FIFO queue implementation and to see how many entries are currently in the file if it is not full.

The second file is "position.lrn" and is, you guessed it, a file that can be shared with others, just like book.lrn. It contains all information needed to reconstruct the position, the score, the depth, etc. and also included the pgn tags for who was what color and when the game was played...

This data can be imported with the new "import" command (the old book learn <filename> is no longer around) which will import either book.lrn type data or position.lrn type data and can tell them apart without your having to do anything. The <clear> option is still there, should you want to use it, and simply removes the position.lrn and position.bin files before starting the import process for position learning.

This can be turned off, if you like, by checking out the "learn" command, which gives you the ability to turn off book learning (as it presently works), position learning, and the next book learning stage which will add to the book in addition to learning which book lines are good and bad.

What is this new "result" learning?

Result learning works just like normal book learning, except that if Crafty is checkmated or resigns, it will step back through the book line to find the last point where it had more than one move to choose from. It will flag the move it chose as "never play again".

This handles the case where the first ten non-book moves produce reasonable scores, but the position is one that Crafty simply can't handle very well. If it loses such a game, it will still vary the next time this opening is played, as otherwise it would possibly repeat the same opening, and would certainly repeat the remainder of the game.

All three learning modes are turned on by default, although any of them can be disabled with the appropriate command option to "learn".