Constraints & Optimization: A Gentle Introduction for the Uninitiated but Curious Undergraduate Student

Jonatan M. Contreras
University of Texas at El Paso
Constraints Research and Reading Group (CR2G)
Dr. Martine Ceberio

Constraints & Optimization: A Gentle Introduction for the Uninitiated but Curious Undergraduate Student

If you are reading this, chances are you just joined a research lab that deals with constraint solving, constraint satisfaction, and/or optimization, and want to become familiar with the large, complex world of these subjects. If you are feeling anything like I did when I first saw the topic expressed by the experts at the University of Texas at El Paso's Constraint Research and Reading Group (CR2G) lab, you may feel confused or overwhelmed by it. The goal of this tutorial is to (hopefully) introduce the topics involved with this type of research in a way that is informative but not overwhelming.

Let's start with what constraints are:

Constraints is a word we may not say out loud often, but constraints are something we deal with every day. For example, if a group of friends are trying to get together, the group will begin to look through each other's schedule in order to find a common meeting day and time; this is *satisfying* constraints; that is, out of the available meeting times every friend provides, the goal is to find at least one meeting time that is in everyone's availability. When shopping, if you were to constrain an item you want to buy to buying above a certain amount of pounds of that item but also constrain your spending to under a certain amount of dollars, you would be modeling a constraint problem and subsequently *solving* those constraints (certain amount of pounds and keep spending under a certain amount). The terms *satisfying* and *solving* are not trivial. These are the terms used to differentiate between different types of constraints; those that are discrete and those that are continuous. That's the main difference between constraint satisfaction and constraint solving, but before we discuss those techniques in detail let's first make sure the meaning of discrete and continuous is clear.

Discrete and Continuous:

You can think of discrete as something countable. Going back to the scheduling example, if we define people's availability as "time slots," then there are a finite, or discrete, number of time slots that a person can give. If the time slots are one-hour time slots between 9 A.M. and 9 P.M. for a range of four days, then the least constrained possible person would give 48 different time slots (12 one-hour time slots times four days). The most constrained possible person would give 0 time slots.

Continuous is a concept that we tend to be more familiar with. The real number line, for example, is continuous. That is, the numbers between 0 and 1 never end; they are continuous. In the shopping example, measurements like pounds can be continuous; there is an infinite number of measurements a weight measurement can give you between 0 and 1 pounds, 1 and 2 pounds and on and on. Although we round our currency to cents, money spent can usually be considered continuous as well.

Differentiating between discrete and continuous constraints is important because different techniques will be more useful depending on what type of constraints you have. If your constraint problem deals with discrete constraints, then constraint satisfaction techniques will be more useful. If your constraint problem deals with continuous constraints, then it is constraint solving techniques that will prove more useful.

Solving Constraint Satisfaction/Solving Problems:

Before we discuss how to solve Constraint Satisfaction/Solving Problems (CSPs), we need to see what a solution should look like. Informally, a solution to a CSP is a set of values that will satisfy our constraints. In the scheduling example, a set of solutions would be time slots that fit everyone's schedule. In the shopping example, the solution would be a certain weight of pounds. There can be no solutions, one solution, or many solutions. We will begin by seeing how Constraint Satisfaction is done followed by how Constraint Solving is done.

Constraint Satisfaction Techniques:

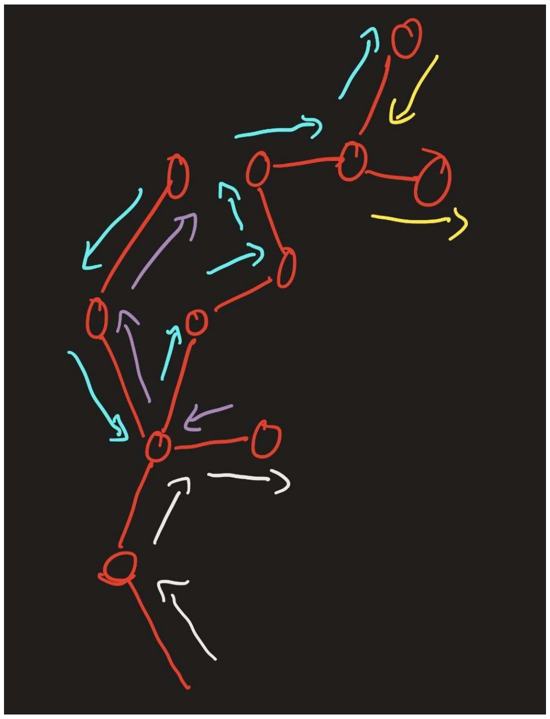
There are two main ways to solve constraint satisfaction problems: Systematic Search and Consistency techniques.

Systematic Search

With Constraint Satisfaction Problems, constraints have been imposed and we are attempting to find possible values that satisfy every constraint. Systematic Search then *searches* through all the combinations of those possible values and tests them to see if they satisfy all the required constraints. There are two main methods of doing this; Generate and Test and Backtracking.

- Generate and Test - This method creates random combinations of possible values and tests them until all constraints are satisfied. This algorithm is very inefficient, especially as your possible values increase, because going through all possible combinations can become very costly in terms of time.

- Backtracking - This method also tests combinations of the values, but continues forward a bit differently. With Generate and Test, a set combination of possible values is created and then tested. With backtracking, we start with one possible value, try it, and if it works, we move forward with the combination of that value and another value. If that next value doesn't work, we step back, or *backtrack*, to the most recent successful combination (the one value at this point) and try another value. The method continues to do this with as many combinations as it needs to try until a solution is found. So, in a way, the method is trying things, "learning" they don't work, then tracing back its steps and trying another value.



Backtracking Algorithm

The different colored arrows demonstrate a path that the algorithm is trying. When it finds zero success, it backtracks (denoted with a different colored arrow) and uses a different set of combinations. The algorithm begins with the white arrows and ends with the yellow arrows.

Consistency Techniques

These techniques use graph notions in order to represent the CSP and implement the techniques; this is a topic usually introduced to Computer Science students in Discrete Mathematics, so I will leave the more formal definitions in Appendix A. For this section, I will just explain them informally.

These techniques are called consistency techniques because they each try to maintain the possible solutions *consistent* with the constraints being imposed. In the scheduling example, if a friend gives the “time slots” 9 AM - 10 AM, 12 PM - 1 PM, and 5 PM to 6 PM, but there is a constraint that the group cannot meet after 5 PM, then the 5 PM to 6 PM time slot is not *consistent* with the constraints. *Node Consistency* is when the possible values for the solution are not consistent with the constraints and thus removed.

Now let’s say that not all friends get along and we don’t want to have friend A at the same get together as friend B. This is a constraint where friend A cannot be at the same get together as friend B. So when friend A’s availability includes a time slot, say 2 PM to 3 PM, and friend B provides that same time slot, then we know that the time slot 2PM to 3 PM does not satisfy the constraint imposed by the two friends. When there is a relationship between variables (here the variables are the group of friends), that relationship can be considered a constraint (the relationship here is the fact that friend A and friend B cannot be at the same get together). *Arc Consistency* is when we remove all the possible values for the solution that do not meet this type of relationship constraint.

Finally, let’s say we want a set of relationships to be satisfied by our values. Let’s say that friend A and friend B cannot be at the same get together; that friend B and friend C *should* be at the same get together; and that friend C and friend D also cannot be at the same get together at the same time. *Path Consistency* takes these types of relationships into consideration and makes sure that the possible values for the solution satisfies all of these relationships.

It is important to note that even after using all of these consistency techniques, it is still possible for there to be possible values being considered for the solution that are not consistent with the constraints. Thus, usually, these techniques are used alongside other techniques like systematic search for example. And there are other techniques that are more advanced that are used; but most of them use these two paradigms, Systematic Search and Consistency Techniques, to build on top of. Thus, it is okay to just know these for now.

Constraint Solving Techniques:

Let’s say that you are going to buy healthy snacks for the week. As a part of your purchase, you want to buy nuts. You want to buy pistachios and almonds and you want to spend 10 dollars. You also like almonds more than you like pistachios so you decide to buy twice as many almonds as pistachios. Let’s also say that almonds cost 5 dollars a pound and pistachios cost 4 dollars a pound. What is the quantity of pistachios and almonds that you should buy? We can model this as a system of equations. Bear with me on the math; it won’t be too difficult.

Let’s say P stands for pistachios (lbs) and A stands for almonds (lbs). Then our first statement, “you only want to spend 10 dollars,” can be modeled as:

$$4P + 5A = 10$$

This means that the total pounds of pistachios bought times 4 (since each pound costs 4 dollars) plus the total pounds of almonds bought times 5 (since each pound costs 5 dollars) needs to equal 10 dollars.

Now we said we wanted to buy twice as many almonds as we did pistachios. That can be represented like this:

$$A = 2P$$

This means that for every one pound of pistachios bought, we buy two pounds of almonds. We can rearrange this equation to look like this:

$$A - 2P = 0$$

Using this equation, we can get the following system of equations:

$$4P + 5A = 10$$
$$A - 2P = 0$$

For those that are familiar with systems of equations, there are various ways to solve this. If you aren’t familiar with systems of equations or don’t remember how to solve one, don’t worry. This is more for illustration purposes.

P and A are variables that represent pounds of pistachios and almonds, respectively. Each of those variables can take on any value that is between 0 and a number that is bounded by either your income or physical reality (let’s call this number 100, since I can’t fathom a store maintaining more than 100 pounds of pistachios or almonds in their inventory). As we mentioned earlier, these are continuous constraints because these variables can take on any value between 0 and 100 (1.5, 11.532, 15.5392, and on and on) pounds. The equations above *are* the constraints we are imposing on these variables. From here, we look to *solve* these constraints using different methods.

Now, it is important to mention a couple of things. First, there are a lot of different methods for solving these types of constraint problems depending on certain characteristics of the equations (whether they are linear, nonlinear, convex, etc). Different methods will be better suited for different types of systems of constraints. However, here we will review *one* method that exists for solving these types of systems called Branch and Prune.

Branch and Prune

The benefits of Branch and Prune is that it's a method that works on all types of systems of continuous constraints and it comes with some guarantees:

- Correctness - It never produces a wrong solution.
- Completeness - It locates *all* solutions to systems of constraints (there can be more than one solution if the system is nonlinear).

Those are pretty serious guarantees right? In order to understand how those guarantees are being made, we will have to cover interval computations. Don't worry; the "how" isn't as important right now so we will put that in Appendix B, but the what and the why are important so we will cover those in as plain English as possible.

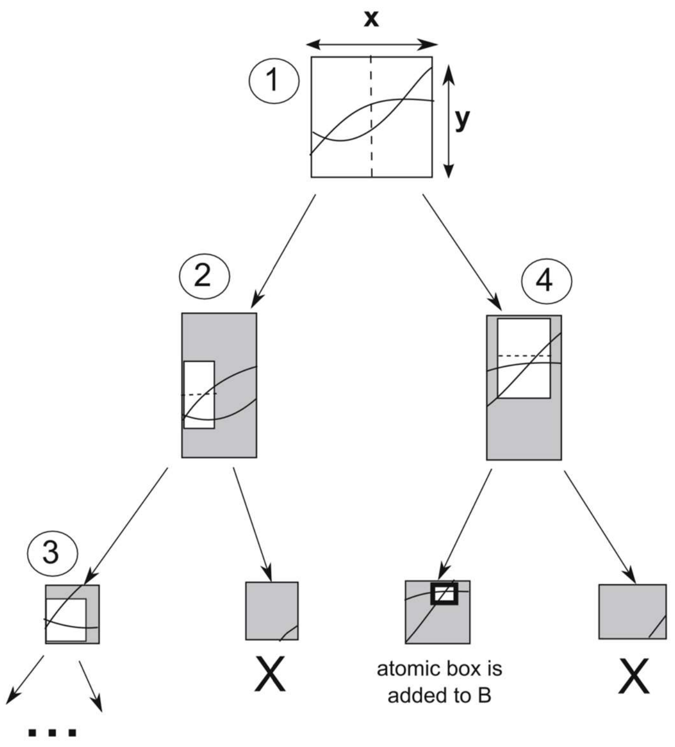
Interval Computations

Intervals are a representation of numbers. In order for it to make sense, we have to get used to the idea of talking about *intervals* of numbers instead of just numbers. For example, instead of saying 1, we can say the interval that contains 1; for example, [0, 2]. This brings up an interesting question: how wide should that interval be? Does [-1000, 1000] still qualify as the interval containing 1? It does but it's also the interval containing 10, 100, 1000, and every number in between. So it is not as insightful as, say, [0, 2].

We can do arithmetic with numbers, so it's important that we can do the same with intervals of numbers since we will be representing numbers using intervals. Interval arithmetic is possible and details on how to do interval arithmetic will be included in Appendix B. For now, all you need to keep in mind is that you can add, subtract, multiply, and divide intervals (amongst a lot of other things too).

Branch and Prune

So how does Branch and Prune provide these guarantees? Using interval computations. If we enclose all the answers for a system of constraints into a set of intervals known as a box (an interval X and an interval Y create a box around some graphs), then we already know the answers are within the box. But as it was mentioned earlier with using the interval [-1000, 1000] to represent the number 1; it is not as insightful. So Branch and Prune tries to *prune* the intervals that contains the answer as much as possible in order to make that interval as small as possible so that we can know the range of the answer, hopefully to such a small degree that it is useful for the application.



Branch and Prune

Image: Araya, I., & Reyes, V. (2016). Interval Branch-and-Bound algorithms for optimization and constraint satisfaction: a survey and prospects. *Journal of Global Optimization*, *65*(4), 837-866.

Above is an illustration of how Branch and Prune works. The box labeled 1 is the beginning point of the method. We can see two nonlinear constraints graphed and they meet at two places; these are the solutions to our constraint system. As mentioned earlier, the space where these two constraints exist is called a box. The goal of Branch and Prune is to split that box into smaller and smaller boxes until there are two boxes that are small and contain the answers within. The method is splitting the box into two; it is *branching* into two boxes continuously. As the procedure continues, it continues to split into smaller and smaller sub boxes until it finds its solution like in the boxes labeled 3 and 4.

Keep in mind, the width of the interval and its relationship to how useful the interval is depends on the application we are using it for. For example, if we are using intervals to compute an estimated landing zone of a spaceship, having the width of the interval be tens of feet apart, say [20 ft, 40 ft] wouldn't be a bad thing since that can be useful. On the other hand, if the interval is something like the amount of a chemical needed for the manufacturing of some pharmaceutical, it would be important to keep that interval tighter since we are dealing with milli or micro grams of difference.

What about Optimization?

So far, we have discussed solving constraints, both discrete and continuous. This means that, when faced with the problems of finding a solution or solutions to a system of discrete or continuous constraints, we have a broad understanding of how one would go about solving those systems. But what if we didn't want just *any* solution? What if we wanted the best solution as dictated by some standard or measure? This is known as optimization.

Optimization is basically what we have been discussing so far (satisfying/solving constraints) with an additional constraint: to maximize, or minimize the solution as dictated by some type of function.

Optimization Methods

As mentioned earlier, there are many different types of optimization methods for both discrete and continuous. We will be covering one; **Branch and Bound**. The Branch and Bound method is used for both discrete and continuous constraint problems.

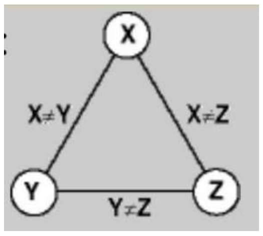
Branch and Bound with Discrete Constraints

Branch and Bound optimizes discrete constraints by trying a value for the first variable and satisfying its constraint. From there, it branches into the possible values of the second variable and attempting to satisfy its constraints; any values that don't satisfy the constraints at this point are removed from the search space entirely. The method then moves on to attempt the different remaining combinations. By the time it is complete finding *a* possible solution to the problem, it uses some function to calculate the bound (upper if attempting to minimize; lower if attempting to maximize). If this is the best bound according to what type of optimization problem we are attempting to do, this will be stored as the answer. Else, it will be replaced by a better solution that is found through the method.

Branch and Bound with Continuous Constraints

Branch and Bound with continuous constraints is very similar to Branch and Prune. As in Branch and Prune, the solutions are enclosed within a box (interval X and interval Y). From here, the search space is pruned based on the domain inconsistencies with the constraints. While doing this, potential interval solutions are evaluated and compared to the current best upper or lower bounds known; if these interval solutions are better than what is currently known, they replace the current upper or lower bounds as solutions. Else, the method continues searching for the optimal.

Appendix A – Formal Definitions of Consistency Techniques



Images: Barták, R. (1999, June). Constraint programming: In pursuit of the holy grail. In Proceedings of the Week of Doctoral Students (WDS99) (Vol. 4, pp. 555-564). Prague: MatFyzPress.

If the above is a CSP modeled as a graph, then the nodes are the variables we are using and the edges are binary constraints between variables.

Node Consistency - Remove values from domains of variables that are inconsistent with the constraints on those variables.

Arc Consistency - Remove values from domains that are inconsistent with the binary constraints between two variables.

Path Consistency - Removes values that fail to create a path between nodes.

Appendix B – Interval Computations

Adding Intervals:

If X = [a,b], and Y = [c,d], then:

X + Y = [a + c, b + d]

Example:

Let X = [0, 2] and Y = [-1, 1]. Then:

X + Y = [0 + (-1), 2 + 1] = [-1, 3]

Subtracting Intervals:

If X X = [a,b], and Y = [c,d], then:

X - Y = [a - d, b - c]

Example:

Let X = [-1, 0] and Y = [1, 2]. Then:

X - Y = [-1 - 2, 0 - 1] = [-3, -1]

Multiplying Intervals:

If X X = [a,b], and Y = [c,d], then:

X*Y = [min S, max S], where S = {a*c, a*d, b*c, b*d}

Example:

Let X = [-1, 0] and Y = [1, 2]. Then:

S = {-1*1, -1*2, 0*1, 0*2} = {-1, -2, 0, 0}

So: XY = [-2, 0]

Dividing Intervals:

If X = [a,b], and Y = [c,d], then:

X÷Y = X * (1/Y), where 1/Y = [1/$\underline{Y}$, 1/*Y*], assuming Y is not equal to 0

(Reminder: X*Y = [min S, max S], where S = {a*c, a*d, b*c, b*d})

Example:

Let X = [1, 3] and Y = [2, 4]. Then:

1/Y = [1/2, 1/4]

S = {1*1/2, 1*1/4, 3*1/2, 3*1/4} = {1/2, 1/4, 3/2, 3/4}

So: X÷Y = [1/4, 3/2]