Yelp Group Decider

CS 182 Final Project

Guy McEleney and Sathvik Sudireddy

December 10, 2018

1 Introduction

The purpose of our CS 182 Final Project is to use Constraint Satisfaction Problems (CSPs) in order to effectively allow groups of people, such as a Harvard Blocking Group, to decide on a place to eat together without having to think too hard or get into unnecessary arguments. We feel this is an ideal problem to solve using CSPs based on research we did on how CSPs can be used to decide more complex, multidimensional issues where many constraints might need to be considered, specifically when there are many different variables (people in a group, in our case) [3]. Our CSP application is unique to the field of artificial intelligence and specifically CSPs because we are trying to satisfy multiple constraints in order to have only one correct response/option from our given domain. Instead of having each of the variables being assigned to different items from the given domain, we are specifically targeting our domain and narrowing down from all the constraints until we can assign the same singular element from the domain to each of the variables. We believe the purpose of this project, for us, goes beyond the unique application of CSPs but instead on how artificial intelligence can drastically change the experience of deciding where to eat as group, where there could be many strong opinions, decision-paralysis, and hurt feelings. Our application takes away the stress and conflicts associated with making food choices.

Our goals are to use what we learned in class and through our own outside research on artificial intelligence to logically, efficiently take in multiple users' preferences and constraints and in return narrow down the choices to ideally return one best suited option. We used a lot of other universities' resources to better comprehend which algorithms we were going to use, including

(1) forward checking, (2) simulated annealing, and (3) a new algorithm loosely based on backtracking. One of our other main goals was to figure out how to manipulate CSPs to best solve our problem of deciding where to eat. We realize from our own lectures that CSPs are frequently used to assign values to many different variables, such as the map coloring assignment example [6]. We wanted to take the idea of backtracking, forward checking, and simulated annealing, and instead of having such restricting restraints for what the algorithms were to give as outputs to instead allow for more flexibility and customizability for each party using the application.

For this project, we had to define the scope of what we were going to accomplish pretty early on in order to make this project feasible. To start, we decided to use Yelp's API, Yelp Fusion, in order to find restaurants with all of the data associated with each place. Yelp Fusion, unfortunately, limits how many items can be returned in a search to only 50 locations, so we knew we were only going to be able to test up to 50 restaurants at a time. A quick note on our experience with Yelp Fusion: we initially started with Yelp's public dataset but had a very hard time narrowing it down to something feasible for this project and time frame. We then chose to work with Yelp Fusion instead. In general, we spent a significant amount of time trying to comprehend Yelp's options and how to integrate them into our project. Thus, there was a large setup time and learning curve for our project before we could even consider implementing CSP algorithms. Also, we made the choice for the scope to make this project targeted at Harvard students, faculty, and residents; therefore, we chose to base the search queries from Harvard University's Lowell House (the house we both live in) and also limited the radius for searches to 10 miles–a more realistic range for students to travel. Additionally, as we were working through this project we had to decide on what were the constraints that were most important for groups of people when trying to decide where to eat. We ended up with scoping the constraints for users to restaurant ratings, price, distance, and cuisine (both what you like and don't like). Beyond what the users can put in as their preferences, we also decided to constrain on whether the restaurants were currently open because that would be a requirement for a group to be able to go to that restaurant.

In regards to the algorithms we tested and implemented, we had to make changes from the published, readily available algorithms–such as Forward Checking, Simulated Annealing, and Backtracking. The reasons for the alterations is because our CSP is meant to recommend a singular restaurant choice. This means that having an algorithm, like Forward Checking, that would immediately eliminate options when a constraint was broken and no longer could help satisfy the overall problem would not work for our needs [5]. We did try that approach in our second implementation but quickly learned that if no restaurant fits the given constraints, then our application would become useless if nothing was recommended. We also implemented a simulated annealing approach for this specific Yelp problem, but had similarly unhelpful results. Simulated annealing has far greater run-time than our other two algorithms because it runs through 1,000 iterations which makes it not ideal for trying to quickly solve this food-eating problem. Therefore, we had to create a different algorithm with a different approach where we instead took a similar, loosely-based approach to Backtracking but decided to keep track of how many constraints are broken per restaurant. Then, at the end we would return the restaurant with the least constraints applied because then we could still have a final recommendation that is customized and unique to that groups' requirements but in a faster run time–the best of the three approaches.

The range of problems addressed in this project are tailored around how to have many different variables with all different constraints but still return one final element that worked for all the constraints for each of the variables. These problems and the unique alterations to the published algorithms are important and offer great discussion and exploration around the ideas for CSPs as covered in CS 182: Artificial Intelligence–specifically in lectures 5 and 6, where the majority of our understanding of these published algorithms comes from [6] [7].

2 Background and Related Work

For our CS 182 Final Project, we looked at our own lives to look for pain points that artificial intelligence could effectively help solve our problems. We both found that deciding where to eat when you want to go to dinner with your blocking group or even with your family can be very stressful and testing of peoples' patience. Therefore, we decided to recreate how groups choose where to eat by using artificial intelligence's CSPs. However, we realized early on that our problem to be solved by CSPs is different than many other problems taught in class–such as map coloring, course plan generator, N-queens, or even Sudoku [6]. Our problem was harder in the sense that we would take in many variables' constraints and have to return a single element of the given domain that work for all given constraints.

When doing research, we did not find any examples of CSPs being used in the format that we have proposed and implemented. However, there are loosely related works that gave us inspiration on how to approach our algorithms and problem. We know that our approach for a CSP is still valid, given that many of the scholarly works on this topic directly indicate that the "list of applications of CSP is as broad as it is deep" [4, p. 244]. Even though we could not find specific projects solving a similar unary problem, we did find a related work on how to make a Limousine company less stressed and more successful. In Andy Hon Wai Chun's article on applying CSPs to the Limousine industry in China, he talks about that his system objectives were to "help planners and controllers handle more orders and manage more vehicles while maintaining high service quality," essentially using CSPs to ease the stress of the workers while allowing them to get quality outputs [1, p. 30]. Our problem, while not based in the Limousine industry, is based around trying to make peoples' lives easier while still giving them a high-quality output–all through using CSPs–a very similar goal to this researcher's application of CSP to Limousine companies. Based on these readings we found it might be in our project's best interest to have one of the implemented algorithms feature tuples of information to keep the data helpful and most dynamical for the problem, something that we first learned was used successfully in this Limousine project [1]. Reading through this article it was very reassuring that the value we had thought CSPs could return and offer was realistic and also could easily be applied to a multitude of industries and problems.

Additionally, we looked into other papers on CSPs that scholars had applied to many different complex problems. One of those problems that we felt similarly fit our project's problem was Fox and Sadeh's scheduling problem [2]. In their paper, they talk about how "a sequence of successively more complex scheduling problems can be modeled as a CSP," something we found to be similar to our situation when you start adding more people to the equation [2, p.1]. This is also the work that led us to consider what happens when no element in the given domain satisfies all the

constraints. In this paper they discuss how realistically the more people involved and the more constraints then the more likely nothing will work perfectly. It leads to them applying a relaxation of certain constraints in order to allow for the CSP to return some output [2]. This work on CSPs is, therefore, very insightful for us as we thought through and worked on our project because we similarly have many constraints that realistically result in no restaurant not breaking a single constraint. Then what do we do? Do we follow the example of this paper where they realized that a relaxation of constraints is needed? Or do we do something else? We discuss this problem and our decided implementation in detail later in our "Approach" section, but we acknowledge that Fox and Sadeh's paper originated our discussions and awareness on this topic.

3 Problem Specification

As per the definition of a CSP, our problem can be defined as follows:
Variables: The people in the group using this app
Domain: The potential restaurants that the group can go to, provided by the Yelp Fusion API
Constraints: Ratings, price, distance, cuisines. If there is a cuisine you prefer, the algorithm accordingly weights those cuisines higher.

4 Approach

For this final project, we decided on implementing three different algorithms, all based on published algorithms discussed in class and in the field of AI–but all with some alterations to fit our problem's specific needs.

4.1 Forward Checking

The first algorithm we implemented was Forward Checking, based on how we learned it in Lecture 5 where when an item in the domain does not satisfy the constraints of the variables then it is eliminated as a valid option [6]. We used lists for our data structure in the implementation to keep track of the restaurants remaining after constraints are applied. The only changes to the forward checking algorithm that we made to fit our problem was that if a restaurant breaks any constraint it is removed from the domain completely because our CSP is solving for one singular solution that works for all the variables–not instead trying to assign a unique element from the domain to each variable, as is written in our class's implementation explained in lecture. We did not incorporate preferences for certain restaurants in this algorithm, because we felt that it would eliminate too many restaurants. For example, if one person preferred Mexican and another person preferred Indian, all restaurants would be eliminated unless they satisfied both cuisines.

4.2 Simulated Annealing

The second algorithm that we implemented was Simulated Annealing, based on how we learned it in class during Lecture 7 and also loosely on how it was implemented during one of our problem sets [8]. In order to find the neighbor of a restaurant, we calculated a euclidean distance between restaurants. In order to do this we took the difference between ratings, price, and distance and summed them up to get a total "distance". We also assigned values to cuisine categories and

attempted to calculate a distance between restaurants based on cuisine category. We implemented simulated annealing similar to the pset. In order to calculate the "value score" of each restaurant, we counted the number of constraints on that restaurant and similar to simulated annealing, we switched to it if it was better and if it wasn't, we switched to it with probability proportional to our global temperature variable which decayed over time. Data structures used were an array of restaurant objects extracted from the Yelp API and a dictionary mapping cuisine categories to a integer.

4.3 Loosely Based on Backtracking

The third algorithm that we implemented was a new algorithm loosely based on how Backtracking was taught and explained during our class's Lecture 5 [6]. The main data structure used is a dictionary mapping restaurants to the number of constraints. Our algorithm then got constraints from the users and then used these constraints and added to the constraint value in the restaurant dictionary. We then sorted this dictionary and returned the restaurant with the lowest number of constraints attached to it. This algorithm is loosely based on backtracking because (as taught in Lecture 5) backtracking "checks constraints as it goes" and considers assignments based on results throughout the process [6]. Our version also checks as it goes and keeps track of all the assignments till the very end when it returns the least constrained option.

4.3.1 Approach for Third Algorithm

For our third algorithm which is loosely based on Backtracking, we had an additional dilemma that we needed to decided while implementing:

When trying to decide what our CSP should do when there are no restaurants after applying all the constraints from each variable, we contemplated multiple different approaches and many different algorithms.

Our first of our two main approaches for the third algorithm based loosely on backtracking was to loosen one or more of the constraints in order to allow for more restaurants to be considered. This would mean that if the minimum rating required was 5 stars, we could loosen it to be 4.5 or even 4 stars to allow for maybe a few more restaurants to be considered against the other constraints. This loosening would not be limited to just rating, but could also include price, distance, category loosening.

Our other approach is to choose the restaurant with the least amount of constraints applied to it by the given inputs from the different variables (people in the group). What this means is that if there were no remaining restaurants after all constraints were applied, then we would go through and return the restaurant that had the least constraints applied to it. For example, if Tatte was only "eliminated" because it was too far away, but fit all the other constraints, then we would hypothetically return Tatte if it had the least amount of categories that made it become eliminated in the first place.

We decided after lots of consideration to choose the second approach where we would default to the restaurant with the least amount of constraints instead of loosening constraints. We made

this choice because we both believe the chosen approach would result in more accurate, better fitting choices for the unique group. Our fear with loosening random constraints would be that whatever ended up being the choice might be arbitrarily chosen and not actually be the best possible choice for the specific group using our application.

5 Experiments

For our testing and experimentation, we decided to create four distinct groups with different ranges of constraints and needs. Then we tested each of those groups against each of the three algorithms we implemented.

5.1 Evaluation

Our first group was with two people and had the loosest overall constraints of the four (1.0 for minimum rating, $$$$ for maximum price, 10 miles for maximum distance, and no preferences for cuisines). Results for Group 1 can be found in Table 1.

Algorithm	Run Time (seconds)	Restaurant Output
Forward Checking	0.00026	Hot Box
Simulated Annealing	0.49	Sakana
New Algorithm	0.00025	Abide

Table 1: Results from Group 1

The second group was with four people and had a mixed level of overall constraints (3.5 for minimum rating, $$$ for maximum price, 3 miles for maximum distance, and do not want Italian, Thai, or Salad for cuisines). Results for Group 2 can be found in Table 2.

Algorithm	Run Time (seconds)	Restaurant Output
Forward Checking	0.0003	Hot Box
Simulated Annealing	0.5	Elmendorf Baking Supplies
New Algorithm	0.00042	Abide

Table 2: Results from Group 2

The third group was with 6 people and had slightly tighter level of overall constraints than the second group (4.0 for minimum rating, $$ for maximum price, 1.5 miles for maximum distance, and do not want Seafood, Greek, or Poke, Bubble Tea for cuisines). Results for Group 3 can be found in Table 3.

Algorithm	Run Time (seconds)	Restaurant Output
Forward Checking	0.00036	Hot Box
Simulated Annealing	0.47	Hot Box
New Algorithm	0.00028	Area Four

Table 3: Results from Group 3

The fourth group was with 8 people and had the tightest overall constraints (5.0 for minimum rating, $ for maximum price, .5 miles for maximum distance, and do not want Italian, Burgers, Cafes, or Mexican for cuisines). Results for Group 4 can be found in Table 4.

Algorithm	Run Time (seconds)	Restaurant Output
Forward Checking	0.00029	N/A
Simulated Annealing	0.47	Mainely Burgers
New Algorithm	0.00052	Beat Brew Hall

Table 4: Results from Group 4

5.2 Analysis

One insight that we found is that over all four tests, simulated annealing took significantly longer than our other two algorithms. This could be due to the fact that over 1000 trials, it is gradually moving between neighbors to find an optimal solution. While this algorithm may work for many problems, for our problem, it is clear that there are more efficient solutions. We found that forward checking was quicker for all trials, but with tight constraints, it may end up returning no restaurants. This is because forward checking is eliminating values from the domain as constraints are applied to it, and with tight enough constraints, it is possible that it eliminates every restaurant. Our new algorithm, while taking slightly longer than forward checking, always returns a restaurants as it returns the restaurants with the least constraints attached to it. This avoids the elimination problem while also being a much faster algorithm than simulated annealing.

5.3 Critique

Our first implemented algorithm is Forward Checking. We decided that this popularly used algorithm in the field might be a great choice for our problem of eliminating elements from the domain once they break constraints assigned by the group of people. This could be an ideal approach given that we want to narrow down the restaurant options efficiently and return the best fit option. Our forward checking algorithm works great specifically when the users' constraints are very loose (i.e. the users are open to any cuisine, price, distance, etc.), such as in our first test group; however, the moment that users become more specific in what they are looking for, the chance that there are no available restaurants that work with all given constraints becomes very high–especially given Yelp's limitation of only 50 results at a time. Thus, when no restaurants work, forward checking would eliminate every option and return no final recommendation. This

approach, therefore, is not ideal for our problem because we want to still leave a suggestion for the users on what would be a good place to eat for that specific group.

For our second implemented algorithm, Simulated Annealing, it did successfully return something each time; however, its run time is significantly longer than the other two algorithms. Therefore, the simulated annealing algorithm is not an amazing option for our designated problem because it takes too long.

Finally, when we were testing our third implemented algorithm (which is loosely based on backtracking), we ended up having our best results because of our approach with not eliminating restaurants when they break given constraints–but instead to count how many constraints are broken for each restaurant. This approach also was a lot faster than our Simulated Annealing algorithm when testing. We realize after conducting various tests that our third new algorithm (loosely based on backtracking) is the best overall choice for our problem because it is very fast and always works, always returning some restaurant option that best fits that groups' requirements.

6 Discussion

As we reflect on this final project, we both feel as though overall we have accomplished our goal we sought out–create a CSP that efficiently finds the ideal restaurant for a group of people to go to without inducing any arguments. After implementing three distinctly different algorithms, based on forward checking, simulated annealing, and a new one loosely based on backtracking, we found that our new algorithm based loosely on backtracking was the most optimal choice. In regards to always returning a restaurant choice, the forward checking algorithm we implemented was not always ideal, specifically when the number of people increased and the groups' constraints became tighter. When this would happen the forward checking algorithm would not return any restaurant–not what we were trying to accomplish! We also found simulated annealing to be too slow compared to other options we had.

We learned a lot while working on this project and have realized how applicable CSPs can be to many different, unique problems. However, based on our outside research and our own experience, there are certain times when specific algorithms would be better suited than others. For example, Simulated Annealing and Forward Checking might be great for solving other problems, but they are not the best option for our problem.

For us to think about in the future, we could imagine even better improvements to our application. One additional issue we think could be great to consider are dietary issues. Another issue we would have liked to strengthen if we had the time and resources, would be to assign ideal weights to each of the categories based on users' experiences over an extended period of time. For example, we could implement Machine Learning techniques in the future that account for what users really consider hard constraints versus softer constraints (i.e. price might really matter while cuisines one doesn't want are maybe looser concerns for a person).

We know this application is useful and helpful because everyone we have talked to about this

project has expressed real interest in this solution for their friend food decision-paralysis. Overall, we are proud of this project and its use of CSPs.

7 Appendix

7.1 System Description

You can grab our python files from https://github.com/sathvik22/CS182FinalProject. Our project consists of three separate files: cs182final.py, forwardchecking.py, and yelpsim.py. cs182final.py is what we consider to be our optimal algorithm while the other two files describe forward checking and simulated annealing respectively. First, you will need to run 'sudo pip install requests' in your terminal. You can run each of these files by opening your terminal and running the command: 'python [filename]' with the filename of the file you want to run. The program will present you with instructions which you can then follow. If you test with multiple people, the same following questions will be asked after you have filled them out for the first–essentially, it will loop through the questions as many times as there are people you have in your group. Our program may not check for invalid input so please make sure to follow the instructions precisely. The output of our program should be a restaurant name–the recommended restaurant for your group.

7.2 Group Makeup

While working on this project, Guy and Sathvik worked together on all aspects of this final project. In regards to specifics, we worked on implementing the three algorithms and setting up the Yelp API together at the same time–splitting between coding and researching what might be needed to make the code work. For the final report, we similarly worked on each part together–writing each part together to make sure that it correctly covered all the material needed.

References

[1] Andy Hon Wai Chun. Optimizing limousine service with ai. *AI Magazine*, 32(2):27–41, 2011.

[2] Mark S. Fox and Norman Sadeh. Why is scheduling difficult? a csp perspective. https://pdfs.semanticscholar.org/1e6f/bb23749af2731c926d235883e612abcee5c5.pdf, 1994.

[3] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[4] I. Miguel and Q. Shen. Solution techniques for constraint satisfaction problems: Foundations. *Artificial Intelligence Review*, 15(4):243–267, 2001.

[5] Brian C. Williams. Solving constraint programs using backtrack search and forward checking. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16$_4$10F10$_l$$ec$06.$pdf$, 2010. *Accessed* : $2018 - 11 - 30$.

[6] Haifeng Xu and Scott Kuindersma. Lecture 5: Constraint satisfaction problems. https://canvas.harvard.edu/courses/42715/files/folder/Lecture2018. Accessed: 2018-11-20.

[7] Haifeng Xu and Scott Kuindersma. Lecture 6: Constraint satisfaction problems ii. https://canvas.harvard.edu/courses/42715/files/folder/Lecture2018. Accessed: 2018-11-20.

[8] Haifeng Xu and Scott Kuindersma. Lecture 7: Local search. https://canvas.harvard.edu/courses/42715/files/folder/Lecture2018. Accessed: 2018-11-20.

[9] Yelp. Yelp fusion api. https://github.com/Yelp/yelp-fusion. Accessed: 2018-12-01.

[10] Stanislav Zivny. *The complexity of valued constraint satisfaction problems*. Cognitive technologies. Springer, Heidelberg, 2012.