Charles Weng

112334040

CSE 526 - Final Project Report

ICLP 2020 Programing Contest

Part III. Implementation

1. The source code and implementation can be found on github (8). This is, however, a private repository. Contact me at charlesweng123@gmail.com if you need access to the files.
2. Overall implementation was not hard, but understanding how the Clingo module worked took up a lot of time.
    a. Since I have also not used Clingo (or logic programming for that matter) before this class, I am not sure what many of the included Clingo modules are meant to emulate. The documentation (4) was not too helpful in this regard.
    There was a context object keyword argument that let Clingo execute external functions, which ended up being a pile of headaches. I could not find a use for external functions so this was deleted in the end.
    b. Overall I was surprised at how small the program actually turned out to be. The state of the art I found was a lot longer and harder to understand.
        i. For skyscrapers, the state of the art (7) muddied the solver with a lot of GUI and some extensions I can't identify. Even in his Puzzle.cpp

file (which I assume is his main file for the solver), it is almost 700 lines long and I have a hard time interpreting the results. The Clingo implementation ended up being a lot more straight-forward and overall easier to understand. Even including my comments and generous spacing, I only used around 150 lines of code and the tests I ran all came back instantly.

ii. For Aquarium, the state of the art (5) was discussing the problem via a blog. It was good at explaining how to tackle the problem, ~~but I could not figure out what language the author was using to develop his code~~. (It was Perl, but I still can't understand the intricacies). He implemented it in chunks and in the end solved what I assume to be a 30x30 in ~1s with about 350 lines of code (light commenting). Even including my comments and generous spacing, I only used around 120 lines of code and the tests I ran all came back instantly.

iii. For Masyu, I did not run any comparisons with the state of the art. My implementation, even though it's only ~120 lines of code, turns out to be very slow. The constraints, which were not too hard to represent in Clingo, did not happen to be well integrated with each other and needed a lot of extra things to be defined. The 25x25 case took 4 and half minutes to compute and the smaller 15x15 took 6 seconds. I am not sure how I can improve this implementation.

Part IV. Testing and Evaluation

1. Due to not being able to easily generate puzzles with their solutions (for an NxN puzzle, I can create the input file in O(N) time, but I will have to verify solutions by inputting the O(N^2) entries into the website), I could not test too many examples. For all the ones I have tried (that yielded a solution), it came back instantly. Even the 30x30 aquarium puzzle given in the contest had the solution printed as I hit enter (in less than .05 seconds).
    a. Skyscraper - this was hard to test as I could not find decent puzzles that fit my constraints (a good portion of the puzzles included some prefilled buildings on top of missing hints). Since 3x3 puzzles are almost trivial (2 2 is the only pair of clues for a row/col that can appear and not have a unique solution) and I could only reliably find 6x6 puzzles, I can only conjecture that this one is worse than quadratic (my testing N would not even double). I graphed 4 specific cases for N=4-7 averaged over 100 runs and this does seem to be the case.
    b. Aquarium - this seemed to be somewhere between linear and quadratic (at least for N=6,15, 30). I think this will more closely follow O(N^2) since the constraints are a lot simpler and straightforward. I have graphed 5 specific cases for N=6,15,20,25,30 averaged over 100 runs and it appears to be linear.
    c. Masyu - this seemed to be very slow (possibly exponential). Having to ensure edge constraints by itself already takes a while to compute, but the

connected constraint really kills it (I used the reach function from class). I mentioned runtimes in the implementation section, but to reiterate: 6x6 took .05 seconds, 15x15 took 6 seconds, 25x25 took 4 minutes; larger sizes were not tested, but the ICLP contest went up to 40x40.

Part V. References (APA6 Format)

1. Åhlander, M. (2020, May 08). Skyscraper puzzle with constraint module in python. Retrieved May 12, 2021, from https://stackoverflow.com/questions/61662722/skyscraper-puzzle-with-constraint-module-in-python
2. Alviano. (n.d.). Alviano/lpcp-contest-2020. Retrieved March 20, 2021, from https://github.com/alviano/lpcp-contest-2020/blob/master/README.md
3. Aquarium. (n.d.). Retrieved March 23, 2021, from https://www.puzzle-aquarium.com/

(new) 4. Clingo. (2021). Retrieved May 02, 2021, from https://potassco.org/clingo/python-api/5.5/clingo/index.html

5. E. (2020, April 02). Aquarium - constraints. Retrieved April 25, 2021, from https://github.polettix.it/ETOOBUSY/2020/04/02/aquarium-search/
6. I. (n.d.). Lp/cp programming contest. Retrieved March 20, 2021, from https://iclp2020.unical.it/affiliated-events/lpcp-programming-contest
7. Pcooksey. (n.d.). Pcooksey/skyscraper-ps. Retrieved March 21, 2021, from https://github.com/pcooksey/skyscraper-ps

(new) 8. Weng, C. (2021, May). Iclp2020project. Retrieved May 02, 2021, from https://github.com/unicomputing/iclp2020contest