# Assignment 2

## CMSC 471 (03/01) — Artificial Intelligence

| **Item** | **Summary** |
|---|---|
| Assigned | Friday February 26th |
| Due | Friday March 12th, 11:59 PM Baltimore time |
| Topic | Constraint Satisfaction Problems |
| Points | 90 (+20 Extra credit) |

In this assignment you will gain experience with CSPs: algorithms/heuristics for solving them, and experience casting a problem as a CSP (that can be solved by a dedicated CSP solver).

You are to *complete* this assignment on your own: that is, the code and writeup you submit must be entirely your own. However, you may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Piazza discussions).

The following table gives the overall point breakdown for this assignment.

| **Question** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **Points** | 15 | 25 | 25 | 25 (+20 Extra Credit) |

**Language and External Resources** You may use whatever language you prefer, however we must be able to call your program as specified in the assignment. If necessary, we must be able to compile it as well. You may use and reference code from `https://github.com/aimacode`, though acknowledge if you do so. Failure to do so, or the use of other external resources without prior written permission from the instructor, will be considered an academic integrity violation and result, in a minimum, in a 0 on this assignment.

**What To Turn In** You must turn in two items:

1. A writeup in PDF format that answer the questions.
2. A single `zip` or `tar.gz` file containing all code, environment files (if applicable) and execution instructions necessary to replicate your output.

As part of your submission, be sure to include specific instructions on how to build (compile) your code. Answers to the following questions should be long-form. Provide any necessary analyses and discussion of your results.

**How To Submit** Submit the assignment on the submission site:

https://www.csee.umbc.edu/courses/undergraduate/471/spring21/01_03/submit.

Be sure to select "`Assignment 2`."

## Questions

1. (**15 points**) In the following sub-parts, "briefly describe" means provide a description of a problem in a couple of sentences such that the constraints are clear, and identify the variables, their domains, and what aspect of the problem they correspond to. For example, if you were to "briefly describe" the map coloring problem from the slides, an acceptable answer would be, "Color the regions of a map with different colors, drawn from a fixed, finite set of colors, such that neighboring (touching) regions do not have the same color. Each region is a finite variable, whose domain is the set of available colors."

   (a) Briefly describe a CSP where the variables have a finite domain. You may not use map coloring.

   (b) Briefly describe a CSP where the variables have an infinite, but discrete, domain.

   (c) Briefly describe a CSP where the variables have a continuous domain.

2. (**25 points**) From Ch 4.12 of Poole and Mackworth, answer question 6. For clarity, I've copied it below. You do not have to follow their linear output requirement in part (a), but your answer must convey the same information.

   Consider a scheduling problem, where there are five activities to be scheduled in four time slots. Suppose we represent the activities by the variables $A, B, C, D$, and $E$, where the domain of each variable is $\{1, 2, 3, 4\}$ and the constraints are $A > D, D > E, C \neq A, C > E, C \neq D, B \geq A, B \neq C$, and $C \neq D + 1$.

   (a) Show how backtracking solves this problem. To do this, you should draw the search tree generated to find all answers. Indicate clearly the valid schedule(s). Make sure you choose a reasonable variable ordering.

   (b) Show how arc consistency solves this problem. To do this you must:

   - draw the constraint graph;
   - show which elements of a domain are deleted at each step, and which arc is responsible for removing the element;
   - show explicitly the constraint graph after arc consistency has stopped; and
   - show how splitting a domain can be used to sove this problem.

3. (**25 points**) From Ch 4.12 of Poole and Mackworth, answer question 12. For ease, I've copied it below (except for the figure, and with two clarifying changes, in italics).

   Consider the constraint graph of Figure 4.16 with named binary constraints. $r_1$ is a relation on A and B, which we can write as $r_1(A, B)$, and similarly for the other relations. *This question has you work through two steps of solving this network with VE.*

   (a) Suppose you were to eliminate variable A. Which constraints are removed? A constraint is created on which variables? (You can call this $r_{11}$).

   (b) Suppose you were to subsequently eliminate B (i.e., after eliminating A). Which relations are removed? *Which variables are in the scope of this new constraint?*

4. (**25 points, +20 Extra Credit**) This question has you formulate the "exact cover" problem as a CSP. There is some starter Python code available at `https://www.csee.umbc.edu/courses/undergraduate/471/spring21/01_03/materials/a2/code` but you are not required to use this code (or Python, even). The starter code assumes the use of the `python-constraint` library we went over in class. However, for this assignment, you may use code provided by the `aimacode` github organization: `https://github.com/aimacode`. There is a `csp` file, which contains the implementation of different CSP algorithms. If you are coding in Python, this will be `csp.py` in the `aima-python` repo. You are welcome to use these pre-implemented search functions, but you are not required to.

   The exact cover problem is as follows: Given a set of $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$ of values, and a set $\mathcal{S} = \{S_1, S_2, \ldots, S_J\}$ of $J$ subsets of $\mathcal{X}$, identify $K$ elements of $\mathcal{S}$ such that they are all pairwise disjoint, and the union across them all is $\mathcal{X}$. For example, given $\mathcal{X} = \{1, 2, 3, 4\}$ and $\mathcal{S} = \{S_1 = \{\}, S_2 = \{1, 3\}, S_3 = \{2, 3\}, S_4 = \{2, 4\}\}$, then $\{S_2, S_4\}$ is an exact cover, but neither $\{S_2, S_3\}$ nor $\{S_3, S_4\}$ is an exact cover. (Another exact cover is $\{S_1, S_2, S_4\}$.) Note that if you added $5$ to $\mathcal{X}$ but kept $\mathcal{S}$ the same, no exact cover would be possible.

   The specification for a given exact cover problem requires $\mathcal{X}$ and $\mathcal{S}$: these are provided by a JSON file. This JSON file is a dictionary with two keys: `vertices`, a list (set) of elements, and `subsets`, a list (set) of lists (subsets of vertices). See

   `https://www.csee.umbc.edu/courses/undergraduate/471/spring21/01_03/materials/a2/code/simple_exact_cover.json`.

   for an example encoding the above. We *will* test your program on examples you won't have seen. You may assume that the elements of $\mathcal{X}$ are hashable, but do not assume they will be `ints`. For full credit, your program must work appropriately for these other files we test.

   (a) In your writeup, formulate (precisely, with formulas but without code) the exact cover problem as a CSP. Specifically, given an arbitrary exact cover specification:

   - define the variables (including their domain and meaning with regard to the exact cover problem);
   - write out the constraints. For each constraint, provide a brief written justification for why your formulation is a correct formulation for that constraint. (You may have multiple constraints that are essentially the same, except for a couple of values. In that case, justify each *type* of constraint.)

   (b) Implement your CSP, and make sure you can save your solutions out in a JSON file. If you are using the starter code, the `write_solutions` function saves your solutions in the appropriate form.

      (i) Once you've implemented the constraints, programmatically test them: make sure the "solution" your program returns satisfies *your* constraints. How you do so is up to you, but provide the output (e.g., screenshot, or copy the terminal output) and include it in your writeup.

      (ii) Verify that your solution is correct. Run
      `python3 check_solution.py your_solutions.json`
      on the file your implementation generated. You don't have to turn anything in for this part.

(iii) Demonstrate that your solver can find the correct answers by applying it to the `simple_exact_cover.json` and `simple_exact_cover_no_solution.json` examples provided, using a backtracking solver. Save the resulting solution file as `q4-b-yes-backtracking.json` (for the json with a solution) and `q4-b-no-min-conflicts.json` (for the json without a solution) and include these in your submission.

(iv) In your writeup, describe (in writing) how you implemented the variables and constraints.

(c) Now add support for the `min-conflicts` solver. Demonstrate that your solver can find the correct answers by applying it to the `simple_exact_cover.json` examples provided. Save the resulting solution files as `q4-c-yes-min-conflicts.json` and include this in your submission.

(d) **[Optional: +20 Extra Credit]** Use the script `generate_exact_cover.py` to automatically generate some problem files. You can generate problems with no answer, problems with a unique answer, and problems with multiple answers. You can also choose the size of $\mathcal{X}$. In this problem, you will be generating a bunch of problem files, running your CSP implementation on them, and then analyzing the differences (both in solutions and time required) for backtracking vs. min-conflicts. Specifically:

- Pick a reasonable maximum size for $\mathcal{X}$. It should be at least 5, though I'd recommend no greater than 10. Let this value be $M$.
- For each size of $\mathcal{X}$ between 2 and $M$, generate at least three problem files for each of the "none," "unique," and "multiple" settings. This step should result in at least $(M-1)*3*3$ files. Make sure you save these with unique names. Include these files in your submission.
- Run both your backtracking and min-conflicts solvers on all of these files. Record the time taken to find a solution. Make sure these files have unique names. Include them in your submission.

Having done that, for each of the three types of problems (none, unique, and multiple), plot the average time taken to solve the problem as you vary the size of $\mathcal{X}$. Each graph should be a bar (column) chart, where the horizontal axis is the sizes of $\mathcal{X}$, the vertical axis is the average time taken to solve problems of the size given on horizontal axis, and each of backtracking and min-conflicts is a data series. Provide these three graphs in your writeup, and write down some observations about the results you obtained.

**Code/Running Requirements** Your code **must** be runnable via the bash script `https://www.csee.umbc.edu/courses/undergraduate/471/spring21/01_03/materials/a2/code/run_exact_cover.bash`.

Usage is as follows (the backslash is how your split a command across multiple lines):

```
bash run_exact_cover.bash problem_file.json solution_file.json \
    {backtracking,min-conflicts}
```

The first argument is the path to an JSON file that has the problem specification.

The second argument is the path to a JSON file that will be created (or overwritten) with your solution(s).

The third argument is a string indicating the name of the solver to use. Your program must support “backtracking” (where a basic backtracking search will be used) and “min-conflicts” (where a min-conflicts search will be used). You are welcome to make your program work with others, but you do not have to.