Madison Everett and Omar Martinez

Dr. Hooper

Artificial Intelligence

5 December 2018

Dominosa: Interactive Game and Problem Solver

Within this project, we examined the results of the Backtracking Search Algorithm with various implementations heuristics in the constraint- satisfaction problem, Dominosa, also known as Domino Solitaire. Dominosa is a domino board logic puzzle that involves a square board of varying sizes that contains several numbers that each correspond to a unique domino piece. The goal of the puzzle is to place the corresponding domino pieces on their respective places on the board without having two of the same domino pieces on the board. Like many other constraint -satisfaction problems, Dominosa holds real -world significance due to the implementations that it has within not only Computer based fields but also within other fields. According to Abderrahmane Aggoun, a Computer Science Researcher, CSP problems have effectively addressed a large class of combinatorial problems, such as the optimization of pallets. In addition to this, another scholar, Yarong Chen, a Computer Science PhD Student, explains that CSP “enables the implementation of precise, flexible, efficient, and extensible scheduling systems.” Because of these real -world applications, we decided to attempt to find a faster way of solving Dominosa, which in turn could result in a faster way to solve real-world CSPs. In order to start this project, we applied the Backtracking Search algorithm in order to compare the resulting amount of moves to discover its overall efficiency regarding the board and problem size. We also used several Constraint-Satisfaction related heuristics in order to continue to increase the overall efficiency of this problem solver. Along with this, we also created playable version of this puzzle that can interact with the varying difficulties to compare the moves and efficiency of the AI to a human player. By using this comparison, this resulted in the AI finding and producing more efficient solutions than the human player.

Our first step in this described process was to understand our CSP toy problem, Dominosa. Dominosa is one of many Domino related puzzle games. According to the Simon Tatham’s Portable Puzzle Collection, this puzzle is accredited to O.S. Adler, hence the “-osa” part of Dominosa. Within the puzzle, the player is given a board of numbers that each correspond to a specific domino piece that exist within a set of dominos. For instance, the board may consist of a 2 and 4 next to each other, therefore the domino piece (2|4) may be a part of the total solution. The goal of the puzzle is to find a solution of dominos that satisfy the number placement on a board. However, the only constraint is that each domino piece within the solution must be unique. This constraint is due to there being no duplicates within any given domino set. This case of uniqueness also includes the alternates of each domino piece. This means that (2,4) and (4,2) are the same domino piece. Since Dominosa is an example of a Constraint-Satisfaction Problem toy problem, we decided to attempt to create a solver for it in order to allow it to solve more complicated problems that a human player would normally be unable to complete.

In order to initially begin build this project, we chose use Python due to the reusability of code that exists within it’s files and classes. After making this decision, we continued to code and develop the base interactive game. Within the game, the player is offered a menu that allows them to choose what they would like to do. The options range from playing a Dominosa board to allowing the computer to solve any chosen Dominosa board within the game. Starting with the interactive version, the player is allowed the choice of several different Dominosa boards within various difficulties to solve for entertainment. In order to create the boards, we used arbitrary pieces from our double six domino set, which allows for 11 different board options with 4 different difficulties. Along with this, as discussed before, player can also select a board from the board library for the computer to solve.

In order for the program to solve these various boards, we used the Backtracking Search Algorithm explained within our course’s text, *Artificial Intelligence,* by Russell and Norvig. Our reasoning for this algorithm is due to research performed by Francesca Rossi, an IBM AI Ethics Global Leader. Within her studies, she explains that the Backtracking Search Algorithm is the most important CSP algorithm in practice and it is also is a complete algorithm that has several benefits, such as “removing inconsistencies during search that can dramatically prune the search tree by removing many dead ends and by simplifying the remaining subproblem” (p. 83). Therefore, we decided to solve to the Dominosa boards via a Backtracking Search Algorithm due to the benefit it could hold for solving Dominosa boards. However, in order to our Dominosa boards to be compatible with the algorithm, we first had to make each of our created puzzle boards into CSPs, using the CSP class provided within Russell and Norvig. According to this CSP class, each CSP requires a domain, a set of neighbors, and constraints. Because of this requirement our next steps were to create a domain, neighbors, and constraints for each Dominosa board.

In order to tackle this requirement, we first started with finding the domain of each individual Dominosa board. According to Sally Brailsford, a Professor of Management Science, CSPs are a type of problem that “requires a value selected from a given finite domain, to be assigned to each variable.” This means for our CSP to run correctly, each board must have a finite domain that the Backtracking Search algorithm can use to assign values to the solution. From this, we realized the domain of each individual board is simply the total possible pairs that can be created within the board. This theory was reaffirmed by Vipin Kumar, a Computer Scientist author, who explains that the number of combinations within a CSP’s domain is “the size of the Cartesian product of the variable domains.” By following Kumar’s advice, we found a finite set of domains for each board by finding the Cartesian Product of each number within the board. For example, if the number we started with on the board was 4 and it was surrounded by 2,3,4, we could treat these variables as sets, whereas Set A is {4} and Set B is {2,3,4}. By finding the Cartesian Product of these sets, we would get a set of domino pieces that are {4,2}, {4,3}, and {4,4}, which would be a subset of the overall domain. By using this method for every number on the board, the finite domain for each Dominosa board was created.

After completing the domain, we then had to complete the neighbor and the constraint portion of the CSP. In the case of a CSP, a neighbor is what piece or pieces are directly next to one particular piece on a board and a constraint is whatever rule the chosen algorithm must follow in order to obtain a valid solution to a problem. Within Dominosa, each piece has their respective neighbor and the constraint used was the different-values constraint to ensure that two of the same pieces were never next to each other. However, this led to a problem within our solution for the Dominosa boards. Since the neighbors assigned to the Domino pieces were only the Domino pieces adjacent to each other, the Backtracking Search Algorithm could easily assign two of the same domino pieces on the board. In order to solve this problem, we decided to make every domino assignment the neighbor to every possible assignment within the board. By doing this, it forces the Backtracking -Search Algorithm to never assign two of the same domino pieces on the board, which allows the pieces selected for the solution to always be unique to each other.

After completing the Dominosa CSP, the CSP could now be solved using the Backtracking Search Algorithm. However, there are numerous CSP related heuristics that we used in order to attempt to increase the efficiency of the Backtracking Search Algorithm. Some of these heuristics included MRV (Minimum Remaining Value), MAC (Maximum Arc Consistency), Forward Checking, and LCV (Least Constraining Value). We chose to primarily use these heuristics because they were discussed with Russell and Norvig and they were closely paired with the Backtracking-Search Algorithm and CSPs, therefore we believed that we would gain the best results from using these heuristics.

In order for the user to witness the Backtracking Search Algorithm in action, they are given a choice within the menu to see the computer complete the board. Within this selection, the player can select what difficulty and board the algorithm will solve from the board library. Then the program will print out one solution from the Backtracking Search Algorithm without using heuristics and another solution from the Backtracking Search Algorithm using heuristics. Both solutions will also print how many moves the algorithm had to perform in order to find a valid solution. We then took these results and compared them to the results of a human player playing the same board.

From this process, the Backtracking Search Algorithm performed extremely well in solving the problems compared to the human player. The success and efficiency of this program was purely based the number of assignments and moves that the Backtracking Search Algorithm produced. For an easy 2x3 board, and hard 4x4 board, the Backtracking Search Algorithm without heuristics performs similarly to a human player with 9 assignments and 58 assignments respectfully. However, within the added heuristics of LCV, MAC, Forward-Checking, and MRV, the Backtracking Search Algorithm’s efficiency increased from 9 assignments to just 4 assignments for the easy board and 58 assignments to just 9 assignments for the hard board. This efficiency increase reveals how helpful not only the Backtracking Search Algorithm is for CSPs but also how helpful these heuristics are to further improve the speed getting a solution for this problem.

In conclusion, the Backtracking Search Algorithm is an extremely helpful algorithm to increase the overall efficiency of a Constraint – Satisfaction Problem. This Dominosa program reveals the benefits that the Backtracking Search Algorithm provides for problems that require the assignment of certain variables. In future work, we hope to develop the program further to increase the overall efficiency of the Backtracking Search Algorithm. We also hope to implement the Forward-Checking Algorithm and Conflict Directed Back-Jumping Algorithm to this program to continue to see the efficiency of these algorithms compared to the Backtracking-Search Algorithm in the case of this puzzle. After continuing to develop the program, we also hope to further to expand the program include different problem instances that are more in depth than Dominosa that will have a greater impact on the Computer Science world.

References

Aggoun, Abderrahmane. “Panorama of Real - Life Applications in Logistics Embedding Bin Packing Optimization Algorithms, Robotics and Cloud Computing Technologies.” *HAL Archives*, 30 Nov. 2016, <https://hal.archives-ouvertes.fr/hal-01378469>.

Brailsford, Sally C, et al. “Constraint Satisfaction Problems: Algorithms and Applications.” *European Journal of Operational Research*, North-Holland, 13 Jan. 2011, [www.sciencedirect.com/science/article/pii/S0377221798003646](http://www.sciencedirect.com/science/article/pii/S0377221798003646).

Chen, Yarong, et al. “Technology and System of Constraint Programming for Industry Production Scheduling - Part I: A Brief Survey and Potential Directions.” *SpringerLink*, SP Higher Education Press, 26 Aug. 2010, link.springer.com/article/10.1007%2Fs11465-010-0106-x#citeas.

Rossi, Francesca, and Toby Walsh. “Constraint Programming.” *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, edited by Peter van Beek, 1st ed., Elsevier, 2006, pp. 83–117.

Tatham, Simon. “Chapter 17.” *Dominosa*, Simon Tatham's Portable Puzzle Collection, 2014, [www.chiark.greenend.org.uk/~sgtatham/puzzles/doc/dominosa.html#dominosa](http://www.chiark.greenend.org.uk/~sgtatham/puzzles/doc/dominosa.html#dominosa).

Kumar, Vipin. “Algorithms for Constraint-Satisfaction Problems: A Survey.” *Association for the Advancement of Artificial Intelligence*, Mar. 1992, www.aaai.org/ojs/index.php/aimagazine/article/view/976.