Chess Tests: Notes on Problem Solving

Timings and discussion on "Solved"

 Last update: 01/10/97

Notes on time measurements:

- **For mate problems**, you should note the time when your program actually *STOPS* and *GIVES* the mate, *NOT when it merely sees the mate*.

- **For non-mating problems**, you should note the time when your program's evaluation for the move *is quite similar to the value given here*, NOT simply when it sees the move *but assigns it a significantly distinct value*.

Notes on when a test position is to be considered SOLVED:

My test suite is still growing, and I carefully select each position so that they demonstrate something about the programs solving or trying to solve them. They are mostly *very difficult*, and most of them cannot be easily solved within 3 minutes, which is what I would consider as a reasonable time for actual game-playing.

But notice that in some of my tests (i.e. **Test 81**) all of the programs tested find the correct move, however I do *NOT* consider any of them to have solved the problem, because their evaluation of the position shows that they don't see the draw, and do not even know how to *maintain* the draw afterwards.

In my opinion, a *meaningful* test should include not only *time* and *solution*, but also the minimum value the evaluation should have to consider that the program has *really* found the solution.

For instance, say *test XX* should be solved within *10 minutes*, it should find *Nb3*, *and* it should have *Eval >= +5.00* points because that move wins the queen. If the program *does find Nb3* within 10 min., but evaluates the move as *+0.32*, then the program has not seen that the queen is won, and certainly has *NOT* solved the problem.

Perhaps if let alone for another 20 minutes it would then evaluate it as *+5.12* in which case *it would have solved it*, but *not* with *Eval = +0.32*.

Same goes, as another example, for certain drawn positions (see **Test 81**), where white can draw enclosing the enemy king in a *fortress*. The promoted queen *cannot* mate the white king alone, and so the position is drawn.

If the program evaluates this as *Eval =-9.00* because of the queen, the program is not seeing the draw and is not understanding the position at all, so much in fact that many times it will *ruin* the position even after making the correct first move. Thus, it *cannot* be considered to have solved the problem, either.

Conclusions:

Taking all of this into account, I hereby advocate that test suites should include for each and every test position, not just the desirable move or moves, the moves to avoid, and some time limit, but also the minimum value the desirable move must have to consider the program has *really* seen the consequences of the move, and is not chosing the right move *for the wrong reasons*.

Failure to take this into account could result in some or all of these sad scenarios:

- **Program XX passes certain test with better timings than program YY.** However, that obscures the *essential fact* that the slower evaluations of YY were *much more accurate* than the quick results of XX.

- **Program XX finds the correct move for a certain test.** However, letting the program play that position, it quickly degrades it and eventually *loses* any advantage, because the real features of the position which justified that move *were never understood nor its consequences foreseen*.