Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Problem Set 3
Tutorial

In the spirit of Numerical Recipes in C, write a tutorial on this reasoning technique, such that an interested robotics student could apply it on a scenario of their choosing. Be precise about the inputs and outputs and how the algorithm works, but convey intuitions over mathematics.

Background of our robot problem:

Captain Fridge is responsible for managing food options for the owner that are stored in the fridge. The robot takes orders from the master for food items, gives the master alternate options if the asked item is currently not available or running low on supply and fetches final items from the fridge to the master. Over time it learns master's preference for food items and preferred combinations of food items, drinks and dessert and so on. It also learns and remembers preference of food items based on time of the day - so which food and food combinations are preferred for breakfast, lunch and dinner. While giving options to master, Captain remembers the preference of items in combination with each other.

For any particular meal, the captain knows the most preferred combination of items, however, due to lack of supplies, a highly preferred item may not be available. In this case, captain presents to its master, a set of choices that are guaranteed to have the minimum level of compromise and inversely, the maximum level of satisfaction for the master.

Since this problem entails finding a set of acceptable options that may not be the most preferred (in case all constraints are not met) but to maximize the aggregate satisfaction level for the chosen combination, we picked fuzzy constraint satisfaction technique as a reasoning technique for our project.

Fuzzy Reasoning Technique for Constraint Satisfaction Problem:

Constraint satisfaction problems (CSPs) are defined as a set of objects whose state must satisfy a number of constraints or limitations[1].

Lots of problems in real-life cannot be expected to have a perfect solution because the problems are over-constrained and do not lead to a feasible solution. In such a case, we relax the restrictions so that we can at least find a solution as close as possible to the actual solution. These relaxable constraints are termed as soft constraints and problem that employ such relaxation are called soft CSP.[2]

In general, in a soft CSP not all the given constraints need to be satisfied — either because all of them cannot be met, theoretically, or in practice it would require too much time to find a solution, or simply, the real problem to be modeled is such that soft constraints are adequate and should be introduced, e.g. to express possible alternative requirements. In the case of a fuzzy constraint, different tuples satisfy the given constraint to a different degree.

---

[1] http://en.wikipedia.org/wiki/Constraint_satisfaction_problem
[2] http://www.constraintsolving.com/tutorials/soft-constraints-tutorial

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Problems in AI such as planning and scheduling are modeled as constraint satisfaction problem. Fuzzy constraint satisfaction based approach finds the best combination of values that satisfy the requirements not in a binary truth values of yes or no but in terms of degree of satisfaction for a given category choices where the preferences are known only among a few of the choices at a time. This also entails creating a method to combine the preferences in some manner to get an aggregate degree of satisfaction.

In fact several real life problems are inconsistent and do not have a solution. These problems can be modeled using fuzzy logic. In real life when we our best choices are inconsistent, we compromise with choices that are no longer feasible or available to us with the next best choice - it may not satisfy us 100% but among the set of alternatives, we can decide to what level of satisfaction the other alternatives give us at that time.

Fuzzy constraint satisfaction problems have been found useful in modeling various realistic problems that have flexible, priority or conditional types of constraints. FCSP are an optimization type of problem where the goal is to maximize the aggregate level of satisfaction of the goal. Since optimization problems are computationally expensive to solve, several heuristics have been applied to FCSP to handle the complexity.

Background:

What do you mean by Fuzzy?
   ○ Fuzzy logic is a superset of conventional(Boolean) logic that has been extended to handle the concept of partial truth- truth values between "completely true" and "completely false". The importance of fuzzy logic derives from the fact that most modes of human reasoning and especially common sense reasoning are approximate in nature.[3]

       i.   In fuzzy logic, exact reasoning is viewed as a limiting case of approximate reasoning.
       ii.   In fuzzy logic everything is a matter of degree.
       iii.   Any logical system can be fuzzified
       iv.   In fuzzy logic, knowledge is interpreted as a collection of elastic or, equivalently , fuzzy constraint on a collection of variables
       v.   Inference is viewed as a process of propagation of elastic constraints.

Representation of a Fuzzy Constraint Satisfaction Problem:

    A FCSP is a tuple P = (X, D, C) defined by:

   ○ A set of n variables X = {X1, ..., Xn}. These are the decision variables in our problem whose assignment we are solving.

---

[3] http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/sbaa/report.fuzzysets.html

○ A set of n definition domains D = {D1, ..., Dn}, Di being the definition domain of variable Xi. Each variable can have any number of possible values allowed in its corresponding domain.

○ A set of m fuzzy constraints C = {C1, ..., Cm} where each constraint C is defined as a pair (V, R):

  i.   V (X) is a set for variable X for which the constraint C fuzzy restricts the values these variables can simultaneously take.

  ii.   R is a fuzzy relation that represents the combinations of values that the constraint C authorizes for these variables which corresponds to the constraint satisfaction ranging from [0,1] which each assignment of these values to the variables.

Level of satisfaction of a Fuzzy Constraint

The solution of a Fuzzy constraint satisfaction problem is also fuzzy, for any instantiation, the degree of satisfaction of the final goal indicates how well the constraints are satisfied jointly. There are several way to define the degree of joint satisfaction of individual constraints using fuzzy set theory methods.

An individual constraint can be considered a membership function of a fuzzy set, where the way we define joint satisfaction of more than one constraint is the problem of defining membership function for intersection of fuzzy sets.

There are several ways to define **joint satisfaction degree** and we cover three of them here:

1. Based on conjunctive combination principle: where the joint satisfaction is the minimum satisfaction of the individual constraints.

2. Based on productive combination principle: where the joint satisfaction is the minimum satisfactions of the individual constraints.

3. Based on average combination principle where the joint satisfaction is the average satisfaction from all individual constraint.

3. Describe the solution approach at a high level. Illustrate how the reasoning process works on a simple example.

Lets study the FCSP technique for a simple example for illustration of the technique.

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Suppose we are hosting a party and have a lunch menu to plan for a three-course meal with assorted drink. We have 3 types of appetizer, 3 types of main dish 2 types of dessert and 3 types of drink options available as shown below.

The fuzzy constraints relate 3 decision variables at a time and indicate the preferences for the values in each variable against each of the other variable.

Lunch menu (3 decision variables):

1. Appetizer in the fridge (A)
   - Frozen spring rolls (a1)
   - Frozen samosa (b1)

   A belongs to D1= [a1, b1]

2. Main dish (M)
   - Fried rice (a2)
   - Chicken curry (b2)
   - Pasta (c2)

   M belongs to D2= [a2, b2, c2]

3. Drink (D)
   - Wine (a3)
   - Coke (b3)

   T belongs to D3= [a3, b3]

 Constrains (binary):

C1:

Satisfaction	A	M
1	a1 (spring rolls)	a2 (fried rice)
0.4	a1 (spring rolls)	b2 (curry)
0.2	a1 (spring rolls)	c2 (pasta)
0.8	b1 (samosa)	c2 (pasta)
0.5	b1 (samosa)	b2 (curry)

C2:

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Satisfaction	A	D
1	a1 (spring rolls)	b3 (coke)
0.7	a1 (spring rolls)	a3 (wine)
1	b1 (samosa)	a3 (wine)
0.1	b1 (samosa)	b3 (coke)

C3:

Satisfaction	M	D
1	a2 (fried rice)	a3 (wine)
0.7	a2 (fried rice)	b3 (coke)
1	b2 (curry)	a3 (wine)
0.4	b2 (curry)	b3 (coke)
1	c2 (pasta)	b3 (coke)
0.6	c2 (pasta)	a3 (wine)

Fuzzy Solution:

It can be seen that this problem has no perfect solution - i.e. no solution with joint satisfaction level of 1. So, we need to find the next best solution which is as close to the perfect solution as possible.

Now, depending upon how joint satisfaction is defined, the assignment for each decision variable is made to maximize the joint satisfaction of the full assignment. If lets say, the joint satisfaction is a minimizing function then:

Joint satisfaction of [a1,a2,b3] = min ((a1,a2) , (a2,b3))
= min (1.0,0.7)
= 0.7 is the best solution with maximum joint satisfaction value.

Highlight the important features, particularly if they are interesting or novel.

As seen in the simple demonstrative example above, the final solution depends largely on the type of joint satisfaction approach that we chose for the problem. In the example above, we took a minimum approach for joint satisfaction while combining constraints because we wanted a 'safe' or 'pessimistic' approach (we are planning for a party and want to make sure the most constrained preference is satisfied in the menu).

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

However, the choice of type of joint satisfaction function depends on the nature of the problem. The minimum based function is a 'rough' approach since it does not differentiate between two equally bad choices that may have better degree of satisfaction of other constraints.

The average joint satisfaction function does not differentiate between instantiations that fully violate at least one of the constraints while the product joint satisfaction function takes that into account by the virtue of zero multiplication. The definition of joint constraint satisfaction function can be modified to take into account the importance of individual constraints.

In the following sections we delve deeper into the features of our implementation that define 'good enough' solution by defining a threshold value of joint satisfaction of instantiations above which solutions are acceptable. This threshold can be defined in absolute or relative measure of the maximum possible joint satisfaction value known for the solution.

We also discuss about the algorithm that we have implemented to search through solution space (instantiation tree).

Promised capability

In general, all constraint satisfaction problems can be modeled as fuzzy constraint satisfaction problem by carefully assigning satisfaction levels to constraint related variables that denote the level of compromise required if that constraint was not met.

For example, in the simple temporal constraint satisfaction problem from one of our thought exercises, we were given the temporal constraint for starting for
- Leaving MIT (A),
- Reaching North End (B),
- Finishing lunch at North End and leaving for movie (C)
- Arriving at Boston Commons (D)
- Watch movie starting at a fixed time (E)

This problem can be modeled in form of a Fuzzy CSP and solved using our APIs. There are two decision points in the problem for choice of mode of transport for A and C. These two decision points are the decision variables for the problem. Each variable has a domain that represents the choices available for every variable assignment. In this case, the two variables have the domain {walk, mbta, hubway, uber}.

The constraints are represented in form of set of relationships between assignment values for the variables. The relationship represents the level of satisfaction of the constraint for the assigned values of each variable. In this case, the level of satisfaction is the inverse of the cost function associated with the two choices:

XA	XB	$lAB+lCD$	$\delta1+\delta2$	$gA(XA)+gB(XB)$	Cost Saved	Satisfaction Value

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

					(11 - x)	(normalized cost)
walk	walk	65	35	0	11	1
walk	Hubway	55	25	3	8	0.727272727
walk	MBTA	"	"	2	9	0.818181818
walk	Uber	"	"	5	6	0.545454545
Hubway	walk	45	15	3	8	0.727272727
Hubway	Hubway	35	5	6	5	0.454545455
Hubway	MBTA	"	"	5	6	0.545454545
Hubway	Uber	"	"	8	3	0.272727273
MBTA	walk	50	20	2	9	0.818181818
MBTA	Hubway	40	10	5	6	0.545454545
MBTA	MBTA	"	"	4	7	0.636363636
MBTA	Uber	"	"	7	4	0.363636364
Uber	walk	35	5	6	5	0.454545455
Uber	Hubway	25	−5	9	2	0.181818182
Uber	MBTA	"	"	8	3	0.272727273
Uber	Uber	"	"	11	0	0

Once we map the values for each variable in relation to the other variables, we can solve this problem as an optimization problem to find an assignment for each variable that has the maximum satisfaction value for the final solution.

Generalizing Fuzzy CSPs:

In fuzzy CSPs, a satisfaction level of 1 means fully compatible choices of variables. This is equivalent to the absence of a constraint. Inversely, the satisfaction level of 0 denotes complete incompatibility. To convert a Constraint Satisfaction Problem into a Fuzzy CSP, instead of zero, a value between 0 and 1 is assigned to the relationship to depict the level of compromise required while choosing the corresponding values for the decision variables.

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

In general, a fuzzy CSP can have several number of decision variable and several number of acceptable values is their corresponding domains.

Let's say if the n is the number of decision variables and m is the number of values in their domains (for simplicity of illustration of these concepts we are taking a uniform sized domain for all variables, in our implementation, we allow variables to take any domain size)

Problem Size depends on:
       a.   n - number of variables
       b.   m - size of their domains

Problem size can be measured as the product of the domain sizes for each variable, i.e. the number of combinations of values that could be generated as potential solutions.
In general, the problem size = $m \wedge n$

Problem Complexity depends on:
      If all the constraints present in the problem is represented as a graph where each variable is a node and the existence of an edge depicts the existence of a constraint between two nodes, then the connectivity of the FCSP is defined as the fraction of various binary combination of variables that have a non unitary satisfaction value.

As an example, if the constraints are binary, ie they express relationship between two variables at a time, and if the number of decision variables are say, N then there are: NC2 possible constraints.

There is no limitation on the number of variables that can be a part of constraints. A constraint may involve as many variables as the problem requires. Our implementation allows the user to choose the arity of constraint as required by the problem.

The detailed discussion for solution approaches and features in technical detail for solving FCSPs are described below. Most of the discussion is based on papers [1] and [2]:

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Backtracking:

One of the basic solution approaches for solving general CSPs is backtracking. Backtracking is a tree search (usually depth-first search) over the domains of all the variables of a CSP. A tree in CSPs is a tree of domains, with each level denoting a variable, with nodes being the values in
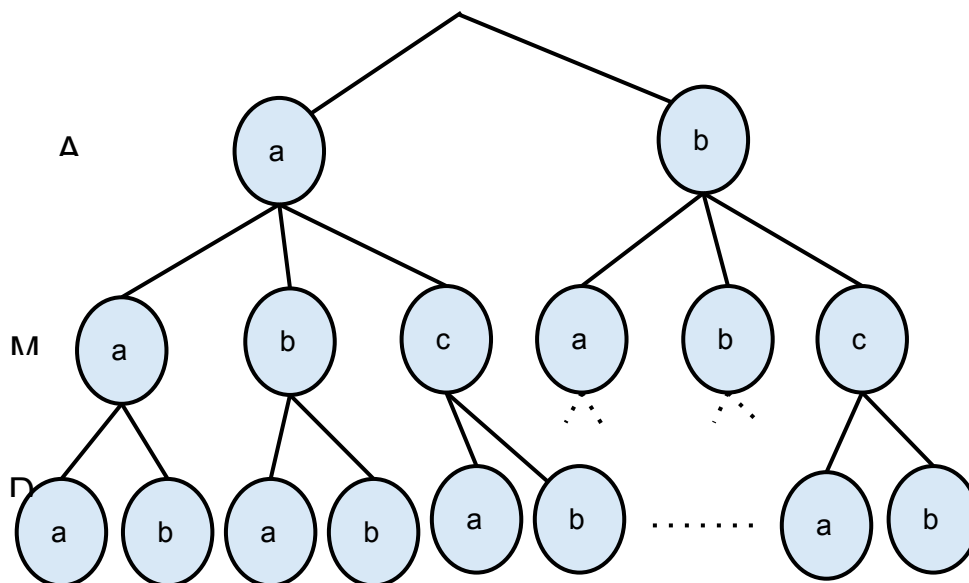

Figure 0 - Search tree for the lunch problem, used by backtracking and branch-and-bound algorithms

its domain. For example, fig 0, shows the tree for our example lunch problem. The variables can be ordered arbitrarily in the tree, and that determines the order in which variables are assigned during the search.

During each assignment of a variable, the consistency of the assignment is checked with all the previously assigned variables. If the assignment is not consistent with previous assignments, the algorithm backtracks and looks for other values of the variable. In the end, backtracking either returns a consistent solution instance, if found, or continues to find other solutions.[2]

Backtracking can be extended to fuzzy problems as well, with custom definitions of "consistency" of two variables. In fuzzy CSP, whether a solution instance is consistent depends on the degree of joint satisfaction sought for the solution, so-called alpha. It may be computationally difficult to check if any two variables are consistent with one another given the alpha for the entire solution, therefore the consistency check can be done after finding an entire solution instance.

Advantages and disadvantages of backtracking:

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Backtracking, in worst case, searches over all tree space. There is no intelligence on which variables to check first or which values to assign first during the algorithm. Also, the pruning doesn't occur in the middle of the search, and the algorithm reaches at the leaf stage until it knows to backtrack.
On the other hand, it is the guaranteed way of finding either alpha solution or the best solution possible. It is simple to understand.

Implementation:

We have implemented a backtracking algorithm for finding alpha solutions, which are the instances of solutions satisfying the problem constraints with joint degree greater than or equal to alpha. We have implemented a version that checks alpha-consistency of a solution instance not at every variable assignment, but once a candidate solution instance is found. It is done so because in fuzzy problem setting, there is not as straight-forward way to check consistency of a partial soution as in the crisp CSPs. Once the solution instance is found, if its joint satisfaction degree is less than alpha, the algorithm backtracks in the tree to try to find other consistent solutions.

Pseudocode:

```
get_alpha_solution_backtracking(alpha):
    #stack keeps track of current vertex to explore and path until that vertex
    stack = [values in first level]
    vertex, path = stack.pop()
    for next in tree[vertex]:
        if tree[next] == []: #current vertex is leaf
            if_path_consistent: #joint_sat >= alpha:
                return solution # if want all solutions, do "yield" instead of return
        else:
            stack.add(next, path+[next])
```

Running backtracking on the example problem:

In our example, following are the steps for backtracking:
1. Assign A= a1
2. Assign M = a2
3. Assign D = a3
4. Check consistency of (a1, a2, a3). In our example, if we take "productive" as our joint satisfaction metric for consistency, joint_sat = sat(a1, a2) x sat(a1, a3) x sat (a2, a3) = 1x0.7x1 = 0.7. Because joint_sat > 0, this assignment is consistent.
5. Return (a1, a2, a3)
6. If choose to continue search, now backtrack and assign D = b3. Joint satisfaction of (a1, a2, b3) = 1 x 1 x 0.7 = 0.7
7. Now backtrack and assign M = b2. Continue similarly

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Branch and bound:
The next natural approach to avoid searching over a lot of space is to try to prune the search tree early on in the tree if some paths are known to yield provably worse solutions than something that we know we can definitely get. One of the well known ways to do that is branch and bound algorithm. The idea is to have some kind of upper bound and/or a lower bound on the satisfaction of the solution, and prune parts of tree that provably result in worse solutions, i.e. whose upper bound of the satisfaction is less than the current lower bound.

Advantages and disadvantages:
In general, it doesn't search over all tree space. However, it still has to finish searching all tree to find the best solution. More importantly, it doesn't control the order of variables to assign, therefore may need to go over all possibilities. Another advantage/disadvantage comes along with the type of upper bound chosen. If it is difficult to calculate upper bound at each node, it may still take a long time.

Implementation:
In our implementation, we first find any feasible solution with one run of backtracking search, and have as the lower bound, its joint satisfaction degree. Then, we run branch and bound search. The lower bound keeps on increasing as we find better and better solutions. At any point in the search, if we know that assigning a value to a variable cannot yield a solution better than the lower bound, we do not visit that variable and onwards. We have implemented a branch and bound solution with an upper bound for a variable assignment called "appropriateness" described in [1] (also described later on in "heuristics search" topic), as well as the upper bound called "joint partial satisfaction" that we came up with (also described later on in "heuristics search"). However, choice of upper bound/lower bound is not fixed in branch and bound.

Pseudocode:
```
get_best_solution_branch_n_bound():
   #stack keeps track of current vertex to explore and path until that vertex
   stack = [values in first level]
   vertex, path = stack.pop()
   feasible_solution = get_a_feasible_solution_backtracking()
   lower_bound = joint_satisfaction(feasible_solution)

   for next in tree[vertex]:
      partial_assignment = path + [next]
      upper_bound = get_upper_bound(partial_assignment)
      if upper_bound < lower_bound:
         continue
      if tree[next] == []: # vertex is leaf
         joint_sat = get_joint_satisfaction(path)
         if joint_sat > lower_bound:
            lower_bound = joint_sat #update current lower bound
```

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

```
    else:
        stack.add(next, path+[next])
```

Run on example problem:
1. Get a feasible solution from backtracking. Let's say it returns (a1, b2, b3) with joint_sat = 1 x 0.4 x 0.4 = 0.16. So, lower_bound = 0.16
2. Assign A = a1.
3. Try M = a2. Get upper bound on resulting solutions with partial assignment (A=a1, M=a2). With "appropriateness" measure, upper_bound = 1.0
4. Assign M = a2 because upper_bound > lower_bound. If that was not the case, algorithm would backtrack and assign M = b2
5. Try D = a3. Get upper bound on solution (a1, a2, a3) which is a tight bound, exactly equal to joint satisfaction, i.e. 0.7
6. Update lower bound to 0.7, and continue search, now checking for (a1, a2, b3)

Branch and bound can be implemented to find the best solution, as in the pseudocode and the example run above, or to find an alpha solution. In case of alpha solution, the lower bound is alpha to start with, and as soon as we find the first alpha solution, we return the solution. So, the lower bound is fixed in this case.

Heuristic search:
One disadvantage of backtracking and branch and bound are that their performance is dependent on how the variables are ordered and how the assignments for each variable are ordered in the search tree. In the worst case, both of them have to search over the entire problem space, and look at all possible solutions.

Heuristic search is an approach to guide the selection of the variables in such a way that we find a good enough solution, if not the best solution, without having to search over all possibilities exhaustively. Heuristic searches are also used in the general CSPs, and the basic idea is to assign the most constrained variables first with the values that are least constraining. For example, if a person only highly prefers wine for drinks over coke or mango lassi or any other values, and for variables other than drinks, he is more flexible, then the variable "drinks" is assigned first, and it is assigned with the least constraining value, which is wine. Progressively, this approach is applied to other variables.

Implementation:

The implementation of our heuristic search is based on the approaches described in [1]. The paper uses the metrics "appropriateness" and "difficulty" to guide the variable assignments.

Appropriateness of a value of a variable:

Taking directly from the paper, "the appropriateness of a value $v \in D_i$ for a variable x_i is evaluated on the basis of the degree of the best possible joint satisfaction of the constraints referring to x_i." To find best joint satisfaction, we should fix $x_i = D_i$ and find the max of joint satisfaction over all possible solution instances with x_i fixed.

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

In our example, to find appropriateness of value a2 for the variable M, we follow these steps:
1) Find all possible instances with M=a2. These instances are: (a1, a2, a3), (a1, a2, b3), (b1,a2, a3), (b1, a2, b3)
2) Eliminate all constraints except the ones referring to the variable M, i.e. eliminate C2.
3) For C1 and C3, the best joint satisfaction is by (a1, a2, a3) is , with joint_sat = 1.0.
4) Therefore, appropriateness for M=a2 is 1.0

In a similar way, we can find the appropriateness of a value assignments with other values fixed. For example, fixing A=a1, the appropriateness of M=a2 is found by getting best joint satisfaction of all constraints containing the variables A and M, i.e. C1. The best joint satisfaction is 1, which is the only joint satisfaction in this case because our example has binary constraints with 2 variables, and a pair of variables has only 1 corresponding constraint.

Difficulty of a variable:

Difficulty of a variable is simply the sum of appropriateness of each of its values. In our example, difficulty of variable M = appropriateness(a2) + appropriateness(b2) = 1.0 + 0.5 = 1.5

Partial joint satisfaction metric:

This is a metric we came up with, while discussing on the upper bound of a partial instantiation during branch and bound search. The paper [1] suggests using "appropriateness" as an upper bound for a partial instantiation while assigning a variable, but we thought we can find a more efficient metric. The metric is similar to appropriateness, but while calculation, we don't need to enumerate all possible instantiations for the purpose of finding the best joint satisfaction.

This is a step-by-step way of calculating this metric, for given partial assignment (x1 =v1,x2= v2,x3= v3, …):
1) If length of partial assignment is less than the number of variables per constraint, partial_joint_sat = 1.0 (in case of binary FCSP, metric = 1.0 when only one variable is assigned).
2) If length >= number of variables per constraint, find all the constraints with the combinations of those variables.
3) Then, take values of satisfaction of each of the constraint, and find the joint satisfaction of those values. For example, if joint satisfaction approach is "productive", multiply all those values. This is the metric

Now, given all these metrics, let's discuss about the heuristic search algorithm.

The steps of the algorithm are:
a. For each variable left to be assigned (given a list of already assigned variables) find the difficulty. We get a list of difficulty of all unassigned variables given already assigned ones.
b. Choose the variable with the least difficulty to assign a value

    c. Within that chosen variable, find the value that has the highest appropriateness
    d. Assign that value to the variable
    e. Go to step a., fixing the current chosen variable

Running heuristic search in our example:

    a. Find difficulty and appropriateness for each variable and values (notation: $a_x(v)$ = appropriateness of value v for value v, d_x = difficulty of variable x):
    $a_A(a1)=1$, $a_A(b1)=0.8$, and thus $d_A=1.8$;
    $a_M(a2)=1$, $a_M(b2)=0.5$, $a_M(c2)=0.8$, and thus $d_M=2.3$;
    $a_D(a3)=1$, $a_D(b3)=1$, and thus $d_D=2$.

    b. Assign the least difficult (= highly constrained) variable A, with least constraining assignment a1. So, A = a1.
    c. Given A=a1, find difficulty of variables B and C with appropriateness of respective values (some appropriateness values decrease as a result of A=a1):
    $a_M(a2)=1$, $a_M(b2)=0.4$, $a_M(c2)=0.2$, and thus $d_M=1.6$;
    $a_D(a3)=1$, $a_D(b3)=0.7$, and thus $d_D=1.7$.

    d. Assign M=a2.
    e. Now, given A = a1 and M = a2, find appropriateness of values of D.
    $a_D(a3)=0.7$, $a_D(b3)=0.7$
    f. Therefore, we can choose any value for D. Say we choose D = a3.
    g. Final assignment is (a1, a2, a3)

Benchmarking results

Benchmark requirements
We measured the performance of Backtrack, Branch and Bound and Heuristics Search algorithms against problems differing in size and complexity as referenced from [3]. Section A discusses our benchmarking result against problem size and complexity. Section B discusses comparison of our algorithms.

Performance Metrics for comparison:
- Number of constraint table lookups (constraint checks):
  - Every time the algorithm queries the constraint table for the satisfaction degree of an instantiation, it incurs a cost of lookup. This metric serves a good measure for algorithm performance.
- Search nodes:

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

- o Every time an algorithm assigns a value to a variable, it is equivalent of expanding the search node in the search tree except for the last instantiation. This serves a good measure of performance for tree based search algorithms like backtracking and branch and bound.
- Runtime
  - o Running time for the particular algorithm on a relative scale.

<u>Problem Characterization (For varying problem size and complexity):</u>
1. Problem Size:

Problem size can be measured as the product of the domain sizes for each variable, i.e. the number of combinations of values that could be generated as potential solutions.

1. n - number of variables
2. m - size of domains (fixed for now)

Problem size = $m \wedge n$

This metric corresponds to the size of the search tree created for tree-based algorithms.
2. Problem Complexity:

We characterized problem complexity using two features of the constraint satisfaction problem:

1. Connectivity of the constraints:

The connectivity of the constraint graph that represents the problem is the proportion of variables that are related by a constraint. We vary this parameter in steps of 20% from not connected to fully connected. For the sake of simplicity, we only use binary constraints.

If a graph is 50% connected this implies that half of the variables in the problem have constraints defined between them.

Since there is no constraint between the unconnected variables, we assign the satisfaction value of 1 for all the constraints between the unconnected variables for the problem.

A fully connected problem is therefore the most complex to find the solution assignments as each pair of variables have to be satisfied with the instantiation of each of the other variables.

2. Tightness of the constraints:

Tightness for a given problem is the proportion of assignment of variables disallowed by each constraint.

Suppose the size of a domain (m) = 6 then number of tuples in a binary constraint = m*m = 36
(a1,b1), (a1,b2), … (a2,b1),(a2,b2) … (a6,b6) = 36 values in C1

If tightness == 0.1 => 0.1*36 = 4 tuples are missing.
If tightness == 0.9 => 0.9*36 = 32 tuples are missing.

The characteristic of tightness relates to the difficulty of finding solution to a given FCSP since the solution space becomes narrower as tightness increases.

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Section A – Performance of APIs with Problem Size and Complexity.

A.1 Analysis of increase in problem size

Runtime Vs Problem Size

Number of Decision Variables	2	3	4	5	6	7
Backtracking	6.91E-05	0.007829905	0.104136944	0.761866093	9.279356956	60.36239195
Branch&Bound	0.000977993	0.005604982	0.017045021	0.131978035	1.287832975	7.957272053
Heuristics	0.001250029	0.022917032	0.277776957	3.025189161	31.74167705	342.457926

Runtime (ms) (relative)
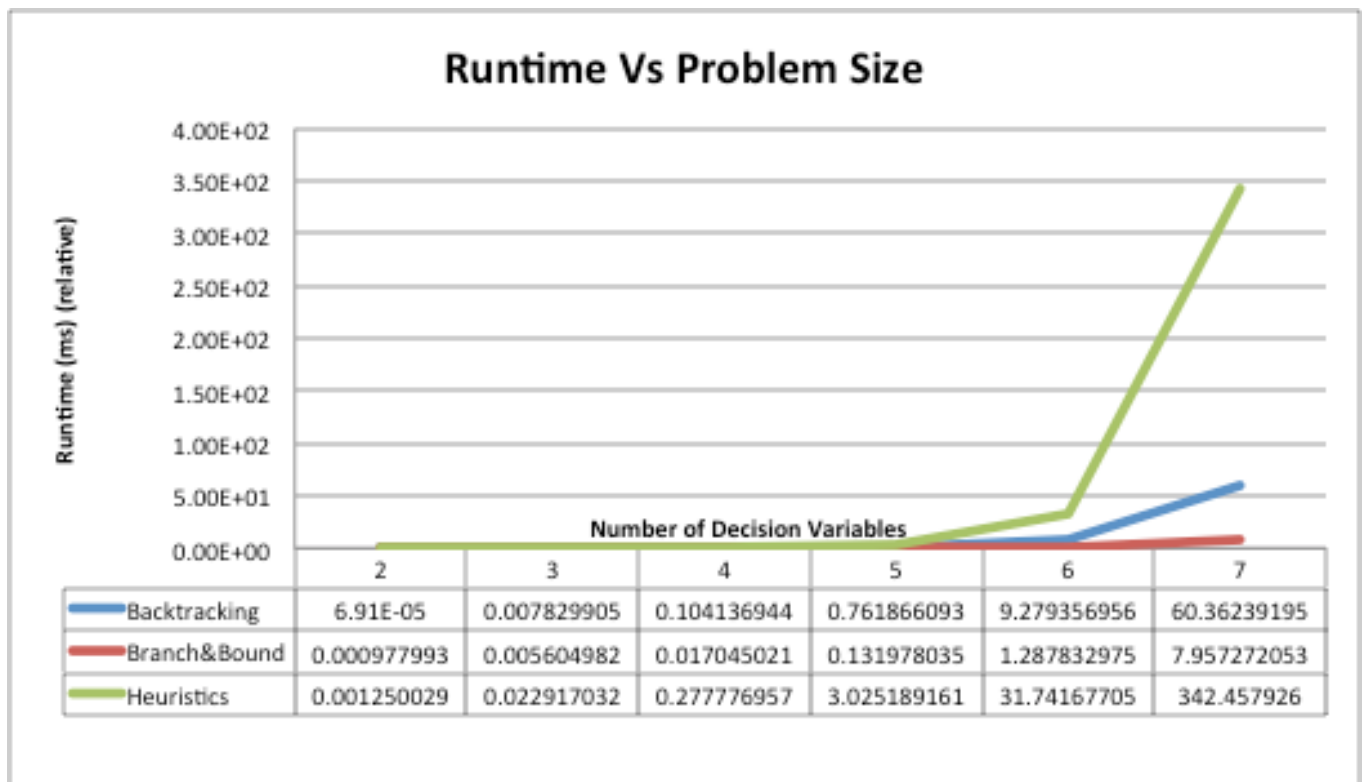
Figure 1 - Benchmarks results for performance of all algorithms relative to problem size.

As seen in Figure 1, the runtimes for Backtracking and Branch&Bound algorithms are more resilient with increasing problem size. We increase the problem size by increasing the number of decision variables and therefore increasing the search space of the problem space of the problem. Since heuristics search front-loads the work of calculation of appropriateness and difficulty in order to pick the variables in the correct order, it has the worst scaling effect with increasing problem size. On the other hand, tree based algorithms do better compared to heuristics search as only the required number of nodes are expanded to find the best solution. Branch&Bound scales the best because of the pruning advantage whereby it chops off a section of the search-space when the joint satisfaction of the partial solution belonging to that sub tree falls below the lower bound.

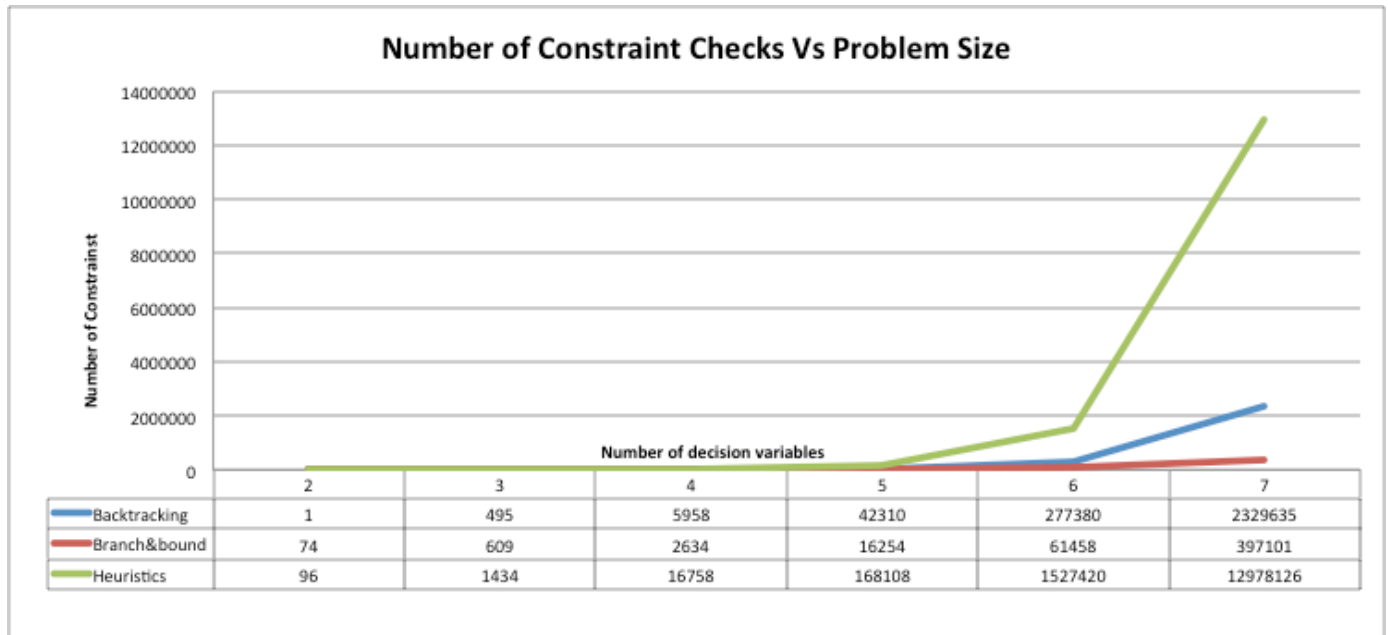Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Number of Constraint Checks Vs Problem Size

	2	3	4	5	6	7
Backtracking	1	495	5958	42310	277380	2329635
Branch&bound	74	609	2634	16254	61458	397101
Heuristics	96	1434	16758	168108	1527420	12978126

Figure 2 - Performance comparison of the APIs with increasing problem size

The number of lookups in the constraint table is directly proportional to the performance of the algorithms as seen when the Figure 1 and 2 are compared. The relative graphs of all algorithms are the same in both the cases. Since the number of lookups increase with increasing search-space of the problem, algorithms that search the entire solution space like heuristics search have the maximum increase in constraint table lookups.

Figure 3 shows the relationship between increase in problem size and the number of variable assignments among all the algorithms implemented. It is not surprising to see that the number of variable assignments for heuristics search is the same as the number of variables in the problem as all computation for variable ordering and value ordering is done as part of preprocessing the heuristics value. On the other hand, the tree based algorithms of backtracking and branch and bound show a linear increase in variable assignments as the search tree size increases. Clearly, backtracking is less optimal than branch and bound due to the advantage of pruning in the later case.
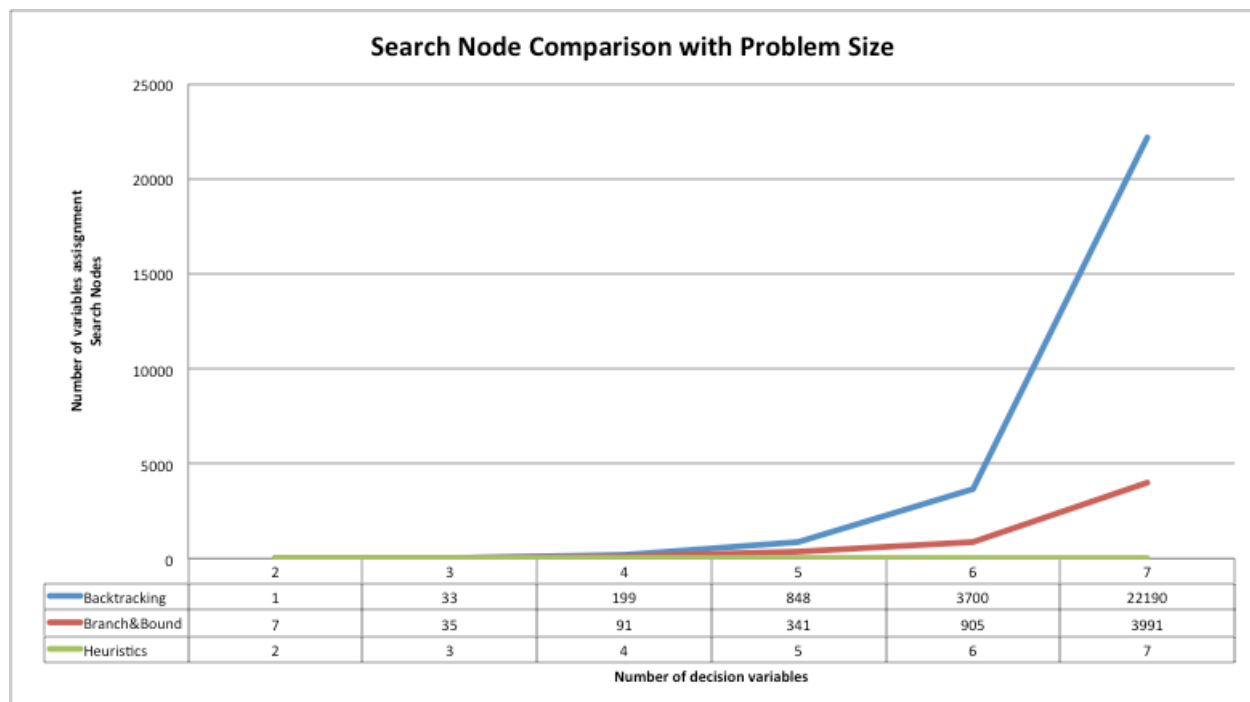
Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Figure 3 - Comparison of search nodes with increasing problem size

A.2 Analysis of increase in problem complexity

As discussed in the beginning of the benchmarking section, we have found two suitable parameters for characterizing complexity of the problem. Connectivity is the connectivity of the graph representation of the constraint satisfaction graph of the problem. The more connected the graph is – the more relationships exists between the decision variables. In general, the branch and bound implementation scales well with increasing complexity of the problem and pruning of unnecessary variable checks lead to better results as complexity increase with a constant problem size. Heuristics search as expected has no relationship with connectivity of the problem as it checks for variable ordering by checking all possible relationships in the problem before it assigns a variable. Backtracking performs well with increasing complexity since the search for greater than alpha solution leads to termination of the search before the entire search tree is visited.

We also measure the performance of the implementation with respect to the 'tightness' of the problem. The tighter a problem is the narrower is the possible solution space compared to the entire solution space. This property is exploited by branch and bound as well as backtracking with alpha-best solution, however, heuristics search remains unaffected by the tightness except for a bump seen in Figure 5 – where the tradeoff for frontloading computation overshoots the variable ordering based optimal partial instantiation and in this case, it has to backtrack several times to recalculate the ordering. The increase in number of constraint checks with a constant problem size is therefore seen high around the 50% tightness ratio.
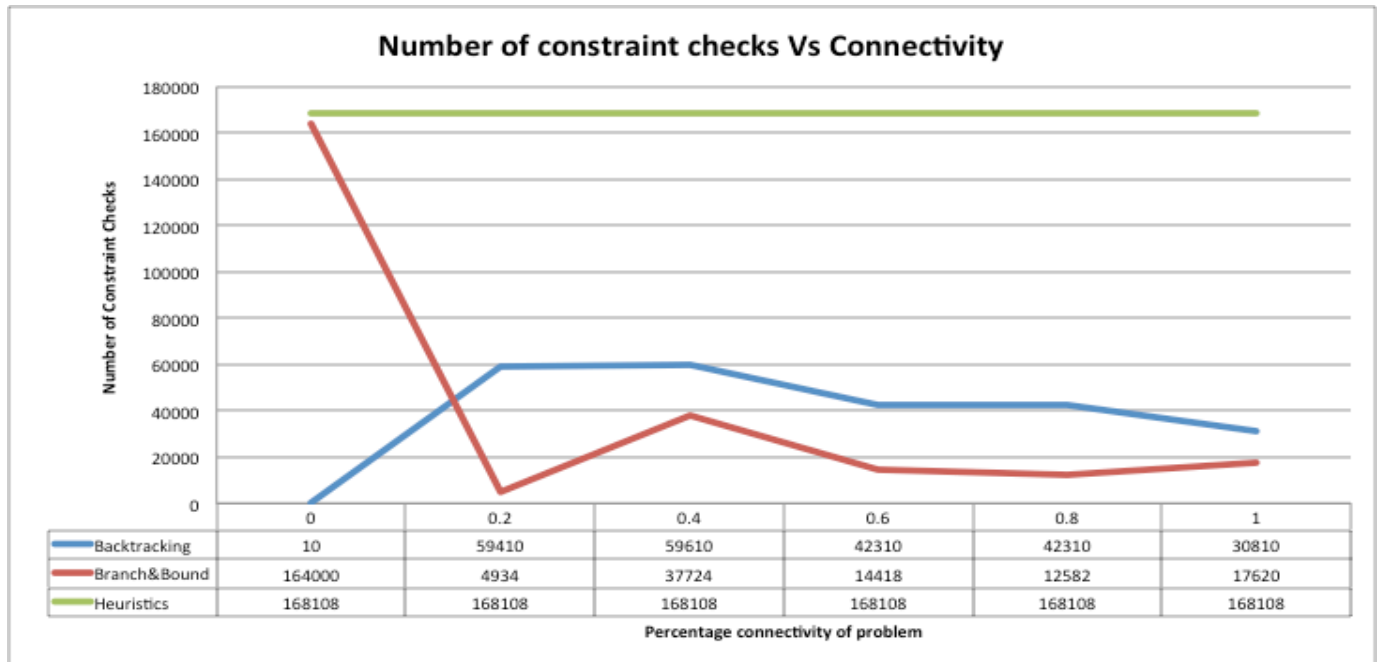
Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Number of constraint checks Vs Connectivity

	0	0.2	0.4	0.6	0.8	1
Backtracking	10	59410	59610	42310	42310	30810
Branch&Bound	164000	4934	37724	14418	12582	17620
Heuristics	168108	168108	168108	168108	168108	168108

Percentage connectivity of problem

Figure 4 - Performance comparison with connectivity of the constraint graph

Number of Constraint Checks vs Tightness

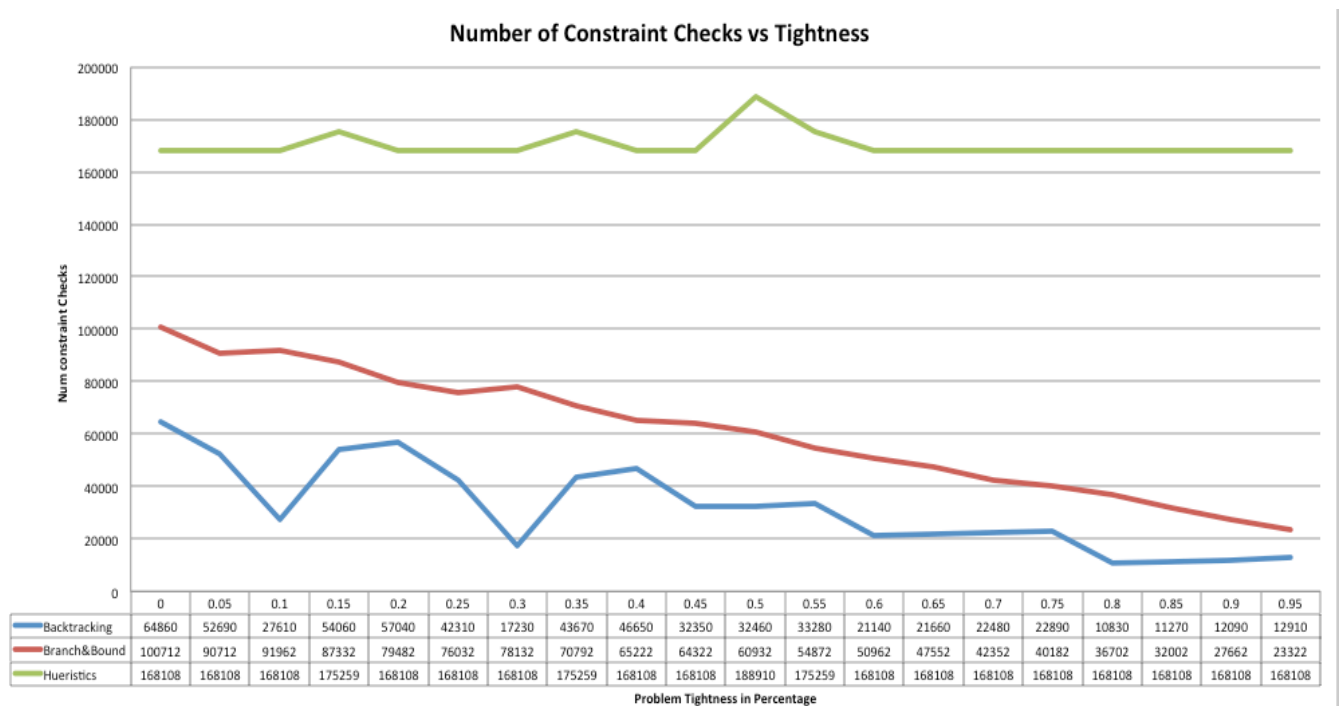|	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
Backtracking	64860	52690	27610	54060	57040	42310	17230	43670	46650	32350	32460	33280	21140	21660	22480	22890	10830	11270	12090	12910
Branch&Bound	100712	90712	91962	87332	79482	76032	78132	70792	65222	64322	60932	54872	50962	47552	42352	40182	36702	32002	27662	23322
Hueristics	168108	168108	168108	175259	168108	168108	168108	175259	168108	168108	188910	175259	168108	168108	168108	168108	168108	168108	168108	168108

Problem Tightness in Percentage

Figure 5 - Performance comparison with narrowing solution space (tightness) within a problem of constant size

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Section B – Comparison of Performance of implemented Algorithms

Comparison of Backtracking and Branch&Bound Performance

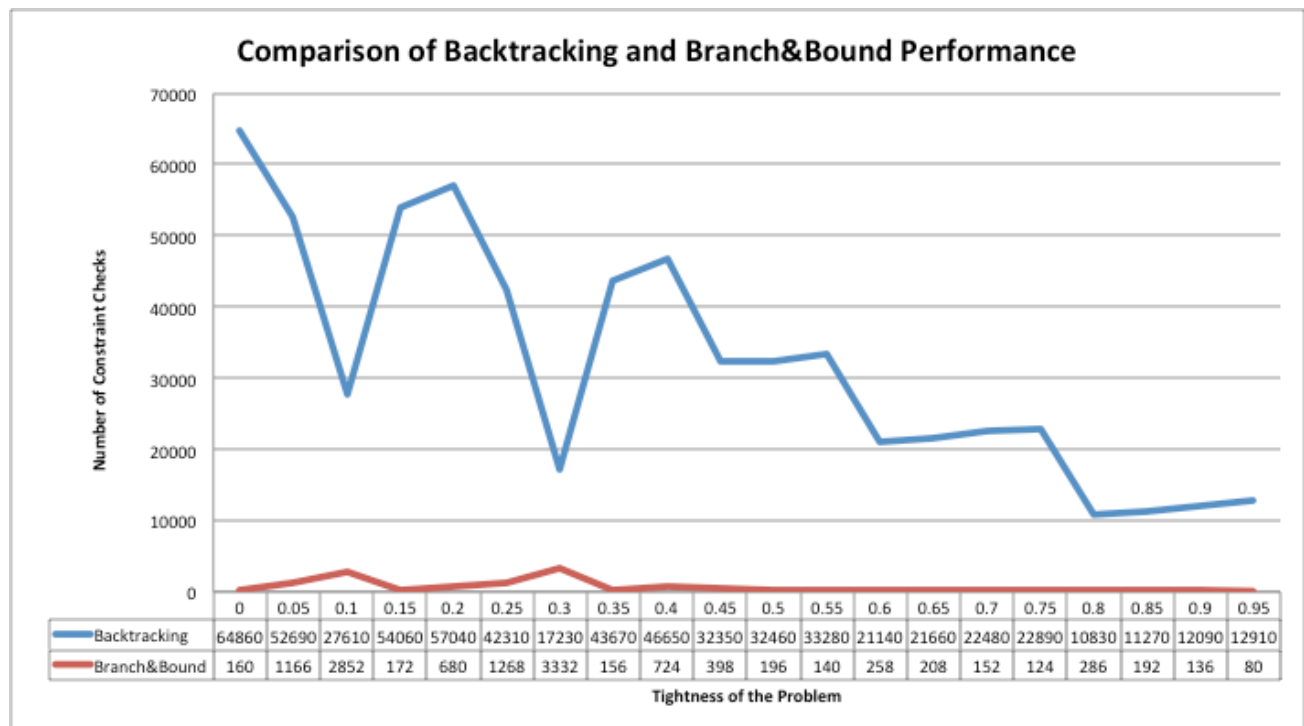| Tightness of the Problem	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
Backtracking	64860	52690	27610	54060	57040	42310	17230	43670	46650	32350	32460	33280	21140	21660	22480	22890	10830	11270	12090	12910
Branch&Bound	160	1166	2852	172	680	1268	3332	156	724	398	196	140	258	208	152	124	286	192	136	80

Figure 5 – Comparison of backtracking and branch&bound with problem complexity

We compare two similar algorithms we have implemented that use tree based search to find the best solution or the best alpha solution. The performance of branch&bound is significantly better than backtracking due to pruning sub-optimal sub trees and reducing the number of lookups. However, backtracking searches for the entire tree in the worst case. As the narrowness of the solution space within the problem space increases, branch and bound is able to utilize the fact to prune the infeasible parts of the tree effectively increasing performance whereas backtracking lacks that feature and still may end up searching the entire tree for best alpha solution.

We compare the features of heuristic search with branch and bound in the Figure 7 shown below. As seen before, pruning in branch and bound decreases the search space effectively – heuristics search however, is not able to utilize the narrow search space of the problem to its advantage and still front-loads all the computation regardless of the nature of the problem.

In figure 8, we analyze the characteristics of various algorithms by comparing the number of variable assignments done or the number of search nodes expanded during the execution of the algorithm. As seen in the graph, heuristics search is the most efficient in this case as it orders the variable before solving the problem and it is able to get the correct order based on the computation for appropriateness for every value.
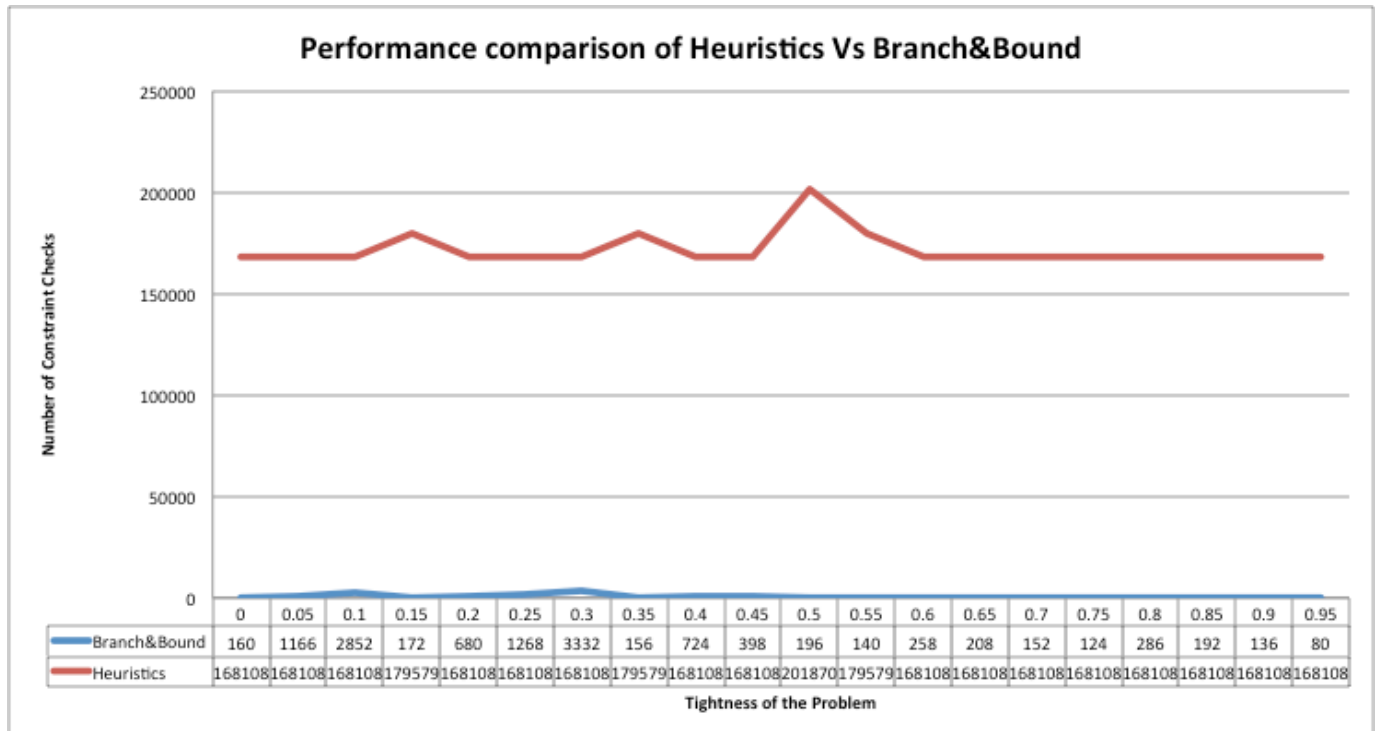
Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Performance comparison of Heuristics Vs Branch&Bound

	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
Branch&Bound	160	1166	2852	172	680	1268	3332	156	724	398	196	140	258	208	152	124	286	192	136	80
Heuristics	168108	168108	168108	179579	168108	168108	168108	179579	168108	168108	201870	179579	168108	168108	168108	168108	168108	168108	168108	168108

Tightness of the Problem

Figure 6 - Analyzing the behavior of heuristics search and branch&bound with problem complexity

Number of search nodes Vs Connectivity

	0	0.2	0.4	0.6	0.8	1
Backtracking	4	1190	1193	848	848	618
Branch&Bound	1558	89	705	306	283	371
Heuristics	5	5	5	5	5	5

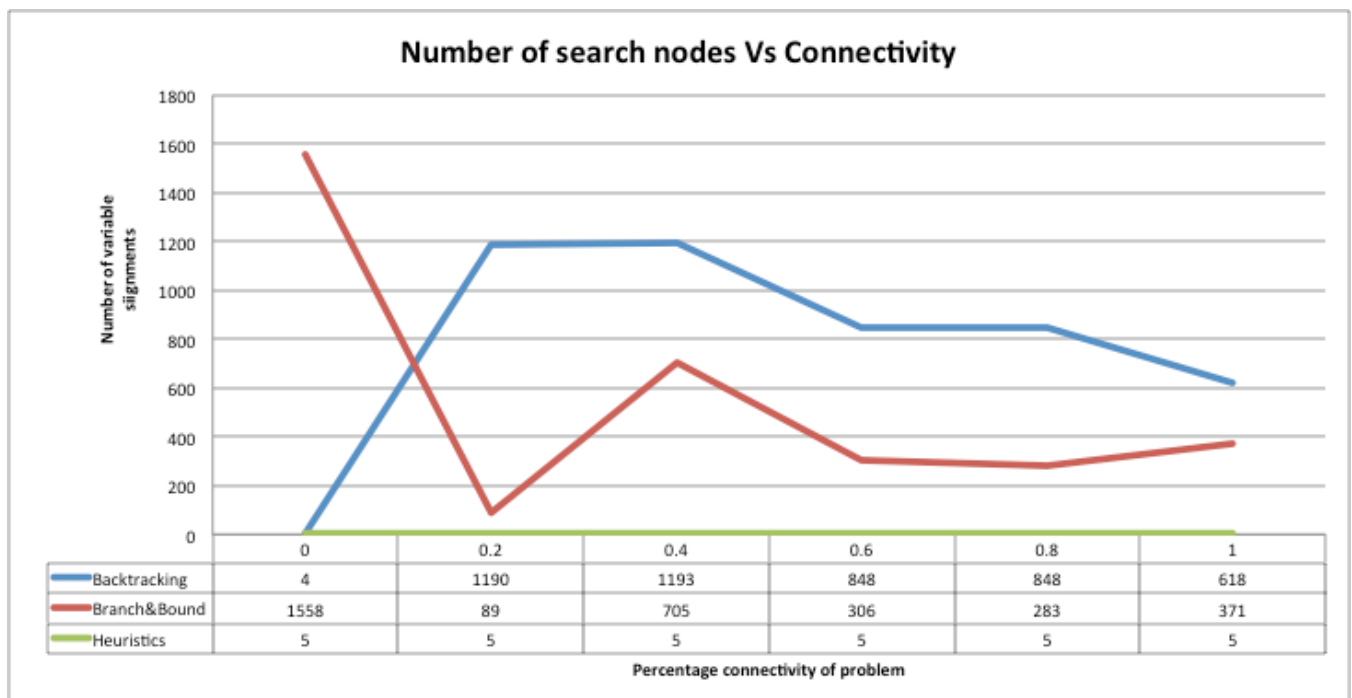Percentage connectivity of problem

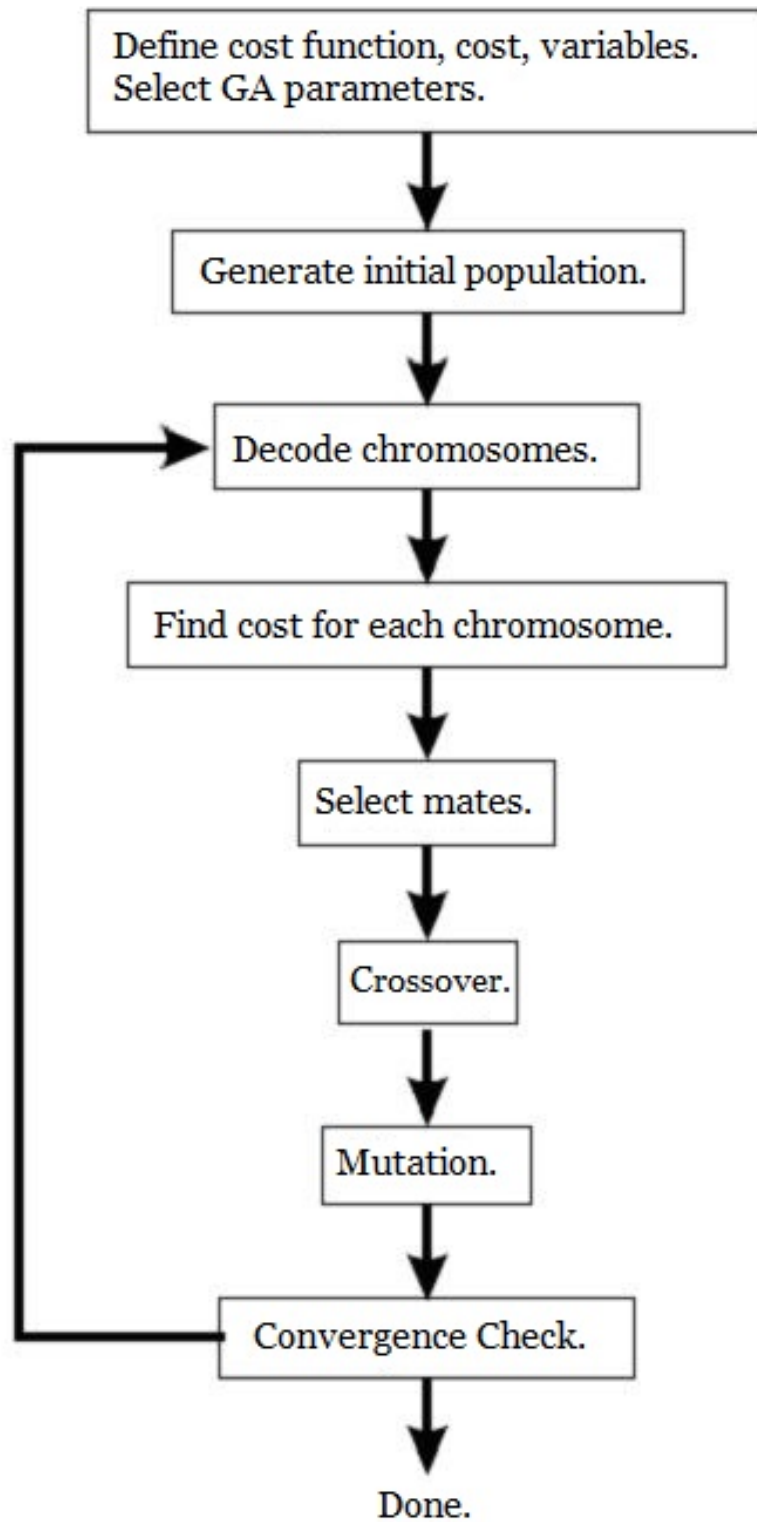**Figure 7 - Analyzing characteristics of algorithm based on problem connectivity**

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Genetic Algorithm Approach:

In this section, we introduce a genetic algorithm approach to fuzzy constraint satisfaction problems. Because such algorithms are (more or less) tailored to the specific problem under consideration (fitness/evaluation function in particular), we consider two new problems, and the algorithms to solve each. We also consider benchmarking and analysis in the context of these problems.

1. A quick introduction to Genetic Algorithms (GA)
Genetic algorithms (GA) are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. They mimic the process of natural selection, and are routinely used to generate useful solutions to optimization and search problems. In the figure below, we show the components of a basic GA, then proceed to explain each component.

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Define cost function, cost, variables.
Select GA parameters.

↓

Generate initial population.

↓

Decode chromosomes.

↓

Find cost for each chromosome.

↓

Select mates.

↓

Crossover.

↓

Mutation.

↓

Convergence Check.

↓

Done.

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Cost function and cost: **Each problem has a cost function. The cost is a value for fitness, and is assigned to each solution.**
Chromosomes and genes: **Genes are usually encoded in bits. A chromosome is an array of these genes. It can also be a solution.**
Initial population: **An initial population is created from a random selection of chromosomes. The number of generations needed for convergence depends on the random initial population.**
Chromosome decoding and finding the cost: **To find the assigned cost for each chromosome a cost function is defined. The result of the cost function is called a cost value. Finally, the average of cost values of each generation converges to the desired answer.**

Cross-over and mutation: The chromosomes with a higher fitness (lesser cost) value are used to produce the next generation. The offsprings are a product of the father and the mother, whose composition consists of a combination of genes from them (this process is known as "crossing-over"). Mutation changes individual chromosomes to provide additional variability to the reproduction process.

Next generation: If the new generation contains a chromosome that produces an output that is close enough or equal to the desired answer then the problem has been solved. If this is not the case, then the new generation will go through the same process as their parents did. This will continue until a solution is reached.

2. Advantages and Disadvantages of GA
Here we list some of the (general) advantages and disadvantages of genetic algorithms, as well as some other limitations particular to FCSP frameworks.

2.1 Advantages
- GA can solve every optimisation problem which can be described with the chromosome encoding.
- It solves problems with multiple solutions.
- Since the genetic algorithm execution technique is not dependent on the error surface, we can solve multi-dimensional, non-differential, non-continuous, and even non-parametrical problems.
- Structural genetic algorithm gives us the possibility to solve the solution structure and solution parameter problems at the same time by means of genetic algorithm.
- Genetic algorithm is a method which is very easy to understand and it practically does not demand the knowledge of mathematics.
- Genetic algorithms are easily transferred to existing simulations and models.

2.2 Disadvantages
- Certain optimisation problems (they are called variant problems) cannot be solved by means of genetic algorithms. This occurs due to poorly known fitness functions which generate bad chromosome blocks in spite of the fact that only good chromosome blocks cross-over.
- There is no absolute assurance that a genetic algorithm will find a global optimum. It happens very often when the populations have a lot of subjects.
- Like other artificial intelligence techniques, the genetic algorithm cannot assure constant optimisation response times. Even more, the difference between the shortest and the longest optimisation response time is much larger than with conventional gradient methods. This unfortunate genetic algorithm property limits the genetic algorithms' use in real time applications.
- Genetic algorithm applications in controls which are performed in real time are limited because of random solutions and convergence, in other words this means that the entire population is

improving, but this could not be said for an individual within this population. Therefore, it is unreasonable to use genetic algorithms for on-line controls in real systems without testing them first on a simulation model.

- GAs are of stochastic nature.
- A genetic algorithm has to be tailored to the problem (especially the fitness/evaluation function), and it is hard to come up with a general algorithm that can be used for various problems.
- The runtime and solution of a genetic algorithm depends on various factors: initial population, number of generations, mutation/crossover probability, selection function, etc. It is hard to come up with the "ideal" factors for getting fast runtime and right solution. At times, it can be a matter of experimentation.
- Benchmarking and analysis is hard for GAs. Even by keeping the given parameters (initial population size, number of generations, etc), due to the stochastic nature of GAs, we need to run the algorithm various time to get the best solution and the average runtime. To "grade" the performance of the algorithm, we have to keep changing the parameters, run the algorithm with new parameters various times, and keep track each time of the values for the measurements of interest (for example: runtime). In the end, we are left with the new task of plotting and averaging the results of the runs. Automating this process is not easy.

2.3.Other limitations particular to FCSPs

Let us consider a FCSP defined by a set of decision variables $X=\{X_1, \dots X_n\}$, a set of domains $D=\{D_1,\dots,D_n\}$, where D_i is the domain of X_i, and is supposed to be finite (with cardinality $|D_i|=d_i$) and a set of fuzzy constraints $C=\{C_1, \dots, C_k\}$. A solution to this FCSP is denoted by **x** and is a vector of values in $X=\{D_{1x}, \dots x_{Dn}\}$. The level of satisfaction for the constraints is defined as: $Sat(x)= \min_k \mu C_k(x)$, and the optimal solution **x***, so called "min optimal" is $Sat(x^*)=\max_x \min_k \mu C_k(x)$. In a fuzzy framework, solving a FCSP problem amounts to find at least one solution that is maximal for $>min$. Hence finding a solution for a FCSP is an optimization problem. Here, we are dealing with FCSPs where no natural order exists for the variable domains.

The satisfaction level of a min-optimal solution for a FCSP is said to be:
- **inconsistent**, if it does not satisfy at least one constraint.
- **fully consistent**, if it completely satisfies all constraints.
- **partially consistent,** if there is at least one constraint -not fully satisfied- and the satisfaction levels of all constraints are positive.

For the min-based FCSPs, the solutions can be discriminated by (at most) L levels of consistency. In each equivalence class, all solutions share the same satisfaction level which is one of the least satisfied constraint(s). Two solutions in the same equivalence class are not discriminated, even if they could be distinguished by considering other levels of satisfaction for the remaining constraints.

Solving a FCSP amounts, generally, to finding an instantiation for its variables that maximizes the satisfaction level of the least satisfied constraint. Indeed, the approach aggregates the constraints by means of the min operator. Since min-optimal solutions to a FCSP may be numerous, further refinements of the ranking of solutions are used. One such refinement is referred to as the *leximin* ordering. This refinement does not only take into account the least satisfied constraint, but also compares other satisfaction levels.

In a leximin order, a solution **x** is preferred to a solution **y**, if there is a threshold **α** such that for all **β < α**, the number of constraints satisfied by x (at level at least β) is equal to the number of constraints satisfied by y, and it satisfies more constraints than y at level α. This is equivalent to comparing the cardinalities at increasing level cuts.

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

When it comes to the optimization function, the choice of the mapping is relevant for those algorithms that need a notion of distance such as genetic-algorithms (GA), in instances when roulette wheel or integral selection is used.

Paper [5] discusses in detail how the structure and property of the leximin affects the search strategies of non-enumerative algorithms, such as GA. The main conclusion of the paper is that the leximin optimization function, for FCSPs with non-ordered variable domains, usually displays very irregular "shapes" on the neighbourhood of many points of the solution space. This property affects the quality of results obtained with non-enumerative algorithms (such as GA) when applied without further information so as to guide the search with special codification.

Two examples of the main information that can be included in the genetic algorithms to improve the search process are:
- use of problem information to generate a good starting population
- and, choosing more suitable mappings for the optimization function.

The paper provides no guidelines as to how this can be done. leaving it as a topic for further research.

3. Application to problems

In the context of FCSPs, constraint satisfaction treated as an objective function can be optimized (maximized) with the use of GAs. In this section, we show how this can be done by considering two problems.

3.1 Problem 1: University Course Selection

This problem appears on paper [4], by Dubois et Al.

Problem: An university course must consist of seven sessions: x1 lectures, x2 exercise sessions and x3 training sessions.There have to be about 2 training sessions, i.e. ideally 2, possibly 1 or 3. Dr.B prefers to give 3 or 4 exercise sessions and Prof.A wants to give about 4 lectures, i.e. ideally 4, possibly 3 or 5. The preference of Prof.A is more important than the preference of Dr.B and it is less important than the constraints about a number of the sessions required.

Description: This problem can be modeled with a FCSP defined with three variables
$x_i \in D_i = \{0, 1, ..., 7\}$, $i = 1, 2, 3$, and four fuzzy constraints (with five satisfaction levels set at 0, 0.25, 0.5, 0.75, 1) as follows:

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

$$C_1: \quad \mu_{C_1}(x_1, x_2, x_3) = \begin{cases} 1 & \text{if } x_1 + x_2 + x_3 = 7 \\ 0 & \text{otherwise} \end{cases}$$

$$C_2: \quad \mu_{C_2}(x_3) = \begin{cases} 1 & \text{if } x_3 = 2 \\ 0.75 & \text{if } x_3 = 1 \text{ or } x_3 = 3 \\ 0 & \text{otherwise} \end{cases}$$

$$C_3: \quad \mu_{C_3}(x_2) = \begin{cases} 1 & \text{if } x_2 = 3 \text{ or } x_2 = 4 \\ 0.5 & \text{otherwise} \end{cases}$$

$$C_4: \quad \mu_{C_4}(x_1) = \begin{cases} 1 & \text{if } x_1 = 4 \\ 0.75 & \text{if } x_1 = 3 \text{ or } x_1 = 5 \\ 0.25 & \text{otherwise} \end{cases}$$

The relative priorities of the constraints are: Pr(C1)=Pr(C2) >Pr(C3) >Pr(C4). The objective of the problem is to find an arrangement for the sessions such that all constraints are satisfied to the maximal possible degree, i.e. the problem is to maximize:

$$\mu_C(x_1, x_2, x_3) = \min\left(\mu_{C_1}(x_1, x_2, x_3), \mu_{C_2}(x_3), \mu_{C_3}(x_2), \mu_{C_4}(x_1)\right)$$

Solution: It is easy to see that the best solution is given by x1=3, x2=3, x3=1. This solution satisfies all constraints to the degree of μc(x1,x2,x3) = 0.75

**Parameters:** The parameters used for this problem are described below:

Representation	List: [x1, x2, x3]
Crossover	cut and crossfill
Crossover probability	0.6
Mutation	Gaussian Mutator
Mutation probability	0.1
Scaling	Sigma Trunc Scaling
Survival selection	RouletteWheel
Population size	20

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Number of generations	500
Initialization of population	Random, range (0,7)
Termination condition	Given number of generation

Analysis and Benchmarking: We experimented with integer and real encoding of the chromosomes, while keeping the rest of the parameters the same. Due to the stochastic nature of GAs, ten independent runs were performed. The performance for both cases was plotted in the graphs below (**x-axis: number of generations, y-axis: algorithm's score**).

- **Integer encoding of chromosomes.**

Each time the GA was able to find the solution of the considered FCSP after around 100 generations. The total time elapsed was between 0.575 and 0.648 seconds.
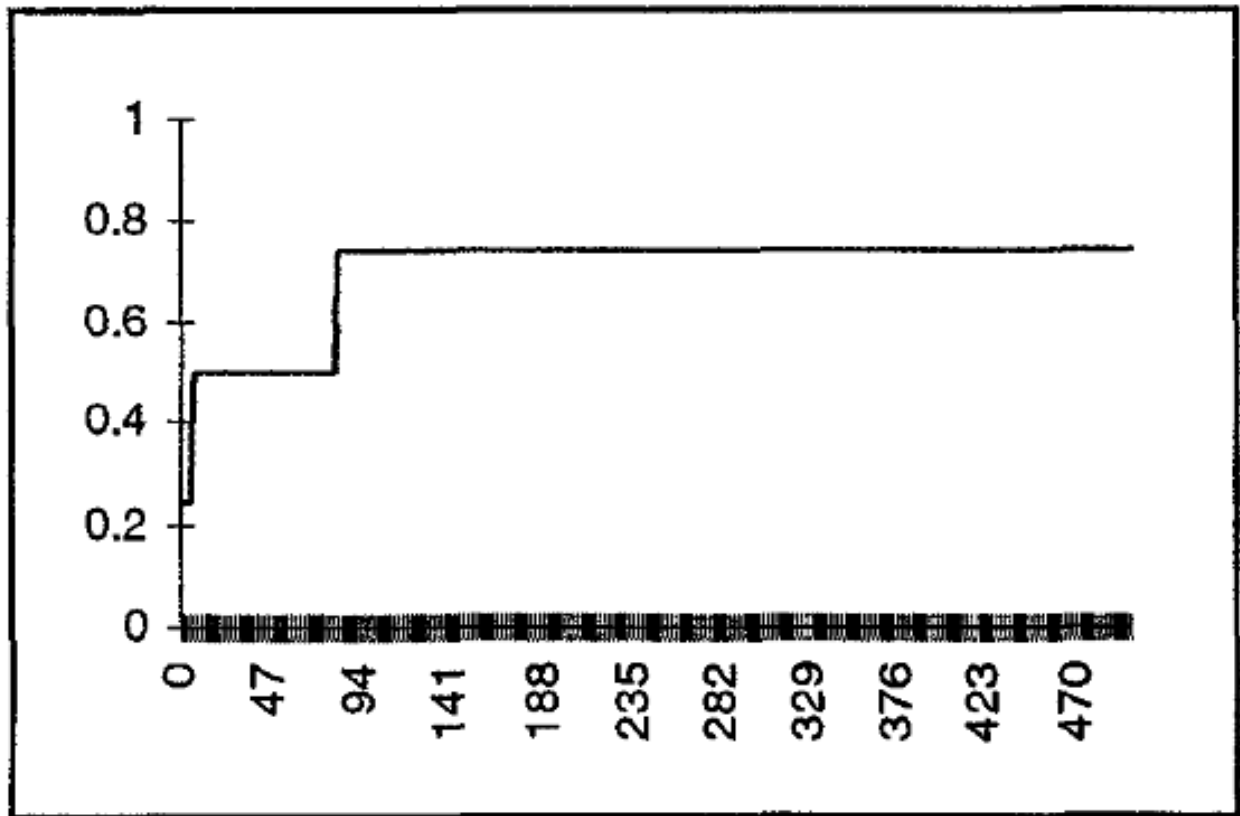
- **Real encoding of chromosomes.**

The solution was found on average during 30 generations. The GA performance was improved about three times. The explanation for this is that, as a result of using different granulation the initial population contained more fit chromosomes than in the previous case.

The total time elapsed remained similar to that of the previous part. This because of the way we structured the algorithm: the algorithm will stop running only after all the generations are run (500 generations in this case).

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics
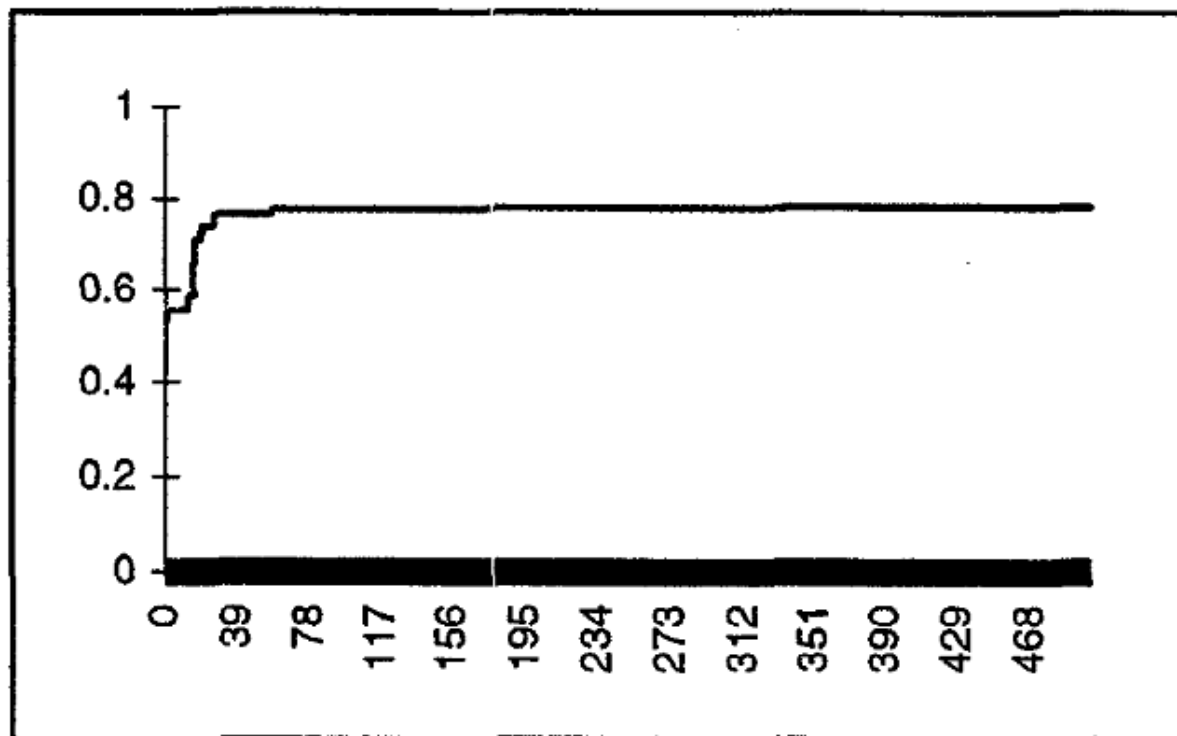
3.2 Problem 2: N-Queen Problem

This problem can be considered as an edge case problem for FCSP, since it involves crispy constraints [6].

Problem: The N-Queen problem is to place N queens on chessboard of size N by N such that no queen can attack any other queen. One queen is vulnerable to attacks from another queen if they share same row, column or diagonal.

Description: There are number of ways these N queens can be placed and each way is one solution. Each solution can be represented as a tuple of length N as (Q1,Q2,.........QN). Index positions of the tuple represents row positions and values represent column position. Reverse can also be chosen. Thus solutions are basically a permutation of the tuple (1,2,3,....N). As each queen will be placed on distinct row and column we do not have to worry about collision of queens on row and columns. We simply have to concern about whether any collision is possible on diagonal positions.

Solution: The fitness value for a correct solution in a N by N board will be N(N-1)/2, which actually represent there is no attack. In the extreme case, when all queens are in diagonal, the fitness value will become zero.

Parameters: The parameters used for this problem are described below:

Representation	permutation
Crossover	cut and crossfill
Crossover probability	1.0

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Mutation	swap
Mutation probability	0.02
Scaling	none
Survival selection	Replace the worst
Population size	100
Number of generations	5000
Initialization of population	random
Termination condition	Best Raw Score

Analysis and Benchmarking:

For a population of size=100 ,generations=5000 and taking values of N=4 to 64, execution time and the generation number where the solution was achieved were calculated. The results are presented in the table below.

N	Generation number	Time elapsed (in secs)
4	1	0.202
5	1	0.106
6	1	0.127
7	9	0.221
8	3	0.118
9	121	1.584
10	278	3.827
11	470	7.789
12	769	17.548
13	3	0.333
14	2183	51.104
15	8	0.911
16	850	28.528
17	2241	77.863

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

18	114	4.764
20	34	1.773
21	11	0.561
22	14	0.795
23	443	25.086
24	423	24.287
25	905	60.928
30	34	3.443
35	112	13.738
40	179	28.385
45	73	14.171
50	32	7.684
64	49	19.323

From the table, we see that the elapsed time and generation number depend on the initial population, and it varies. However, we can see that on average for large N, the elapsed time increases. These drawbacks for the GA were also mentioned in more detail in section 2.

In particular, we present the result of one run for N=64 (4096 chess squares), which took 49 generations to solve. The given solution was:

[45, 4, 50, 41, 9, 33, 19, 62, 48, 13, 30, 7, 31, 52, 44, 40, 0, 15, 18, 12, 51, 54, 38, 3, 56, 39, 35, 8, 29, 58, 36, 53, 14, 6, 17, 47, 11, 60, 10, 63, 61, 27, 37, 20, 46, 55, 43, 34, 22, 28, 26, 59, 21, 1, 32, 23, 16, 42, 25, 49, 2, 24, 5, 57]

The output is presented below:

```
>>> ============================== RESTART ==============================
>>>
Gen. 0 (0.00%): Max/Min/Avg Fitness(Raw) [20.57(25.00)/12.28(6.00)/17.14(17.14)]
Gen. 10 (0.20%): Max/Min/Avg Fitness(Raw) [59.57(55.00)/44.75(47.00)/49.64(49.64)]
Gen. 20 (0.40%): Max/Min/Avg Fitness(Raw) [57.17(58.00)/31.42(30.00)/47.64(47.64)]
Gen. 30 (0.60%): Max/Min/Avg Fitness(Raw) [69.56(62.00)/52.30(56.00)/57.97(57.97)]
Gen. 40 (0.80%): Max/Min/Avg Fitness(Raw) [69.37(62.00)/52.82(56.00)/57.81(57.81)]

        Evolution stopped by Termination Criteria function !

Gen. 49 (0.98%): Max/Min/Avg Fitness(Raw) [70.04(64.00)/53.46(56.00)/58.37(58.37)]
Total time elapsed: 19.323 seconds.
```

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics

Here, we can see how the fitness of the population is improved from one generation to the other. In this sense, this output presents the benchmarking results for the fitness score of each generation.

References:

[1] Fuzzy Constraint Satisfaction, Zs. Ruttkay.
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=343640

[2] Heuristics for Fuzzy Constraint Satisfaction, Hans W Geusgen, Anne Philpott.
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=499457

[3] Dynamic Flexible Constraint Satisfaction and its Application to AI Planning, Ian Miguel.
http://ianm.host.cs.st-andrews.ac.uk/docs/MiguelThesis.pdf

[4] *Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty*, Didier Dubois , Hélène Fargier , Henri Prade
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.7538

[5] *Structure and Properties of Leximin FCSP and its Influence on Optimisation Algorithms*, João Moura Pires, Fernando Moura Pires, Rita Almeida Ribeiro.
http://www.researchgate.net/profile/Joao_Pires5/publication/247659755_Structure_and_properties_of_leximin_fcsp_and_its_influence_on_optimisation_problems/links/00b4952d40b46b5062000000.pdf

[6] *Performance Analysis of N-Queen Problem using Backtracking and Genetic Algorithm Techniques,* Vikas Thada, Shivali Dhaka http://research.ijcaonline.org/volume102/number7/pxc3898667.pdf

Pallavi Mishra
Pramod Kandel
Manushaqe Muco
16.412 – Cognitive Robotics