Conway's Soldiers Extension

James Collerton (46114)

December 10, 2014

I have submitted two different versions of the extension as I attempted two seperate tactics and I would appreciate having them both considered. The first extension is based on optimising the level search and the second is based on changing the ordering of the list of boards dependent upon a weighting.

Once the submission is unzipped each is contained in its own folder, run the pagoda implementation using ./pagoda.

Extended Version One (extended.c)

- Time to reach $(4, 7)$ with conway.c: $375s$ $490ms$
- Time to reach $(4, 7)$ with extended.c: $35s$ $430ms$
- Needs files: makefile, neillsdl2.c, neillsdl2.h, extended.c, extended_functions.c, extended_functions.h

I incorporated a couple of different methods in order to optimise the level search. First of all I recognised that solving for tiles on the right hand side of the board is an identical problem to solving for the left hand side, and so limited searching to one half of the board. When either the regular or symmetric solution was found it was printed accordingly.

I also limited the search for identical boards to within levels and so only looked at boards with the same number of pegs. Boards were added with the basic method so those with the same number of pegs are found next to each other and the search is only conducted until the first board with a different number of pegs is found.

Finally I attempted to introduce a hashing element. I couldn't quite manage a perfect hash but got quite close. My method was to sum all of the pegs in the first row, then the second row, then the third until I reached the top row. Having done this I summed all of the pegs in the first column, then the second

column, then the third until I reached the end column. I then concatenated these digits into a single *unsigned long long* number which formed the hash key (I simultaneously created a symmetric version).

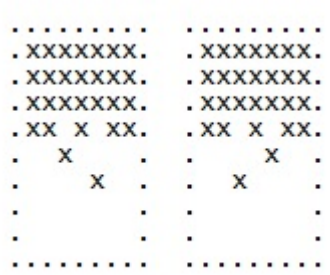

Figure 1: Example boards with same hash number (444444400115777)

After finding the key I took the number *mod* the largest integer I could find that could be used to specify the size of an array. This provided an index for a hash table. These components then formed the basis of the duplicate checking system.

1. Check to see if the index of the hash table represented by the hash key *mod* the large number was occupied. If not then the board was unseen.
2. If 1. indicates the board has been seen then go back through the boards from the end checking the hash keys. If the hash key is never found then the board is unseen.
3. If 2. indicates the board or symmetric board has been seen then manually check the board to confirm.

I appreciate that there are methods to make larger arrays which makes the hashing more effective, as well as methods to fill the array with pointers to boards rather than indicators of having seen the board before, however I thought it was better to focus on the next method, the Pagoda weighting.

Extended Version Two (pagoda.c)

- Time to reach $(4, 7)$ with conway.c: $375s$ $490ms$
- Time to reach $(4, 7)$ with pagoda.c: $0s$ $10ms$
- Needs files: makefile, neillsdl2.c, neillsdl2.h, pagoda.c, pagoda_functions.c, pagoda_functions.h

The pagoda weighting system can be found documented here:

http://arxiv.org/pdf/math/0612612.pdf

I have weighted the boards using this method and arrange them in the list accordingly. They are then searched through in this order until the solution is found, with new boards being constantly added according to their weighting. Symmetry and hashing are both still considered but have much less of an impact than in the previous version, symmetry especially as the weighting favours the actual solution rather than the mirrored version. Note that this method will reach $(3, 8)$ and $(5, 8)$ on the top row if given a small amount of time for the latter.

Although this is a lot quicker than the previous method it does have the disadvantage of having potentially messy solutions. Some moves increase the weighting of the board without actually moving towards the desired position and so when the result is displayed these boards are included and the result is not optimal in that sense.