Homework 3

Constraint Satisfaction Problems, Optimization Problems
(Max score: 100 - Available points: 120)

15-381-Q: Artificial Intelligence (Fall 2018)

OUT: October 6, 2018
DUE: October 14, 2018 at 23:55pm

Instructions

In order to get the maximum score, you need 100 points. The total number of points available from the questions is 120. Each of the 6 sections “delivers” at most the number of points indicated at the section level. For instance, Section 2, on Continuous optimization, delivers at most 25 points. This means that, if you answer correctly to questions 2.1 and 2.5, that would amount to 20 + 15 = 35 points, you will only get 25 points. In other words, you are free to decide which questions to try to answer within each section, but cumulatively you can’t get more points than those indicated at the section level.

Homework is due on Autolab by the posted deadline. You have a total of 6 late days, but cannot use more than 2 late days per homework. No credit will be given for homework submitted more than 2 days after the due date. After your 6 late days have been used you will receive 20% off for each additional day late.

You can discuss the exercises with your classmates, but you should write up your own solutions. If you find a solution in any source other than the material provided on the course website or the textbook, you must mention the source.

You can work on the programming questions in pairs.

Submission

Please create a tar archive of your answers and submit to Homework 3 on Autolab. You should have one PDF file in your archive with all your answers (this can be compiled in latex or be handwritten), plus other files with the code for the programming questions. These code files need to be accompanied by a README.txt file with all instructions for running the code on my computer and for performing the different test. With no README.txt I won’t execute any code, and you will get 0 points.

Contents

1 Constraint Satisfaction Problems (CSP) (25 points)
   1.1 A special class of CSP (14 points) ............................................ 2
   1.2 Binarization of constraints (12 points) ..................................... 2
   1.3 Car fair: modeling floor occupation as a CSP (10 points) .......... 3
   1.4 Variable selection, assignment, and forward checking (20 points) 3
   1.5 Planning as CSP (15 points) .................................................. 3

2 Continuous optimization (25 points)
   2.1 Proving convexity (20 points) ................................................ 4
   2.2 Properties of convex functions (8 points) ................................. 4
   2.3 Convex sets and local minima (8 points) .................................. 4
   2.4 KKT conditions and global minimum (15 points) ...................... 5
   2.5 Modeling fuel production with linear programming (15 points) .... 5
1 Constraint Satisfaction Problems (CSP) (25 points)

1.1 A special class of CSP (14 points)

Consider a CSP with \( n = 15 \) variables \( X_1, X_2, \ldots, X_{15} \) each with domain \( D = \{1, 2, 3, 4, 5\} \). The following set of constraints hold among the variables: \( X_i > X_{2i}, X_i > X_{2i+1}, \forall i = 1, \ldots, n. \)

1. Draw the constraint graph (start by \( X_1 \));
2. Prune variables’ domains by enforcing binary arc consistency, report the pruned domains in the graph;
3. Using the constraint graph, find a solution using backtracking: report the steps of your backtracking procedure and make it explicit the criteria that you have used for selecting variables and values;
4. Can you identify an order for selecting the variables for which backtracking search is backtracking-free?
5. Provide a bound on the complexity of solving any CSP with \( n \) variables and \( d \) as domain size whose constraint graph has the same structure of this problem.

1.2 Binarization of constraints (12 points)

Consider the following CSP with \( n \)-ary constraints:

- Variables: \( \{X, Y, Z\} \);
- Domains: \( D(X) = \{0, 1, 2, 3\}, D(Y) = \{1, 2, 3, 4\}, D(Z) = \{2, 4\} \);
- Constraints: \( X + Y + Z = 6, Y < Z \)

1. Let’s consider first the case when there’s only one constraint, \( X + Y + Z = 6 \). The problem can be reduced from a problem with one 3-ary constraint to a problem with 3 binary constraints and four variables. This is done by introducing an auxiliary variable \( A \), which is a tuple \( < x, y, z > \), whose domain includes all the \( < X, Y, Z > \) tuples that are feasible given the constraint and variables’ domains, that is: \( D(A) = \{(0, 2, 4), (0, 4, 2), (1, 3, 2), (2, 2, 2), (3, 1, 2)\} \). Any feasible assignment to \( A \) automatically satisfies the original constraint. Draw the annotated (with all the necessary information) constraint graph (hint: \( A \) is a tuple with a one-to-one correspondence to the original variables).
2. Let’s now consider the full case, with two constraints. Transform the CSP into one with binary constraints using one auxiliary variable (and keeping the original variables). Draw the annotated constraint graph.
3. Still considering the full case, transform the CSP into its a CSP with binary constraints. However, in this case, the new CSP must feature two auxiliary variables and no variable from the original formulation. Define the two variables and draw the annotated constraint graph (this will be the dual graph of the problem since each constraint will be substituted by a vertex-variable).
4. Describe in words how the last approach can be generalized to a generic CSP instance.

1.3 Car fair: modeling floor occupation as a CSP (10 points)

Company FloorShopping rents spaces for events. It has received an order for a large car fair. The fair will feature a relatively large number, \( n \), of exhibition booths. Each booth will occupy a rectangular space, that varies according to the company renting the booth. The fair will happen at the ground floor of a building shaped as a rectangular. FloorShopping has to identify a feasible partitioning of the floor space in order to feasibly and safely accommodate all the exhibition booths. The available space is represented in a Cartesian plane \((x, y)\), where the coordinates of the rectangular area span from \((0, 0)\) to \((W, H)\).

Provide a precise CSP model (variables, domains, constraints) for the problem of FloorShopping (note: since the rectangular areas for the booths are not specified, you have to use some appropriate parametric representation of these rectangles).

1.4 Variable selection, assignment, and forward checking (20 points)

The following is an instance of a cryptarithmic puzzle problem:

\[
\begin{array}{c}
O \\
D \\
D \\
+ O \\
D \\
D \\
\hline
E \\
V \\
E \\
N \\
\end{array}
\]

Each letter stands for a distinct digit, and the goal is to find an assignment of digits to letters so that the sum is correct. It is also required that there be no leading zeroes.

1. Describe the CSP: variables, domains, and constraints (hint: introduces auxiliary variables for the carry operations).

2. Solve the cryptarithmic problem using backtracking search. While you progress with the search, use forward checking. Make use of the minimum remaining values and least constraining value heuristics respectively for variable selection and variable value.

Note: I won’t accept any “heuristic / ad hoc” solution of the puzzle. I’m not interested in solving “this” simple puzzle, instead, you must show that you can implement the general algorithm (e.g., what if the puzzle was involving all 25 letters of the alphabet over 10 three sentences? You will need a computer to solve it!)

In the first round, assign a value to a variable in \( E, V, O, D, N \) (i.e., do not consider the auxiliary variables). Among the variables in \( E, V, O, D, N \), break ties for variable choice alphabetically, and then for assignment choice by choosing the smallest remaining value.

For each step (variable selection and assignment), precisely show your reasoning (i.e., show how you apply forward checking, how you apply the heuristics, when and how you do backtracking). Please keep in mind that you should get to the solution in about 10 steps or less (and without incurring into backtracking). If you find yourself being engaged in a lengthy process, you are on the wrong track.

3. When solving the CSP, you will be facing an issue that in principle you don’t know how to precisely address based on what has been presented in the classes (i.e., you don’t the “algorithm”, such that you will have to use some ad hoc reasoning). Point out the issue and discuss how you did deal with it.

1.5 Planning as CSP (15 points)

Consider the flat tire planning problem presented in the class (Lecture 6, slide 35, and Russel and Norvig, 10.1.2 Example: The spare tire problem). The problem has been introduced using the PDDL formalism. Convert the PDDL problem into a CSP one. You need to show what are the variables, their domains, and the constraints among them. Since we know that the optimal plan has length 3, you can use this information constructing the CSP. Be precise in the way you represent the CSP!
Hints: Each condition defines a CSP variable. Also each action defines a CSP variable. The PDDL plan consists of a sequence of $n$ actions, that has to be translated into $n$ sets of CSP variables. Initial and goal conditions can be easily expressed through constraints. Actions (now CSP variables) have precondition constraints (now variables) and results into effects (now variables). All these relations result into CSP constraints.

2 Continuous optimization (25 points)

2.1 Proving convexity (20 points)

Which of the following mathematical programming problems are convex? Prove your statements.

1. $\min 3x_1 - 5x_2$
   $s.t. \ x_1^2 + x_2^2 \leq 1$
   $x_1, x_2 \in \mathbb{R}$

2. $\min 3x_1 - 5x_2$
   $s.t. \ x_1^2 - x_2^2 \leq 1$
   $x_1, x_2 \in \mathbb{R}$

3. $A$ is an $m \times n$ matrix and $b \in \mathbb{R}^m$.
   $\min \ \exp \left( \sqrt{\sum_{i=1}^{n} x_i^2} \right)$
   $s.t. \ Ax \leq b$
   $(x_1, \ldots, x_n) \in \mathbb{R}^n$

2.2 Properties of convex functions (8 points)

Which ones of the following statements are necessarily false? Which ones could be false? Explain why.

1. The convex function $f(x)$ has three minima of the same value.

2. The convex function $f(x)$ has infinite minima of the same value.

3. The convex function $f(x)$ has three local minima of different values.

4. Function $f(x)$ is discontinuous and convex.

5. The convex function $f(x)$, defined in the closed set $[0, 1]$ has maximum in $(0.5)$.

6. The convex function $f(x)$, defined in the open set $(0, 1)$ has maximum in its domain.

2.3 Convex sets and local minima (8 points)

Prove or disprove the following statement: the set of all local minima of a convex function is a convex set.
2.4 KKT conditions and global minimum (15 points)

Consider the following constrained minimization problem:

$$
\min_{x} f(x) = x_1^2 + x_2^2 + x_3^2
$$

s.t.  

$$
g_1(x) = 2x_1 + x_2 - 5 \leq 0
$$

$$
g_2(x) = x_1 + x_3 - 2 \leq 0
$$

$$
g_3(x) = 1 - x_1 \leq 0
$$

$$
g_4(x) = 2 - x_2 \leq 0
$$

$$
g_5(x) = -x_3 \leq 0
$$

1. Define the Lagrangian function and apply the KKT conditions in order to find the critical points. You should be able to define a system of equations that can be solved manually by simple inspection and one single solution point should be found. Be careful defining the domain of the multipliers (based on the minimization nature of the problem).

2. Why can we say that the found critical point is the global minimum for the original problem?

3. In the solution, a few multipliers will have value zero. What does this mean for the solution? (i.e., in terms of constraint activation, it terms of “price” to pay for constraint violation, ...).

2.5 Modeling fuel production with linear programming (15 points)

The QFuel company produced two types of fuel, each obtained by blending three different oils, A, B, C, in some proportions. More specifically:

- Fuel blend 1 requires: at least 20% of A, at most 40% of B, at least 45% of C;
- Fuel blend 2 requires: at most 40% of A, at least 35% of B, at most 35% of C.

Each liter of blend 1 can be sold for 1.10$ and each liter of blend 2 can be sold for 1.20$. Long-term contracts require at least 20,000 liters of each blend to be produced.

Cost and availability of the different oils are the following:

- Oil A - Cost: 0.3 ($/l), Availability: 6,000 (l)
- Oil B - Cost: 0.4 ($/l), Availability: 10,000 (l)
- Oil C - Cost: 0.5 ($/l), Availability: 12,000 (l)

Formulate this blending problem as a linear programming problem. The goal is to find the quantity of each oil that must be employed in each fuel blend respecting the constraints (availability and proportions) and maximizing the overall profit (assuming that 20,000 liters of each blend will be sold at the indicated prices).

2.6 Function optimization using analytic methods (12 points)

The function

$$
f(x_1, x_2, x_3) = x_1 + 2x_3 + x_2x_3 - x_1^2 - x_2^2 - x_3^2
$$

has one critical point, in $x_0 = \left( \frac{1}{7}, \frac{2}{7}, \frac{4}{7} \right)$, which is found by solving $\nabla f(x) = 0$.

1. Determine whether the point is a minimum or a maximum using algebraic methods;

2. Determine whether the function is well conditioned or not by:

   (a) considering the projections of the function on the planes $x_1 - x_2$, $x_1 - x_3$, $x_2 - x_3$, and studying the shape of the resulting isocontours;

   (b) using the eigenvalues of the Hessian matrix.
2.7 Function optimization using numerical methods (16 points)

The following optimization problem is given:

\[
\max_{x_1, x_2} f(x_1, x_2) = 4x_1 + 6x_2 - 2x_1^2 - 2x_1x_2 - 2x_2^2.
\]

1. Find the optimum of the function using gradient ascent. The optimum occurs at \( \left( \frac{1}{3}, \frac{4}{3} \right) \) and is reachable with good approximation in 6 iterations of the gradient algorithm starting from \((1, 1)\). Manually perform and report the details of the 6 iterations, and measure the final approximation error (for finding the step size at each iteration you can use calculus or even a small program (if necessary) to solve the one-dimensional optimization problem).

2. Show in a graph the progress of the algorithm reporting the performed steps and the isocontours (don’t need to do it for all steps, at least the first three ones).

3 Integer optimization (25 points)

3.1 Modeling staff scheduling (15 points)

As it is common practice in many companies (as well as hospitals, police departments, etc.), the 24h open YummyPizza fast-food divides the daily work schedule into time windows. Each time window \( t \) corresponds to 3 hours of work and requires a specified number of workers \( d_t \). A work shift \( s \) covers \( T = 3 \) consecutive time windows and has associated a specific cost \( w_s \) for paying the single employees. YummyPizza makes use of \( n = 4 \) work shifts and organizes its time windows as shown in the table below. The table also reports the number of workers required per time window and the cost of each work shift.

<table>
<thead>
<tr>
<th>Time window</th>
<th>Shift 1</th>
<th>Shift 2</th>
<th>Shift 3</th>
<th>Shift 4</th>
<th>Workers required</th>
</tr>
</thead>
<tbody>
<tr>
<td>6am - 9am</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>6am - 12 noon</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>46</td>
</tr>
<tr>
<td>12 noon - 3pm</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>3pm - 6pm</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>6pm - 9pm</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>9pm - 12am</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>12am - 3am</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>20</td>
</tr>
<tr>
<td>3am - 9am</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>30</td>
</tr>
<tr>
<td>Wage rate per 9h shift</td>
<td>135$</td>
<td>140$</td>
<td>190$</td>
<td>188$</td>
<td></td>
</tr>
</tbody>
</table>

The goal of YummyPizza is to define the number of workers per work shift such that all time windows are covered (at least) as required and the costs for paying the employees are minimized.

Formulate YummyPizza’s problem as a binary programming problem. Be precise (and careful) in the definition of the variables and in the mathematical formulation of the problem.

3.2 Branch-and-bound: general tree analysis (20 points)

Consider the branch-and-bound tree in the figure. For each sub-problem it has been possible to compute both a lower bound and an upper bound that are reported between square brackets (i.e., both a feasible solution
and a relaxed solution are computed out of the sub-problem).

1. Is the problem of minimization of maximization? Justify your answer.

2. Given the current status of the tree, is it possible to fathom some nodes? Which ones? Why?

3. Given the tree, what is the feasible range of values for the optimum?

4. After possible node fathoming operations, which sub-problem will be expanded next by a best bound first strategy, and which one by a depth-first strategy?

5. Let’s assume that after solving the sub-problem indicated by best bound first strategy, two new sub-problems are generated, $SP_7$ and $SP_8$. Let the solution of the relaxation of $SP_7$ return unfeasible. Define a pair of values $[lb, ub]$ that could be feasibly returned by solving $SP_8$ and that would let finish the branch and bound search process (returning a feasible solution).

6. Let’s assume that the problem is one of minimization and that at each sub-problem only the lower bounds are computed using a relaxation. Moreover, let $SP_4$ be the only sub-problem that returns a feasible solution, of value 14.7. Show how the value of the UB changes step-by-step while solving the different sub-problems.

### 3.3 Branch-and-bound for a 0-1 problem (22 points)

In the figure, a branch-and-bound tree is shown for an IP problem. Explain what happens during the search, step-by-step. Is it a max or a min problem? (justify the answer). The index $t = i$ indicates the order followed for solving the sub-problems, while the number on the top left corner of sub-problems’ box indicates the order of generation of the sub-problem. What are the criteria/heuristics that have been employed to decide about the next branching variable and the next sub-problem to solve? The solution of the relaxation is given by $\bar{z}$. A number like $43\frac{1}{4}$ means $43 + \frac{1}{4}$ (i.e., the fractional part of the value is singled out). Note that in the first
sub-problem $x_4 = 0$ is missing by mistake.

4 Programming: CSP (20 points)

4.1 Backtracking and constraint propagation for graph coloring (20 points)

Implement, in any programming language of your choice, the code for performing the backtracking + forward checking algorithm and solve the two test problems on vertex coloring contained in the folder `testbed/VertexColoring`. The data files have the following format: a first line that specifies the parameters of the problem, it follows the list of edges in the constraint graph. For instance, the file `27-nodes-3-colors.dat` looks like the following:

```
p 27 nodes 68 edges 3 colors
1 2
1 22
2 1
2 22
3 4
```

The first line, starting with a `p`, defines the problem: 27 variables, 68 binary constraints, 3 colors available for the coloring (i.e., the “chromatic number” of the problem is 3). Then, the list of edges (i.e., binary constraints in the graph) follows, where 1 2 means that there is a constraint between vertices (variables) 1 and 2. The graph is undirected, such that there will be also a constraint between 2 and 1.

You need to find a feasible assignment, and check that it is indeed feasible! In the PDF report, for each problem, you need to report the solution, time it took to find the assignment, and the number of backtrackings.

Note that the problem with 27 nodes is the same as the one shown in the videos during the class (i.e., you have a way to assess how bad / good you are doing).
4.2 Local search for graph coloring (18 points)

Consider the same graph coloring problems of the previous question and find a feasible assignment, or an assignment with minimal number of conflicts using a Local search approach. You’ll need to design your local search for the problem.

Describe the local search algorithm (neighborhood definition, criterion for exploring the neighborhood, criterion for accepting or rejecting a solution, criterion for stopping). Report the solution found (number of violations), the CPU time to reach it, and the number of iterations. If you have also answered the previous question, you should compare results and performance.

5 Programming: Function optimization and gradient methods (15 points)

5.1 Gradient ascent and step size (15 points)

The following optimization problem is given:

$$\max_{x_1, x_2} f(x_1, x_2) = 4x_1 + 6x_2 - 2x_1^2 - 2x_1x_2 - 2x_2^2.$$ 

This is the same problem of Section 2.7.

1. Implement the gradient ascent with adaptive step size. Among the different approaches discussed in the class, you are free to select the adaptive step size method that you judge as the most appropriate one for the problem. Describe and justify your choice and in a table report: the optimum, the time to reach the optimum with some small approximation error, the number of steps.

2. Implement the gradient ascent with fixed step size and report in a table the behavior of the algorithm for three different step sizes at your choice. Justify your choices and discuss the results. In the table report: the optimum, the time to reach the optimum with some small approximation error, the number of steps (give the algorithm a maximum of 10,000 steps in case of lack of convergence).

5.2 Weber point with deterministic and stochastic gradient methods (15 points)

Consider the case of a Weber point problem (Lecture 9, slide 11): inside a specified rectangular area n points are given (e.g., they might represent location of facilities, warehouses, individuals/spots to monitor, . . .) and the goal is to find the point in the area whose sum of the Euclidean distances from all points is minimized. Let the area be defined as the Cartesian product $[0, 100] \times [0, 100]$ and let $n = 10,000$.

You are asked to generate the n points to be randomly distributed within the area and determine the coordinates of the Weber point with some approximation. At this aim, you should explore the following three possibilities:

1. Use a gradient descent approach with a fixed step size. Try out a few step sizes, justify your choices.

2. Use a gradient descent approach with an adaptive step size at your choice, justify your choice.

3. Use a stochastic gradient descent approach with a fixed step size.

For each approach: report the found point, the value of the function at the point, the number of iterations. If an algorithm doesn’t seem to converge, make use of a stopping criterion based on a maximum time or number of iterations. Compare and discuss the results.

6 Programming: Combinatorial optimization (10 points)

6.1 Solve and study set covering problems (10 points)

In set covering problems, given a set of “activities” (e.g., flight crews), and a set of “requirements” (e.g., flight routes), the goal is to select a subset of the available activities that covers all requirements at the minimum
cost. Each activity is a resource that can cover one or more requirements with a certain cost (e.g., a crew of Spanish-speaking people based in Spain can effectively be employed for covering multiple routes from Spain to South America destinations, with a cost which is higher than, for instance, that of a flight crew that covers flights from Doha to Kuwait). Overlappings in coverage are admitted (e.g., in the final solution two crews that can cover the same route can be selected, but only one will be actually used).

A general formulation of a set covering problem is as follows:

$$\min \sum_{j=1}^{A} c_j x_j$$

subject to

$$\sum_{j=1}^{A} a_{ij} x_j \geq 1, \quad i = 1, \ldots, R$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, A$$

where \(A\) is the number of activities, \(R\) is the number of requirements, \(x_j\) is 1 if activity \(j\) is selected in the final solution, 0 otherwise, and coefficients \(a_{ij}\) are 1 if activity \(j\) covers requirement \(i\), 0 otherwise.

1. Implement a general (parametric) model for set covering problems using the PuLP python library.

You will need first to install the library. At this aim, you can conveniently use `pip install pulp` if you use pip. More options are documented here: https://pythonhosted.org/PuLP/main/installing_pulp_at_home.html.

Documentation and examples of use are available on the library’s web site: https://pythonhosted.org/PuLP. The use of the library is indeed quite straightforward. In the files of the homework you will find an example, knapsack.py, that shows how to model and solve a knapsack problem (the parameters of the instance are random values that I use as input). You are invited to first go through the example, that should be self-explanatory and contains all the elements that are necessary for you to write the set covering model using PuLP.

2. You are asked to generate random instances of set covering problems and solve them. You have to do this inside the PuLP code. The properties of the instance are determined by the value of four parameters:

   - Number of activities, \(A\);
   - Number of requirements, \(R\);
   - Probability \(p_c\) that an activity covers a requirement: you can generate the matrix of parameters \(a_{ij}\) by assigning 0 or 1 to a specific \(a_{ij}\) based on the probability \(p_c\). For instance, if you set \(p_c = 0.75\), then, approximately 75% of the \(a_{ij}\) will have a value of 1 and 25% a value of 0, of course these proportions would be accurate only if a large number of instances would be generated.
   - Cost of an activity, \(c_j\). Also this value can be set at random, by sampling costs uniformly in the discrete interval between 1 and \(C\).

You can start by using the following set of values: \(A = 7, R = p_c = 0.25, C = 50\). This would allow to generate very small instances that should be solved in almost no time. Check the solution values in order to confirm the correctness of your model. If you keep generating and solving random instance based on the above four values for the parameters, you will notice that most of the instances are actually reported as “infeasible”. Why is this the case?

3. If the four parameters are set to different values, depending on the values the resulting random instances are expected to show a variety of behaviors in terms of feasibility vs. infeasibility, time to solve the instance (e.g., hardness of the instance), cost of the solution (which is also related to how constrained is the problem).

You are asked to reason on the problem structure and to play with the four parameters in order to perform an empirical sensitivity analysis of the problem: get some understanding about how parameter values impact on the characteristics of the solution.

More precisely, for each one of the scenario below, try to find a set of parameter values such that the “majority” of the resulting randomly generated instances would have the property that defines the scenario. The scenarios are defined as follows:
(a) Most of the generated instances are infeasible;
(b) Most of the generated instances are feasible;
(c) Most of the generated instances are solved in less than 10 seconds;
(d) Most of the generated instances are solved in way more than 10 seconds;
(e) Most of the generated instances have a total cost less than 100;
(f) Most of the generated instances have a total cost quite larger than 100.

In your empirical analysis, the values assigned to $A$ and $R$ must be always larger than 50. The “majority” above means that you mostly observe the property over at least 10 randomly generated instances.

For each scenario, report the parameter values and discuss “why” those values generate the observed behavior.