BES502811

Auto Generating and Optimizing Modular Air Duct Layouts in Revit - BES502811

Daniel Kaye
Design Technologist | EvolveLAB

Ben Guler
Partner + CTO | EvolveLAB

Learning Objectives

- Experience how Generative Design can be used for mechanical layouts within Revit.
- See how incorporating DfMA workflows into Revit can streamline the design process.
- Understand the benefits of utilizing co-authoring and UI/UX design to promote user adoption.
- Learn about the real-time potentials of the Revit MEP API.

Description

What happens when some enthusiastic software developers team up with a few innovative mechanical designers?

Henderson Engineers is an MEP design firm that's been working on developing a modular DfMA (Design for Manufacture and Assembly) air duct system for a while now, and has created some clever Revit families and Dynamo scripts to help its designers model them. The firm joined forces with EvolveLAB, a Building Information Model (BIM) consulting and software development company, to create a Revit add-in that focuses on the real-time generative design of these modular duct layouts.

In this session, we'll review how these companies used some unique properties of the Revit API to create a co-authoring system that allows a mechanical design to sketch out a main duct layout in Revit while the computer simultaneously and continually produces the most optimal layout using the DfMA kit of parts for that sketch. Once the design is finalized, it is converted into a fully connected Revit duct system.

Speakers

Daniel Kaye - Design Technologist | EvolveLAB

Daniel Kaye is an architect, computer nerd, and avid rock climber hailing from snowy Buffalo, New York. He holds a B. Arch degree from the Syracuse University School of Architecture, as well as a minor in computer science. As a licensed architect he has had the opportunity to work at a variety of firms ranging from a 2-man operation with his former professor to the third largest AEC corporation in the world.

During Daniel's six years walking the line between architect and BIM Manager on residential, commercial, rail, and aviation projects across the country he became attuned to the inefficiencies inherent within the AEC industry. Combining his knowledge of our digital design tools with his recreational programming pursuits, Daniel joined EvolveLAB to help improve the software used to create the built environment in which we live.

Having been raised entirely on the East coast of the United States, Daniel cycled from Boston to Los Angeles before falling in love with the Rocky Mountains of Colorado, where he now resides. When not writing code or rock climbing, he can often be found outside sketching or jamming with friends on his drum kit.

Ben Guler - Partner + CTO | EvolveLAB

As a design technology fanatic, Ben has been a vital driver for computational design, process management, and standardization. With a range of technological avenues at his disposal, he has successfully identified and executed appropriate solutions for a robust set of project deliveries.

His architectural background, knowledge of BIM platforms and software engineering allows for a technological bridge that is critical to being effective in the computational design paradigm. As a husband and father, Ben enjoys going on hikes with his wife and daughter and occasional archery range sessions.

With Ben's experience in C# and Python, he has experience writing custom Revit Add-ins and stand-alone software to help tie the AEC market together. Ben has extensive experience using Dynamo to automate task and create solutions to complex design challenges. He also leverages Dynamo often to storyboard out his custom Revit add-ins.

0 – Introduction & Table of Contents

This paper is a case study of an application built as a collaboration between EvolveLAB and Henderson Engineers. It is broken down into the six (6) sections outline below:

1 – The Team

The people and companies behind the tool.

While there is a bio about Ben and myself at the start of this handout, lets dive a little into the companies that came together to make this happen

EvolveLAB

EvolveLAB, where both Ben and myself work, is a diverse collection of Licensed Architects, App Developers, and BIM Managers, and more.

EvolveLAB started in 2015 as an on demand computational design and BIM consulting firm. We work on everything from training and BIM implementation strategies to computational design software products and custom add-ins. They are the developers of Glyph (an auto-documentation tool for Revit), Helix (an interoperability tool for Revit, Sketchup, AutoCAD, and other AEC softwares), and various other BIM products and services.

Henderson Engineers

Henderson Engineers (partner company of Henderson Building Solutions) is a multi-disciplinary engineering firm. They "provide a wide array of building system design engineering services across a variety of building types."

For this project in particular, we worked with closely with the following team members:

- Adam Roth | BIM/VDC Director, Associate
- Dustin Schafer | CTO
- Nick Boyts | Senior BIM/VDC Application Specialist
- VJ Qureshi | Director of Software Development

2 – The Problem

Why can't the contractor just work it out?

Modularized Ducts

Before reaching out to us, Henderson Engineers had partnered with US Engineering Innovations to develop a modularized air duct system.

Sean Turner and Adam Roth actually shared a white paper on Autodesk University on the topic, entitled *DfMA for MEP with Generative Design and BIM Automation* in 2022, which you can see at the following link:

https://www.autodesk.com/autodesk-university/article/DfMA-MEP-Generative-Design-and-BIM-Automation-2022

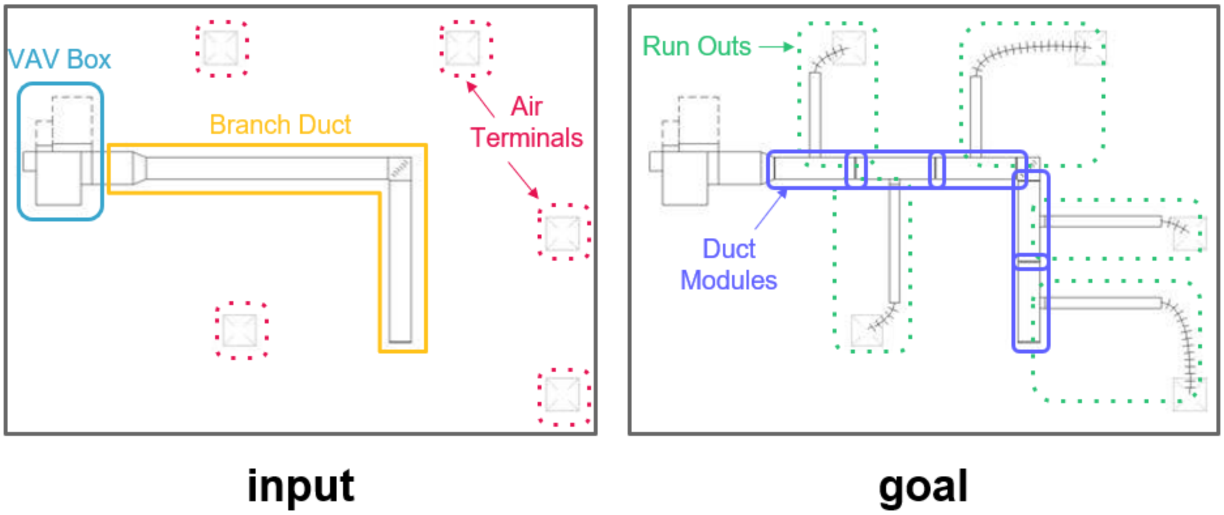
The question Henderson Engineers had for us was, essentially, how we can design using our newly developed DfMA kit-of-parts directly inside Revit?

A photo of some manufactured parts from the kit-of-parts modular air duct system Henderson had designed.

Image from "DfMA for MEP with Generative Design and BIM Automation" article, www.autodesk.com/autodesk-university/article/DfMA-MEP-Generative-Design-and-BIM-Automation-2022

Connecting VAV Boxes to Air Terminals

We started with a very simple and defined problem – how do we connect a Variable Air Volume (VAV) box to a set of pre-located Air Terminals, in Revit, using the module air duct system that Henderson has developed?



input

goal

Input: If you take a look at the above image, the starting condition is defined on the left. We have:

- A VAV Box, already placed in a specific location.
- 5 Air Terminals, already laid out to distribute air throughout the space.
- A main Branch Duct, presumably drawn by a Mechanical Designer / Engineer, that will take air from the VAV box and bring it closer to each Air Terminal.

Goal: Now take a look at the image on the right. You will notice that:

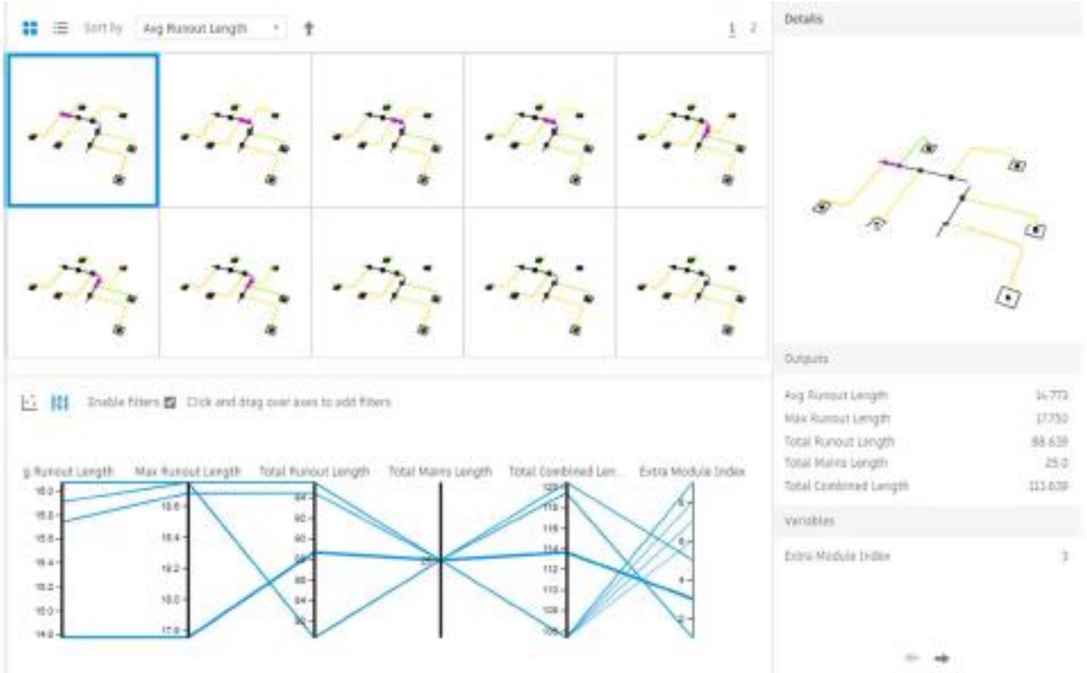
- The Branch Duct has been broken up into 5 'Duct Modules', which are unitized modular pieces that you can construct the branch duct out of.
- Each 'Duct Module' has been connected to one (and only one) Air Terminal, via a Run Out Duct (the straight part connects to the Duct Module) and a Flex Duct (the curve piece made of little lines that connects to the center of the Air Terminal).
- Note that both the 90-degree mitered corner piece, and fixed length Run Out Ducts, are also both created from pre-design modular pieces of Henderson's modular air duct system.

Prototyping with Dynamo & Revit GD

Before approaching EvolveLAB, Henderson Engineers had already:

- Created the parts of their modular system as Revit Families
- Made a proof-of-concept in Dynamo that could layout a basic system using these modules
- Using Revit's Generative Design tool to generate and optimize layouts

Dynamo, which comes with Revit, is a great prototyping tool for ideas like this. That, plus the fact that this system was already built and tested with Revit components, means that Henderson already knew what was possible to automate. In fact, the first version of our application almost literally translated and used their Dynamo script to generate modular duct networks.



Outputs

Avg Runout Length	14.773
Max Runout Length	17.750
Total Runout Length	88.639
Total Mains Length	25.0
Total Combined Length	113.639

Variables

Extra Module Index	3

Screenshot of the proof-of-concept – Henderson Engineers' Dynamo script being run through the Revit Generative Design tool to product and optimize modular duct layouts.

Image from "DfMA for MEP with Generative Design and BIM Automation" article, www.autodesk.com/autodesk-university/article/DfMA-MEP-Generative-Design-and-BIM-Automation-2022

As Ben likes to say, "Power Users design tools for other Power Users." The above workflow was an incredible proof-of-concept, but what Henderson Engineers really wanted was to get their entire team designing with modules – which is where we, EvolveLAB, really came into play. How can we take this proof-of-concept and create an intuitive user-friendly tool that works seamlessly within Revit so all Henderson Engineers' engineers could use this modular duct system?

3 – The Solution

The Mechanical Run-Out Duct Generator

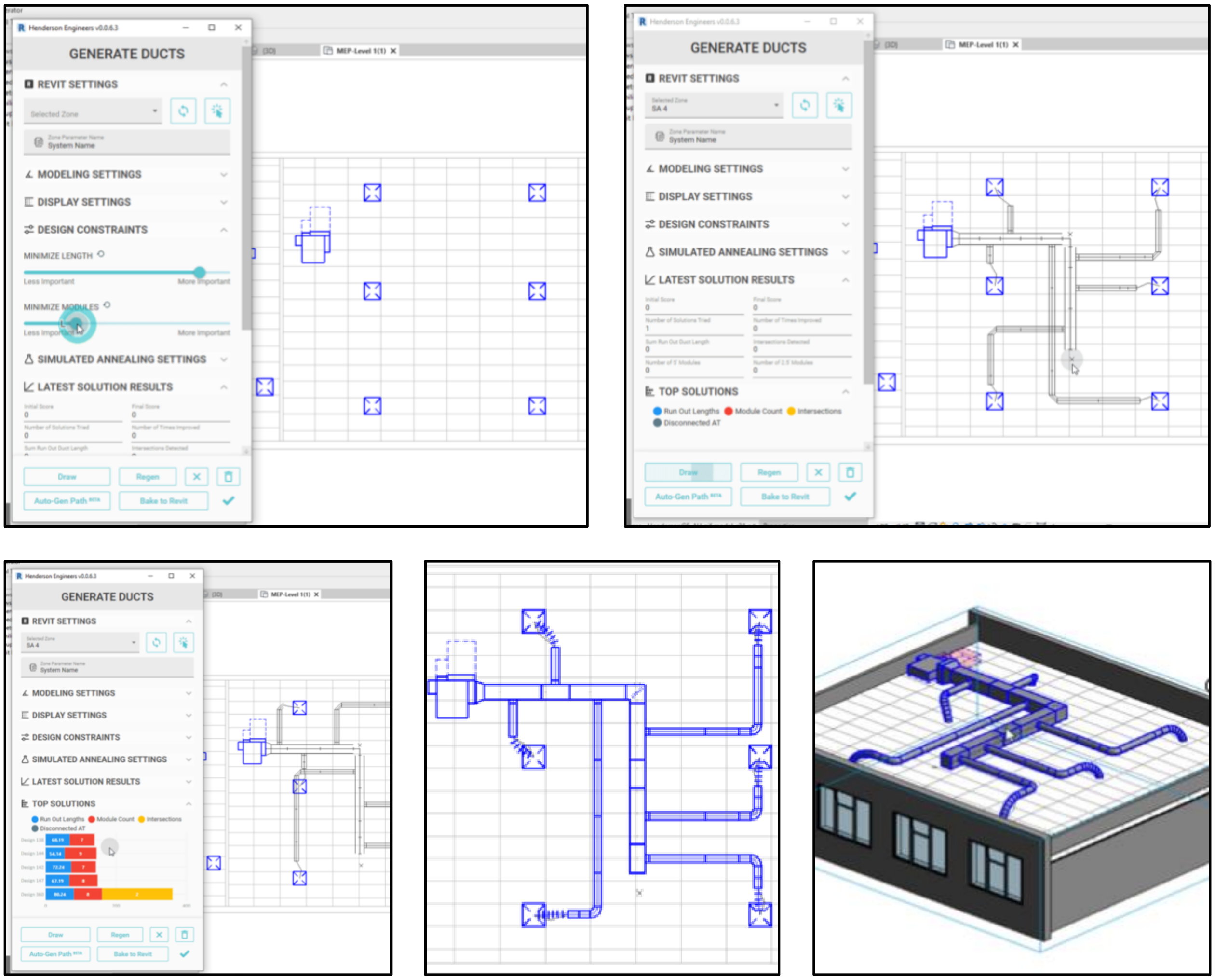
Since this is a case study, we are going to get right down to it and show you the application we are case studying, which we have dubbed the “Modular Duct Generator App.”

The Modular Duct Generator App

What is it?

Is it a Revit Add-In that generates and optimizes Run Out (and sometimes Branch) Duct layouts, using a specific set of DfMA modular parts.

It combines both human and computer / generative design input to produce a constructable duct system, and then when completed, ‘bakes’ the selected design into a fully connected mechanical system within Revit.



Screenshots from using the Modular Duct Generator App

The Modular Duct Generator App 101

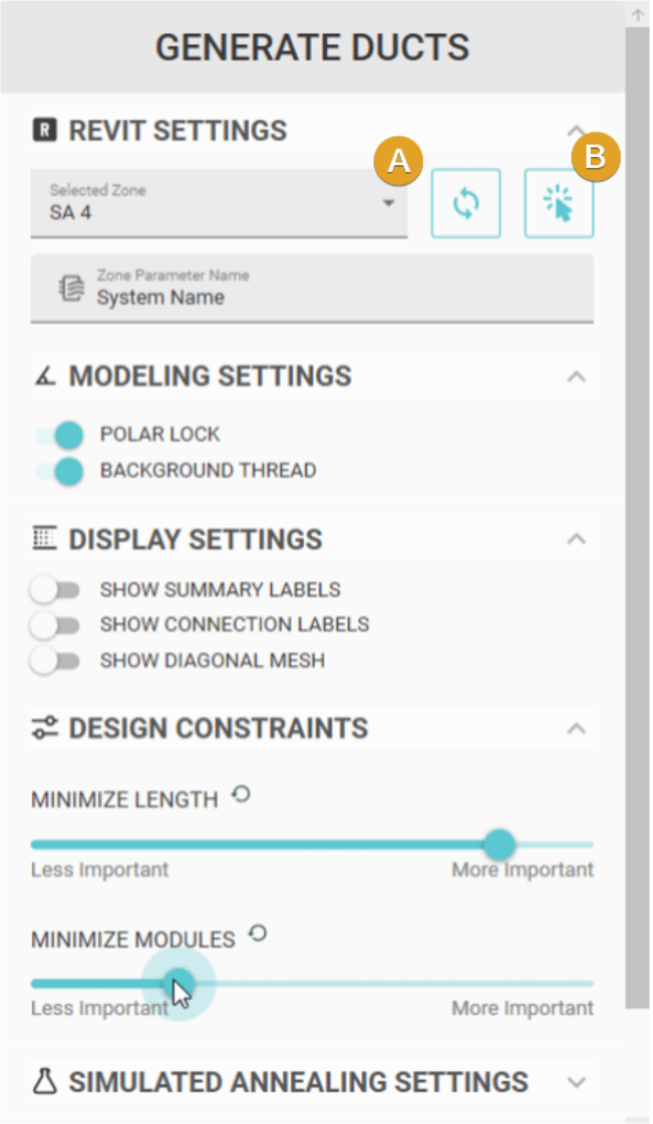
A simple 'how to' for using your custom mechanical generative design application.'

In case you have never used a Modular Duct Generator App before (which is likely, since this is the first ever that we know about), we have created a simple 6-step guide!

1. Select VAV Box & Options

The first step is to select a VAV Box, which automatically selects a mechanical zone associated with that box. You can do this either by selecting a zone from the dropdown or using the mouse button to select one directly from the Revit model.

Once a VAV Box is selected, you can adjust the many other settings shown in the screenshots, including:



GENERATE DUCTS

REVIT SETTINGS

Selected Zone: SA 4

Zone Parameter Name: System Name

MODELING SETTINGS

POLAR LOCK

BACKGROUND THREAD

DISPLAY SETTINGS

SHOW SUMMARY LABELS

SHOW CONNECTION LABELS

SHOW DIAGONAL MESH

DESIGN CONSTRAINTS

MINIMIZE LENGTH

Less Important — More Important

MINIMIZE MODULES

Less Important — More Important

SIMULATED ANNEALING SETTINGS

A screenshot of some of the settings from the Modular Duct Generator App.

Modeling Settings
Options controlling how the duct system gets modeled once you start drawing

Display Settings
Toggles various optional graphics on or off

Design Constraints
Chooses how important each of the design constraints will be in optimizing your duct network.

These could be the most important settings in this application. In this case, we are working with a **multi-objective optimization** problem, which means that we are trying to optimize multiple competing metrics, where improving one metric often makes others worse

For example, if we want to have less Duct Modules, it will make the Run Out Ducts lengths longer. Having the sliders allows the user to change the priorities for each design constrain, which in turn allows the algorithm to generate intelligently-optimized results.

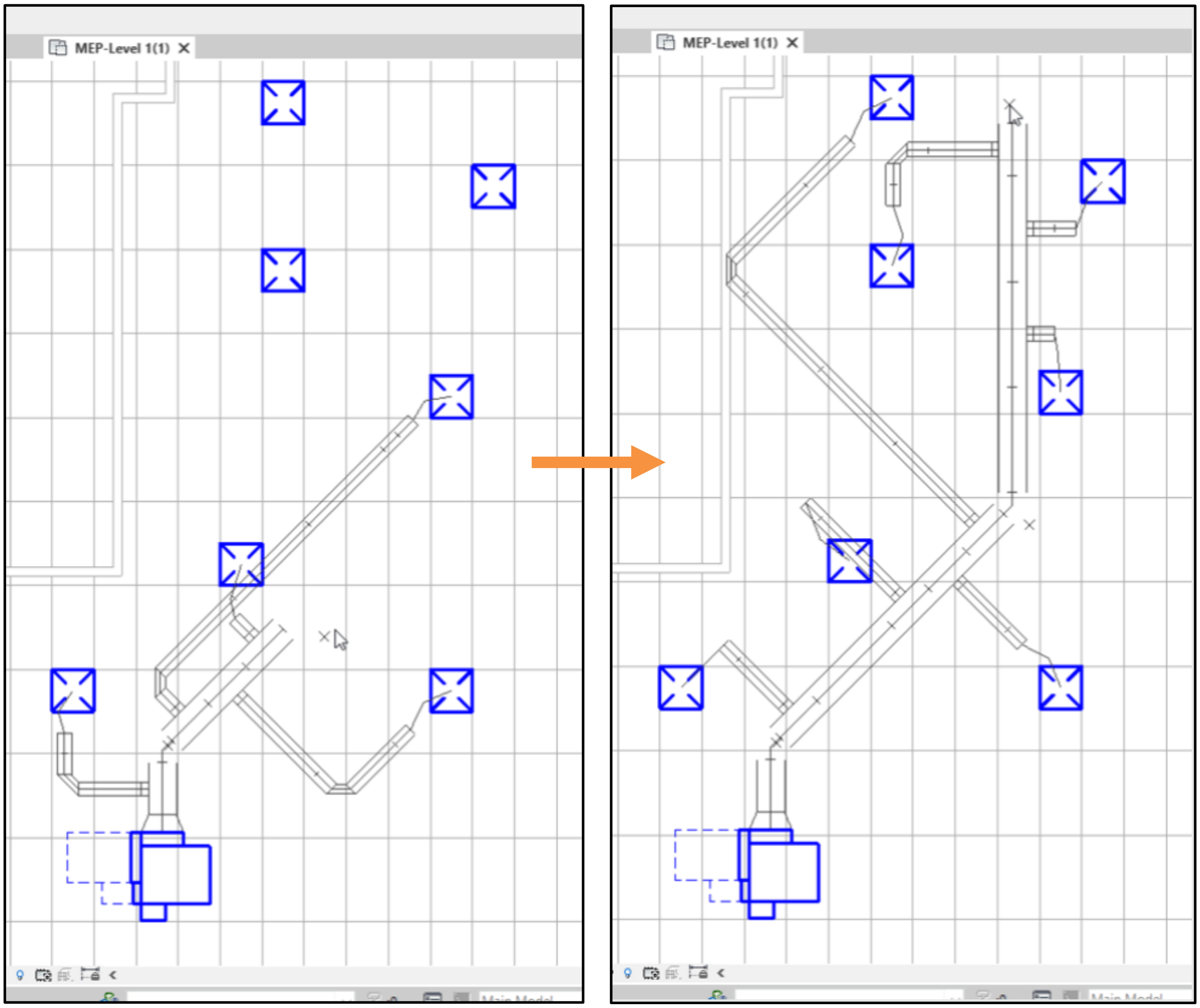
Simulated Annealing Settings
Controls how fast and in-depth the Generative Design algorithm runs. We'll get further into Simulated Annealing in the next section.

2. Draw Main Branch Duct

Once you have a VAV Box selected and your settings updated, it’s time to draw the main Branch Duct. To do so, simply click on the ‘Draw Duct’ button at the bottom of the app, and then proceed to click in your Revit view.

As you click, a modular Branch Duct will automatically be generated. It wills start from the previously selected VAV Box and connect all of your clicked points in order using the standard module sizes.

Noe that as you drag the mouse, the math is done automatically for you. You are only able to draw at 45-degree angles, and the mitered corner pieces and straight duct modules are automatically placed and sized to best-fit themselves to your mouse clicks.



Screenshots of drawing a main Branch Duct using the Modular Duct Generator App.

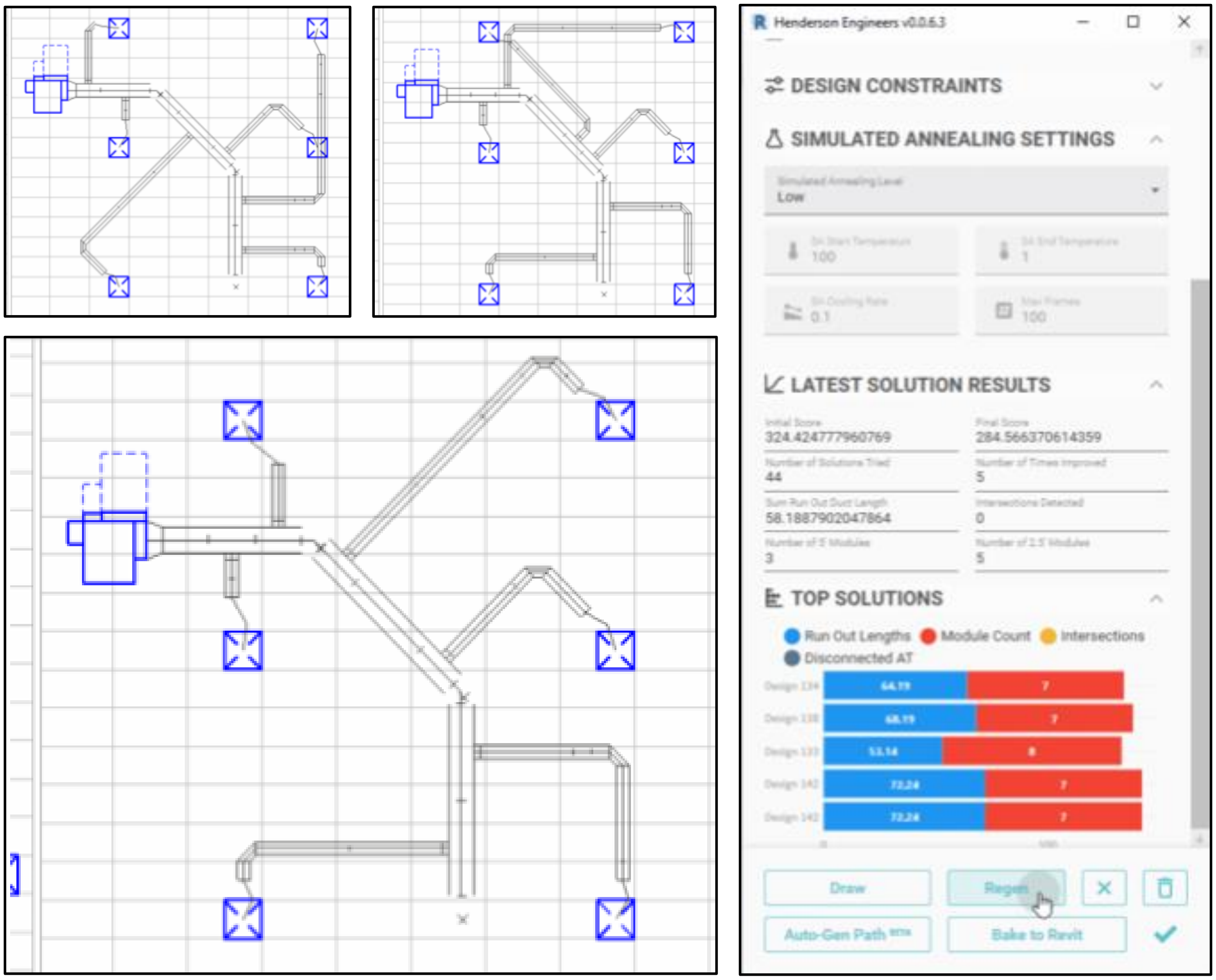
3. Generate Solutions

As soon as you start drawing a Branch Duct, possible solutions are generating. The longer your mouse stays still, the better the solutions will get. Although thousands of solutions are being generated in the background, only the best one that has been found so far will be drawn to the screen.

Once you have finished drawing your Branch Duct, the solutions will finish generating. This could take from half a second to 10 seconds, depending on your Simulated Annealing settings.

If you'd like to generate more solutions for the last Branch Duct you have drawn, you can simply click the 'Regen' button in the bottom action bar of the app, and a new set of solutions will be generated.

Note that for a given drawing, no matter how many times you regenerate solutions, the top solutions from all attempts will be shown (not just the best from your previous regeneration). This is useful because you can either get a quick solution or adjust the settings and regenerate as many times as needed to improve the current solution until you have an optimal one you like.



Simulated Annealing Level: Low

SA Start Temperature: 100

SA End Temperature: 1

SA Cooling Rate: 0.1

Max Frames: 100

Initial Score: 324.424777960769

Final Score: 284.566370614359

Number of Solutions Tried: 44

Number of Times Improved: 5

Sum Run Out Duct Length: 58.1887902047864

Intersections Detected: 0

Number of 5' Modules: 3

Number of 2.5' Modules: 5

Screenshots of generating duct layout solutions using the Modular Duct Generator App.

4. Examine Solutions

Once solutions have been generated, you can use the Top Solutions section of the app to compare different solutions.

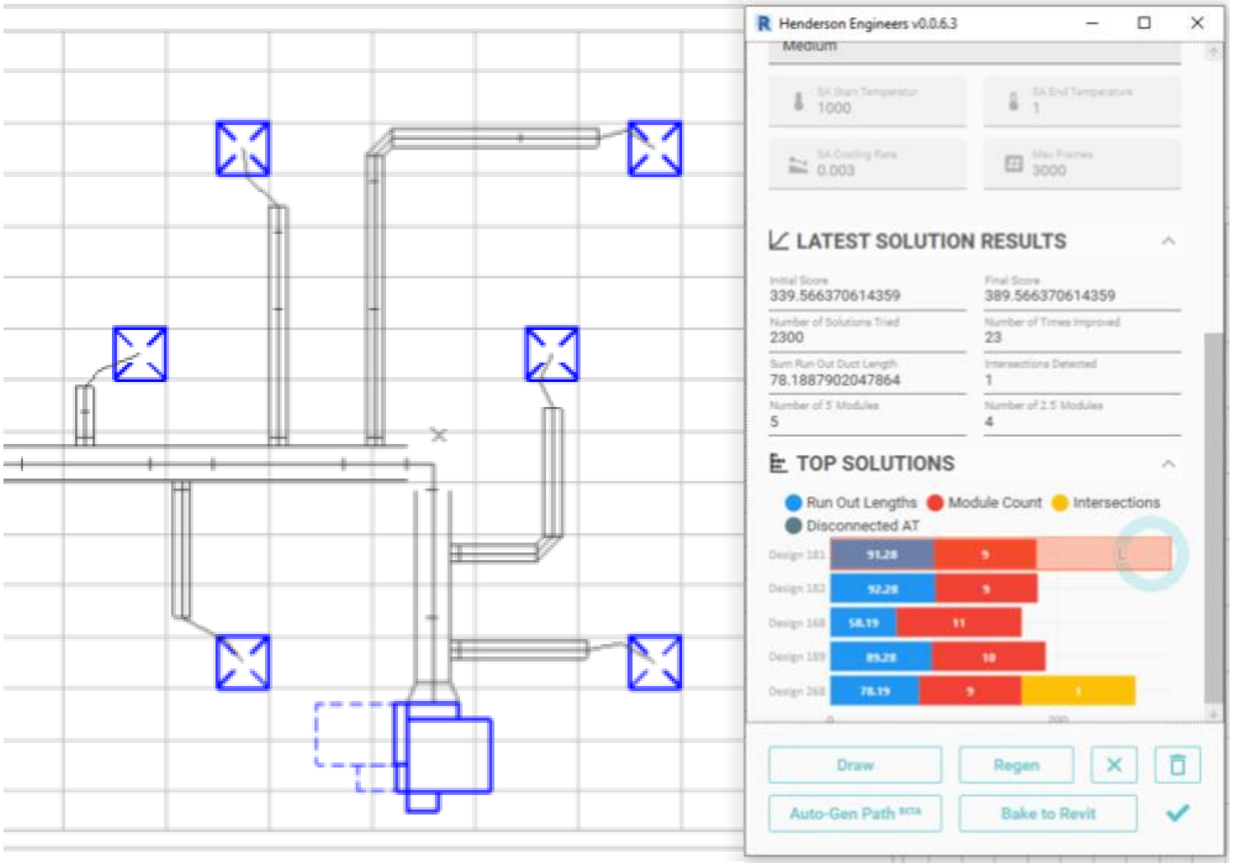
If you hover or click on a particular colored bar, the corresponding solutions will be shown in the Revit view.

Each color represents a different metric of that design. The number inside the bar graph is the actual value of the metric – for example, the number in the blue ‘Run Out Lengths’ section is the total length, in feet, of all Run Out Ducts in that solution.

The width of the bar, however, corresponds to the weight, or importance, of each metric. For example, in the solution that has a large yellow bar, the ‘1’ indicates that that solution has one detected intersection. Even though it is only 1, the penalty for intersections is quite high, since they are usually not constructable, so the width of the yellow bar makes up over 1/3 of the score for that solution. In this case, a lower score, AKA shorter bar, is better.

If you hover over any bar for a few seconds, even more detailed metrics about that solution will be shown (not pictured, see gif in presentation slides).

After analyzing your solutions, you can redraw the branch duct, or adjust relevant settings, and then regenerate more solutions, until you find one you are happy with.



Henderson Engineers v0.0.6.3

Medium

SA Start Temperatur: 1000
SA End Temperature: 1
SA Cooling Rate: 0.003
Max Frames: 3000

LATEST SOLUTION RESULTS

Initial Score: 339.566370614359
Final Score: 389.566370614359
Number of Solutions Tried: 2300
Number of Times Improved: 23
Sum Run Out Duct Length: 78.1887902047864
Intersections Detected: 1
Number of 5' Modules: 5
Number of 2.5' Modules: 4

TOP SOLUTIONS

Screenshots of examining multiple duct layout solutions using the Modular Duct Generator App.

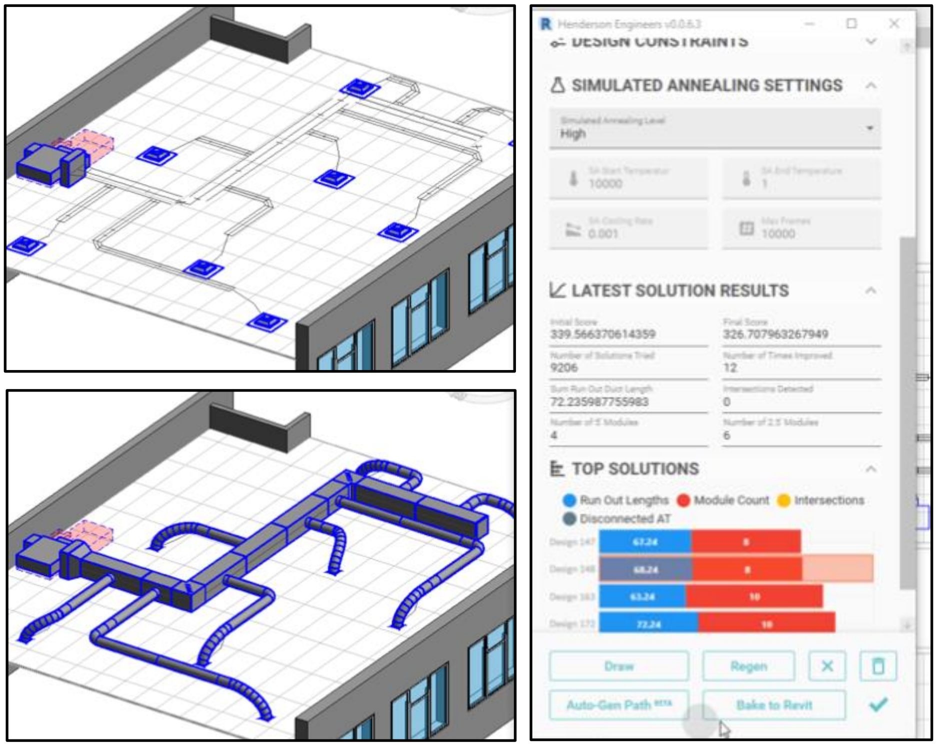
5. Bake to Revit

Once you've settled on a solution, simply click on the 'Bake to Revit' button in the action bar at the bottom of the app, and your design will be created as a fully-connected mechanical duct system in Revit, using a mix of native Revit duct families and the pre-built modular duct families Henderson Engineers has created. Even though we have only been looking at the tool in plan view so far, everything it's been doing and generating has been happening in 3D.

I'd like to give a small shoutout to the Revit MEP API, which is incredibly powerful. It was added after the original release of the Revit's public API and affords us a lot of flexibility to take advantage of the agnostic 'connectors' present in all mechanical, electrical, and plumbing families to manipulate systems and create properly connected parts! If you are interested in learning more about the Revit MEP API, I suggest checking out The Building Coder (AKA Jeremy Tammik) blog post on it here:

https://thebuildingcoder.typepad.com/blog/2009/09/the-revit-mep-api.html

The ability to 'Bake to Revit' allows your models to be a bit less sacred. What I mean by that is – without a tool like this, if you had to modify or redo part of your mechanical model, you may encounter resistance due to the time and effort it took to originally make the model, as well as even the thought of having to redo a lot of work to re-make it. With tools like this, modeling complex systems is reduced to the click of a button – so making changes (perhaps later than you would like in the design schedule) is much easier – you can simply regenerate, and bake a new one. This also allows you to quickly model and test more options in the earlier schematic design phases, without worrying about wasting effort on creating detailed systems too early in the design process.



Simulated Annealing Level: High

SA Start Temperatur: 10000

SA End Temperature: 1

SA Cooling Rate: 0.001

Max Frames: 10000

Initial Score: 339.566370614359

Final Score: 326.707963267949

Number of Solutions Tried: 9206

Number of Times Improved: 12

Sum Run Out Duct Length: 72.235987755983

Intersections Detected: 0

Number of 5 Modules: 4

Number of 2.5 Modules: 6

Before (top left) and after (bottom left) baking a duct layout solution to Revit using the Modular Duct Generator App. Note the 'Back to Revit' bottom in the bottom right of the App.

6. Auto-Gen Branch Duct

This final (optional) step is to let the app draw the Branch Duct for you, instead of manually drawing it as we did in step 2.

This feature is still in beta, but I think it's important to mention because it allows you to imagine the possibilities of where this, or similar tools, could go with more time.

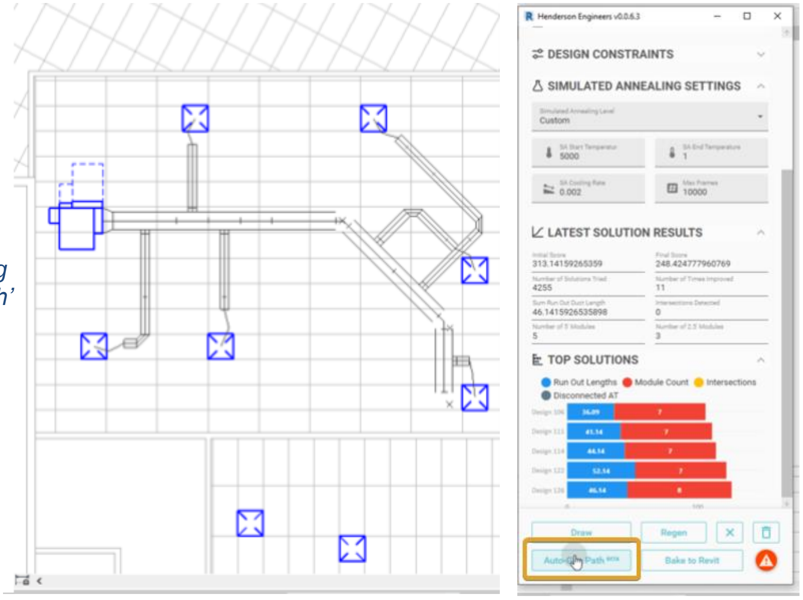
Starting with a well-defined problem like we did here, allows us to quickly produce a tool that can be tested and used in production right away. It provides instant user feedback, and a time-saving return-on-investment for Henderson Engineers. This is all part of the Agile development process, commonly used by software development teams. The basic idea of Agile is to 'deliver value early.' If you are unfamiliar with the term, I think a Dave Griffiths' YouTube videos provide an excellent explanation:

https://www.youtube.com/c/DavidGriffithsEsq/videos

As this app grows , we could expand the tool to start slowly optimizing other tasks. We already started on this Auto-Gen Branch Duct feature, but in the future, we could see this tool also diving into:

- Auto-Generating Duct Layouts for multiple VAV Boxes and zones at once
- Layout out Air Terminals
- Optimizing VAV Box placement
- Creating mechanical zones throughout the building

We will dive more into this in the final section '6 – Beyond Ducts', but in my mind, the possibilities are really endless.



Simulated Annealing Level
Custom

SA Start Temperatur
5000

SA End Temperature
1

SA Cooling Rate
0.002

Max Frames
10000

Initial Score
313.14159265359

Final Score
248.424777960769

Number of Solutions Tried
4255

Number of Times Improved
11

Sum Run Out Duct Length
46.1415926535898

Intersections Detected
0

Number of 5 Modules
5

Number of 2.5 Modules
3

An example of a Branch Duct path generated by using the 'Auto-Gen Path' beta tool, which is located in the bottom-left of the app's UI.

4 – Simulated Annealing

The brains behind the ~~brawn~~ BIM

Now that we have a basic understanding of what the tool does, let's dive into how and why it works – How does it find the 'best' optimized solution from all the potential possibilities?

Fair warning – this section gets a bit more technical. In my opinion, it also gets a bit more interesting too. Let us dive in!

Generative Design

What is Generative Design?

This is one of the most popular buzz words in the AEC industry right now, and there are a wide range of interpretations of what it means. Autodesk provides a few definitions which I like, quoted below, but for our purposes, I would simply describe it as "an iterative process from a computer's point of view."

Here's those Autodesk quotes:

> *"Quickly generate high-performing design alternatives—many that you'd never think of on your own—from a single idea. With generative design, there is no single solution; instead, there are multiple great solutions. You choose the design that best fits your needs." -Autodesk*

> *"Generative design is a design exploration process. Designers or engineers input design goals into the generative design software, along with parameters such as performance or spatial requirements, materials, manufacturing methods, and cost constraints. The software explores all the possible permutations of a solution, quickly generating design alternatives. It tests and learns from each iteration what works and what doesn't." -Autodesk*

Both quotes are from this page on Autodesk's website:

https://www.autodesk.com/solutions/generative-design

The Generative Design Process

The ‘Manual’ Way

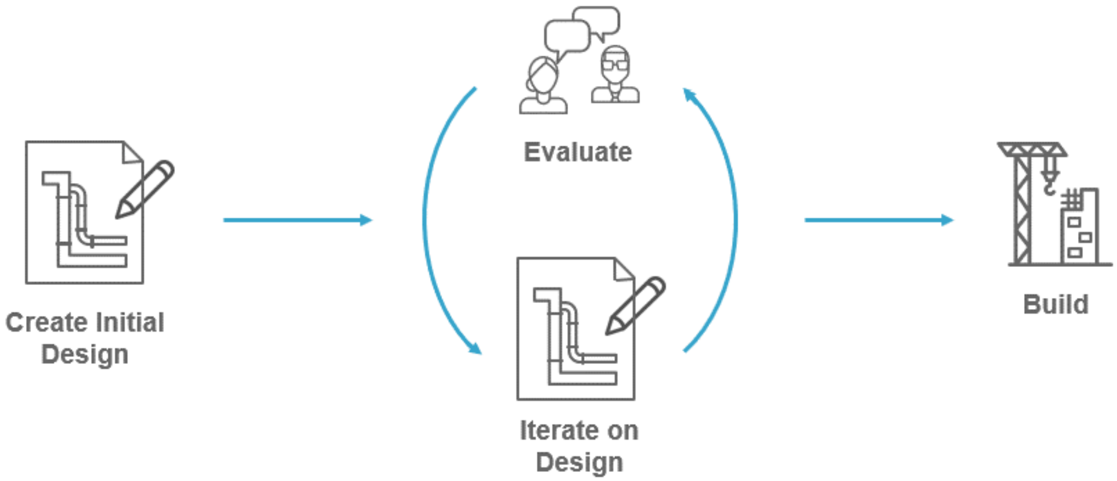


Diagram of the ‘manual’ design process for generating designs.

An easy way to understand the generative design process is by first looking at how we traditionally create designs:

- First, we sketch out an initial idea
- Next, either individually or as a group, we look over our design and suggest edits and improvements.
- This cycle continues, until the design is complete and correct enough to construct

Of course, in real life the process is not always as simple as the diagram describes, but it will help us understand how this compares to generative design as performed by a computer.

The 'Computational Design' Way

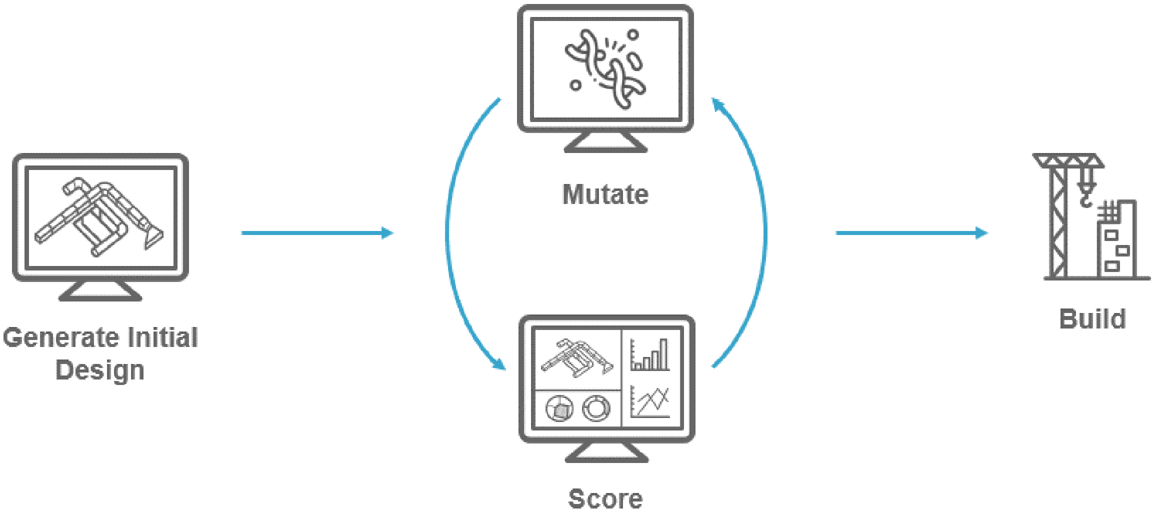


Diagram of the 'Computational Design' process for generating designs.

Generating designs with a computer can follow a similar pattern – all you have to do is place your diagram icons inside of monitor icons!

That's a joke of course, but the basic premise is the same:

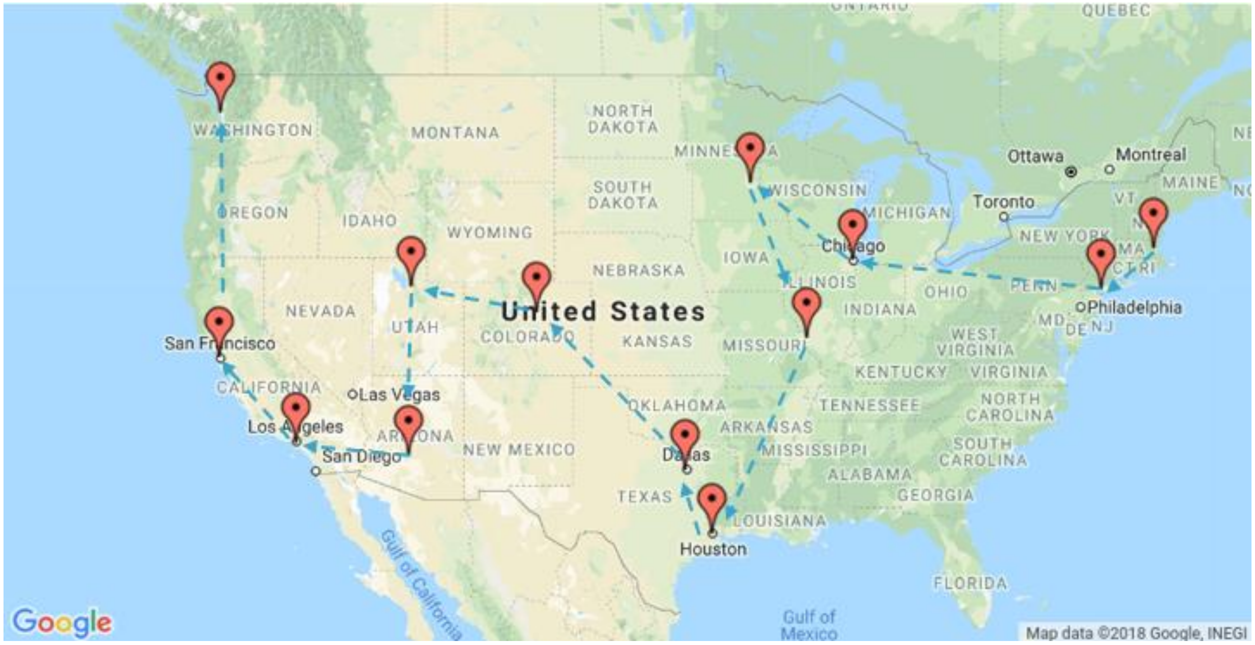
- First, the computer generates an initial design.
- Next, the computer assigns that design a numeric score. This is somewhat analogous with the 'Evaluate' stage from the manual process diagram above.
- Then, the computer 'mutates' the design, which means it makes a new slightly altered design.
- This cycle continues, until the design is complete and correct enough to construct

At this point you may have wondered why, with today's computers, we can't simply just generate all possible designs – instead of the generate-score-mutate loop – and then just pick the best one.

Let's take a step into the computer science world for a bit, to understand why that just isn't feasible.

The Traveling Salesperson Problem

A yet unsolved problem



A Google Map image of the USA that illustrates the Traveling Salesperson Problem by showing 13 major cities, and arrows depicting one potential path to connect them all.

To explain why you cannot simply try all possible solutions, we are going to look at the Traveling Salesperson Problem.

The basic idea is that you have a list of cities, and a traveling salesperson needs to visit them all, in any order. The problem is finding the shortest possible route that connects all the cities.

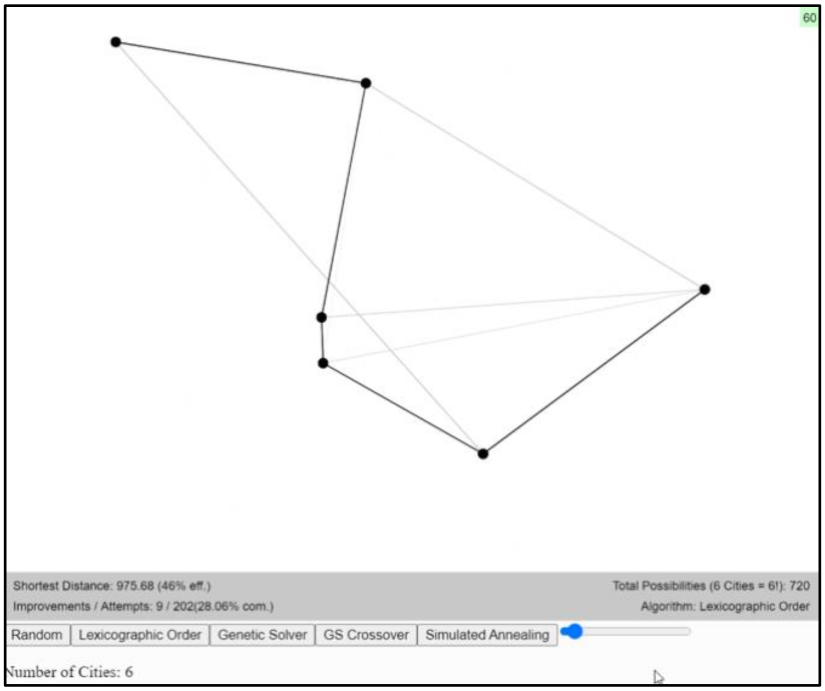
An analogous example would be to think of an Amazon Prime delivery truck route. The truck needs to deliver a certain number of packages to a specific list of addresses. What would be the fastest way for the drive to get to each address?

In the field of computer science, believe it or not, this problem has not been definitively solved yet.

The number of possibilities is equal to the factorial of the number of cities. For example:

- With 6 cities…
  - There are 720 possibilities (6!)
  - 12 seconds to solve*
- With 8 cities…
  - There are 40,320 possibilities (8!)
  - ~11 minutes to solve*
- With 50 cities…
  - 30.4 vigintillion possibilities (50!)
    - 30,400,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000 possibilities (50!)
    - FYI: 1 vigintillion = 10^{63}
  - ~ 16,055,627,510,117,791,688,711,465,754,571,818,807,680,369,975,739,946 years to solve*

**This solver tries 60 solutions per second. It is, of course, possible to generate solutions much faster, this speed was chosen to create meaningful and understandable animations. This number is the amount of time it would take, at 60 tries/second, to try all possible solutions for the given number of cities.*



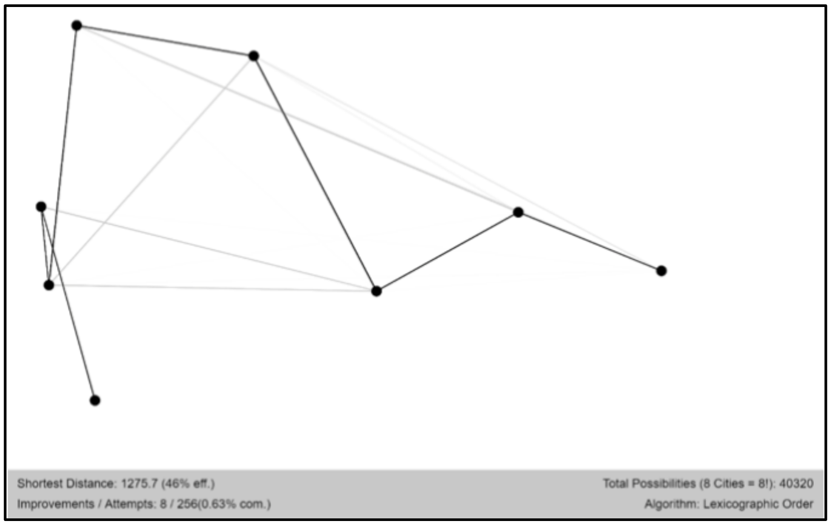
Shortest Distance: 975.68 (46% eff.)
Improvements / Attempts: 9 / 202(28.06% com.)
Total Possibilities (6 Cities = 6!): 720
Algorithm: Lexicographic Order

Number of Cities: 6

The EvolveLAB TSP Solver, working on a TSP problem with 6 cities, using a 'Lexicographic Order' algorithm

Look at the exponential jump in the number of possible paths just by increasing the city count by 2 (from 6 to 8)! Even with modern supercomputers, we do not have the time to solve for this problem with 50 cities.

While that may seem like a lot of cities, imagine the design of a building – there are typically way more than 50 different variables, and each variable can often have an almost infinite number of possibilities. For example, a single duct can be placed in many different 3D locations and orientations within the ceiling plenum – in the TSP with 50 cities, each city only has 49 other finite options of other cities it can connect to.



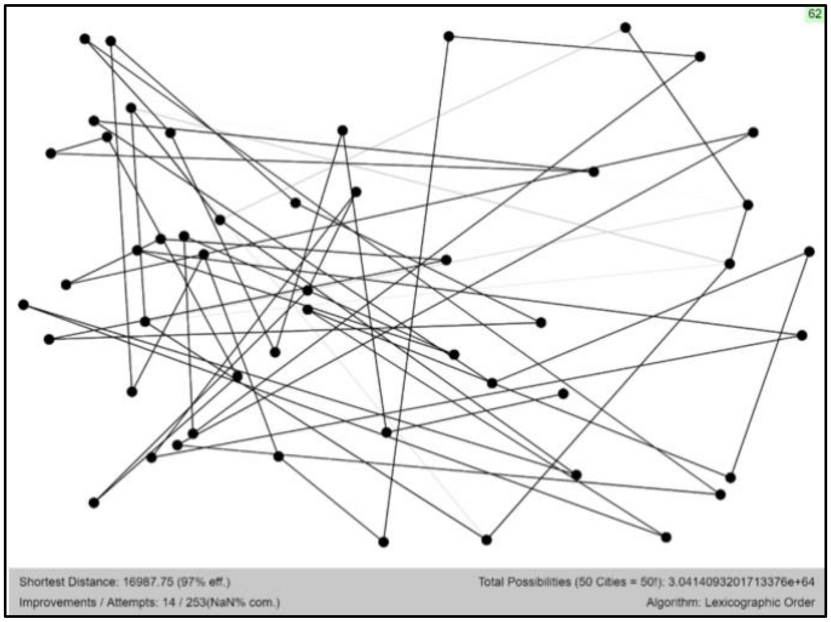
Shortest Distance: 1275.7 (46% eff.)
Improvements / Attempts: 8 / 256(0.63% com.)

Total Possibilities (8 Cities = 8!): 40320
Algorithm: Lexicographic Order

The EvolveLAB TSP Solver, working on a TSP problem with 8 cities, using a 'Lexicographic Order' algorithm

Check out the screenshots of our TSP solver spread across these two pages. The point here is, even with today's computing power, we are nowhere close to being able to brute force this problem, let alone the design of a mechanical system.

So the question becomes, how do we solve problems with such a huge possible solution space?



Shortest Distance: 16987.75 (97% eff.)
Improvements / Attempts: 14 / 253(NaN% com.)

Total Possibilities (50 Cities = 50!): 3.0414093201713376e+64
Algorithm: Lexicographic Order

The EvolveLAB TSP Solver, working on a TSP problem with 50 cities, using a 'Lexicographic Order' algorithm

The above screenshots are from a Traveling Salesperson Problem (TSP) solver developed by EvolveLAB to test out, compare, and explain various algorithms. This TSP solver animates the entire process, so actually watching it run can makes it easier to understand what's going on. You can find animated GIFs in the PowerPoint file for this class, or you can even play with the interactive open-source solver at the link below:

https://editor.p5js.org/danno484/full/uWgbvmUlu

Metaheuristics

Educated Guessing

Metaheuristics is certainly a big word, and I feel smart on those rare occasions when I get to use it in conversation. If we break it down into two parts, it's a bit easier to digest:

"meta" – "heuristics"

Let us start with "**heuristics**." Basically, a heuristic is an educated guess. When we have a problem for which we do not know the answer, and to figure out the exact right answer is either too complicated, too time consuming, or impossible. Instead of trying to solve it, we take our best guess. As Wikipedia currently states:

"Examples that employ heuristics include using trial and error, a rule of thumb or an educated guess."

Heuristics is a field itself in many different areas, including Psychology, Anthropology, Law, Philosophy, and more.

Meta (no, not the company that owns Facebook) is a Greek prefix meaning 'big' or 'overall.' It has even come back into light in recent years as a slang word, defined by Urban Dictionary as:

"...seeing the thing from a higher perspective instead of from within the thing."

When you string them together, **metaheuristics** is basically making large overall educated guesses in attempt to find a good-enough solution to a complex problem in a short amount of time.

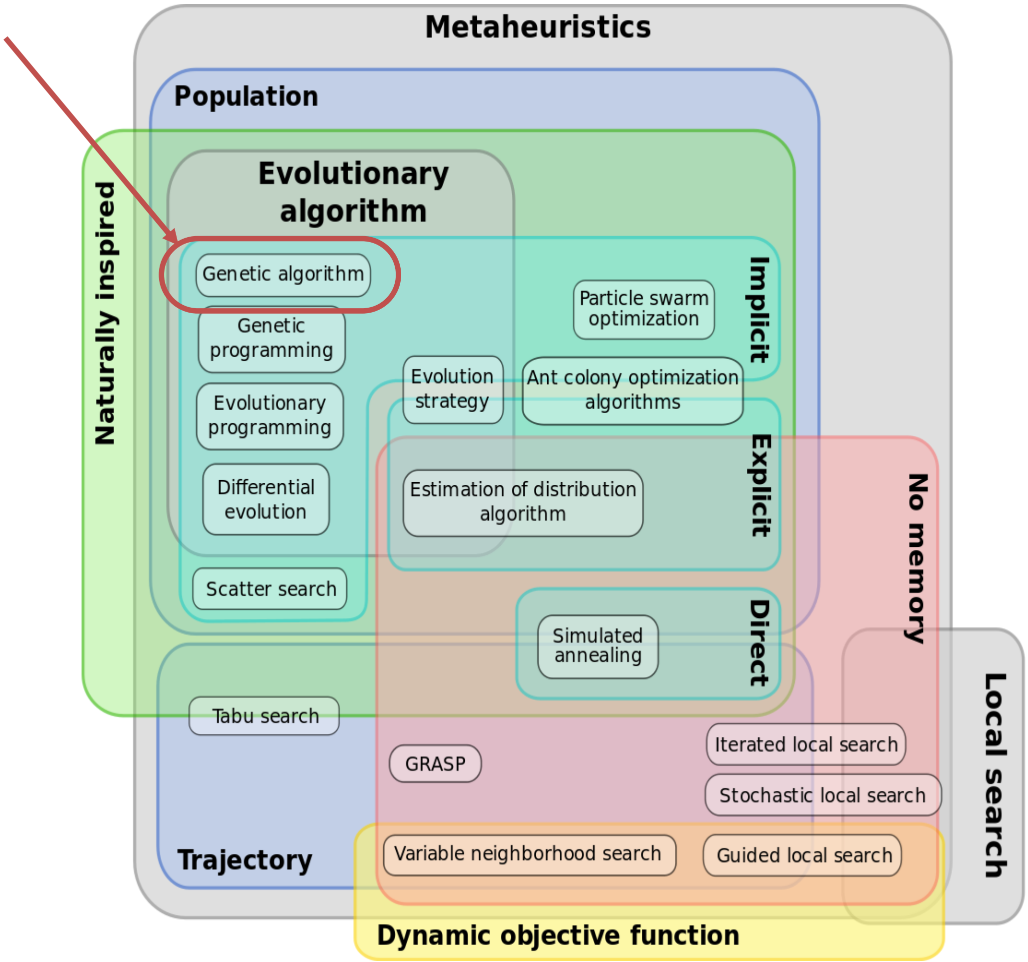
Metaheuristic Algorithms

A computer's best guess

A metaheuristic algorithm is just an algorithm that tries to solve a problem using a metaheuristic approach. There are tons of them out there. The chart below by Johann "nojhan" Dréo attempts to diagram and organize some of the more popular ones.'

Each has its own pros, cons, and use cases. Many are also nature inspired, including the two we are going to look at today.

First up, we'll look at a Genetic algorithm (circled below), as this is easy to understand, and is what Revit's Generative Design tool uses (specifically, it uses the non-dominated sorting genetic algorithm II, or NSGA-II)



"Different classifications of metaheuristics shown as a Euler Diagram" by Johann "nojhan" Dréo, 28 August 2011 , with the genetic algorithm circledhttps://commons.wikimedia.org/wiki/File:Metaheuristics_classification.svg

Genetic Algorithm

It made us, so that's something right?

Genetic algorithms are inspired by Darwin's theory of evolution. Unfortunately, they can also be slow – like the evolutionary process.

Essentially, the algorithm will take a 'parent' (or parents) and use it/them to generate a specific number of 'offspring.' The offspring are simply slight mutations (remember that word from the Diagram of the 'Computational Design' process earlier?), together referred to as a generation. Then, each offspring will be scored, and the best offspring will be used as the parents for the next generation. It is like selective breeding!

In the diagram below, you can see the process at work – starting from two average folk, and in this case trying to produce some construction workers with really cool looking eye protection:

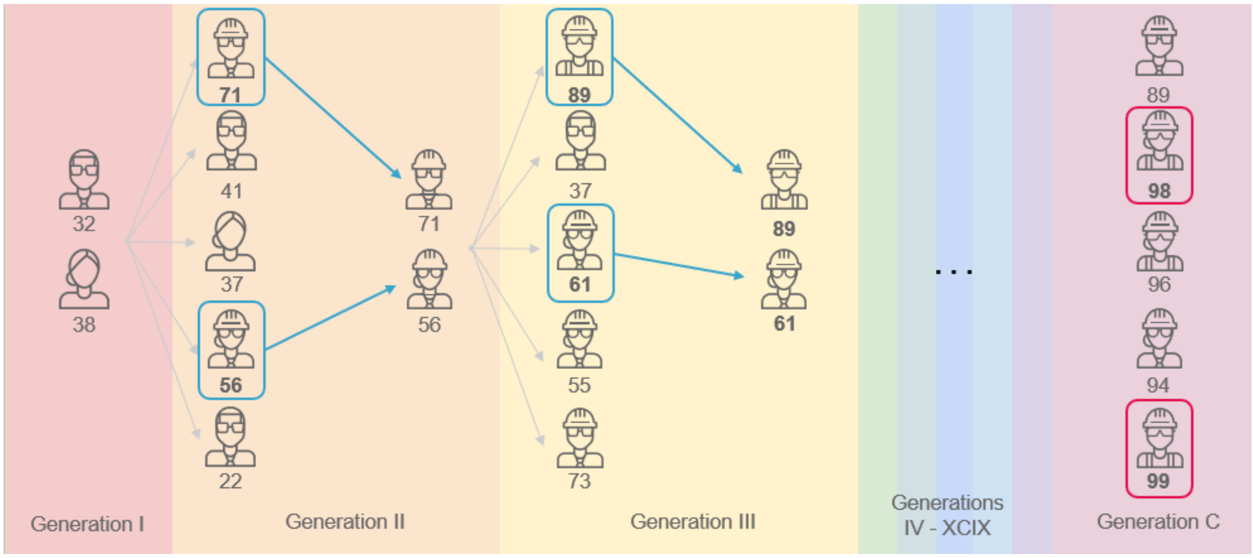
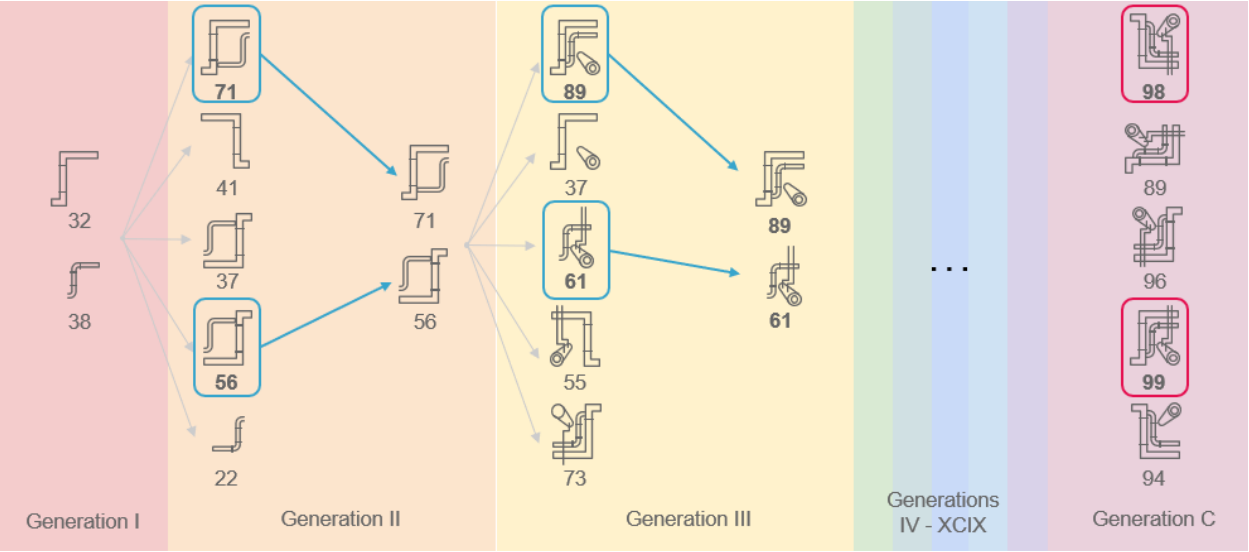


Diagram of a 'Genetic Algorithm' going through 100 (C) generations in attempt to produce a construction worker with really cool sunglasses. As a side note, this (and most diagrams produced in this class) were made exclusively from Icons provided by Autodesk, of which I am quite proud!

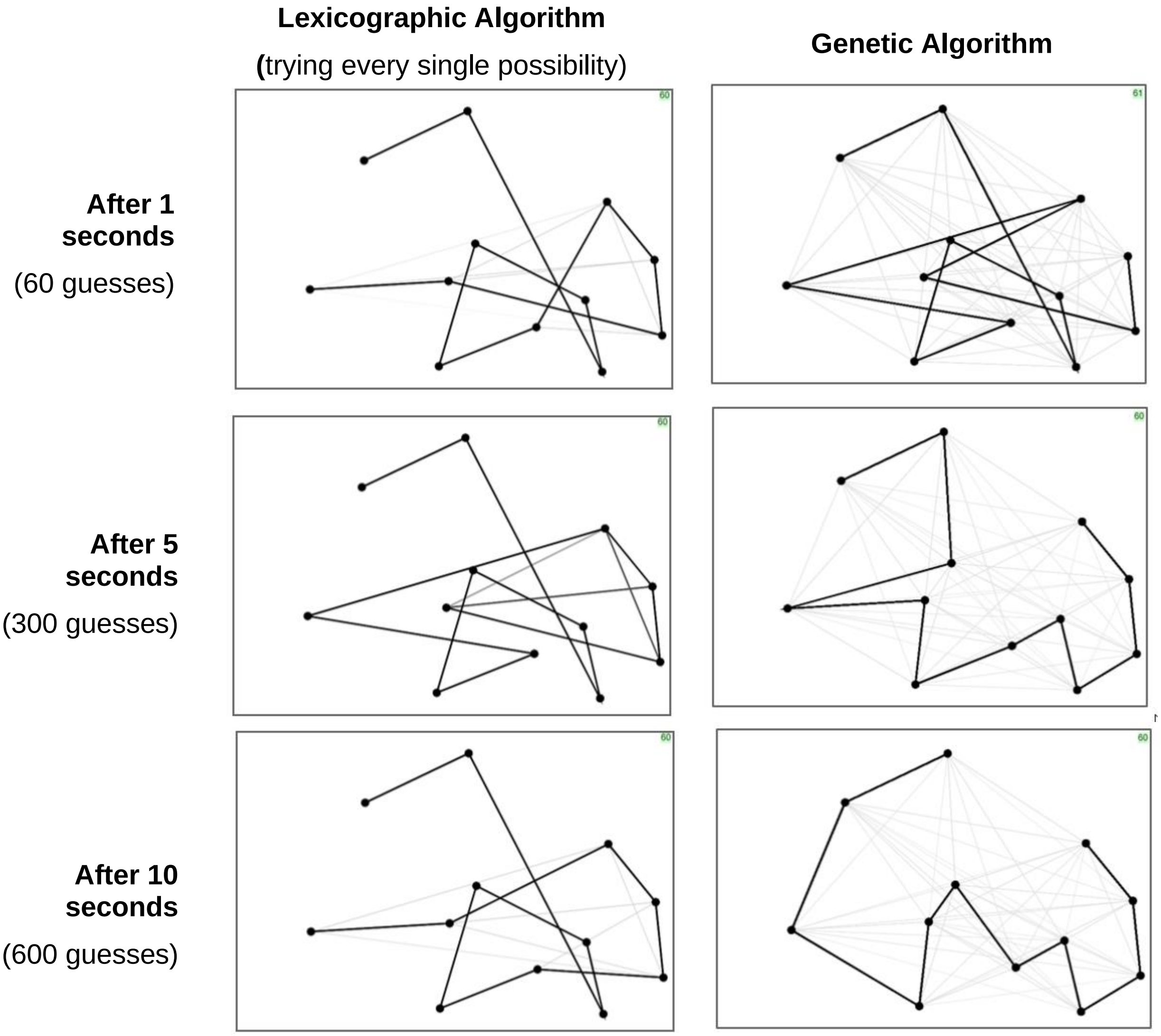
Can you imagine doing a similar process, but with a mechanical duct design? To understand how that might work, you can pretty much substitute the people in the diagram above with mechanical duct layouts, illustrated in the diagram below. Same process!



The same diagram as before, but now with mechanical designs instead of people. Can you see how it could work in a very similar way?

Genetic Algorithm applied to TSP

Let’s jump back to the Traveling Salesperson Problem (TSP) for a bit. In the below images, we pit a genetic algorithm against simple lexicographic ordering (AKA trying every single possibility in order). The screenshots show each algorithm after a certain number of seconds:



Screenshots from the EvolveLAB TSP animated solver highlighting differences between a lexicographic and genetic algorithm.

Note that these algorithms were run at the exact same speed (the same number of guesses per second). Therefore, in 10 seconds, they both made 600 guesses. The key point is that the **genetic algorithm made *better* guesses**. That’s metaheuristics at work!

The lighter lines in the background are related to animating how each algorithm works. Notice that in the genetic algorithm images, the background gray lines all offspring in the current generation. Check out the animated GIFs in the PowerPoint to really get the gist!

Cons of Genetic Algorithm

Local maxima / minima

The downside to using a Genetic Algorithm is that they can often get suck in what are called local maxima and minima. Even with extra mutations, just like evolution the variation in each generation tends to decrease.

Looking at the diagram below, as each generation (1, 2… through 8) is generated, the variation decreases. Even though the 'best' offspring are picked each time to create the new generation, if that variation is not enough to get from one peak or valley to another, the results will tend to cluster at what is called a local maxima. Where we really want to get to is the global max.

Please note that animated diagram makes this concept a bit more clear, so please refer to the class video or power point presentation for more information on this concept.

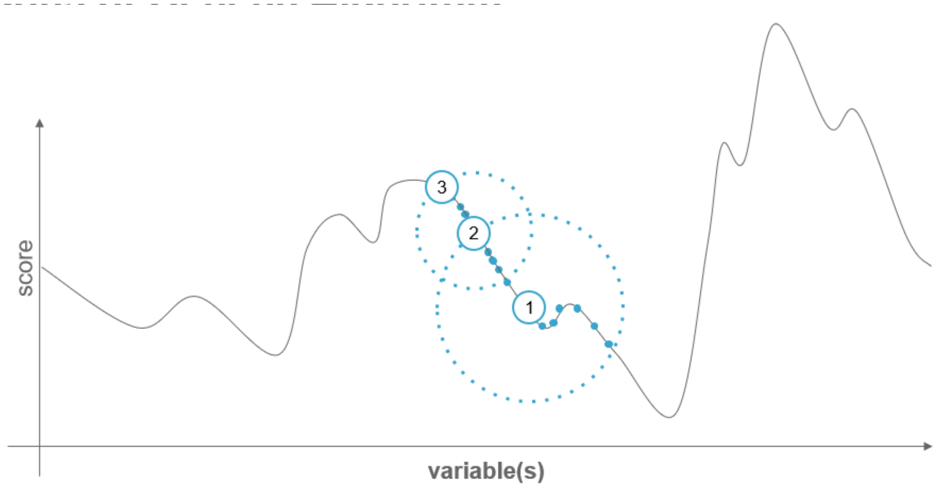


Diagram of how a genetic algorithm can get stuck in a local maxima, showing generations 1-3

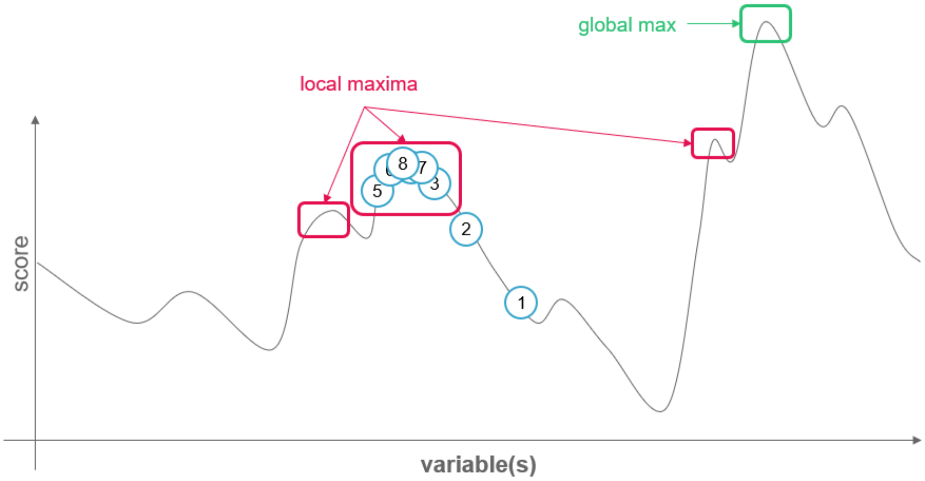


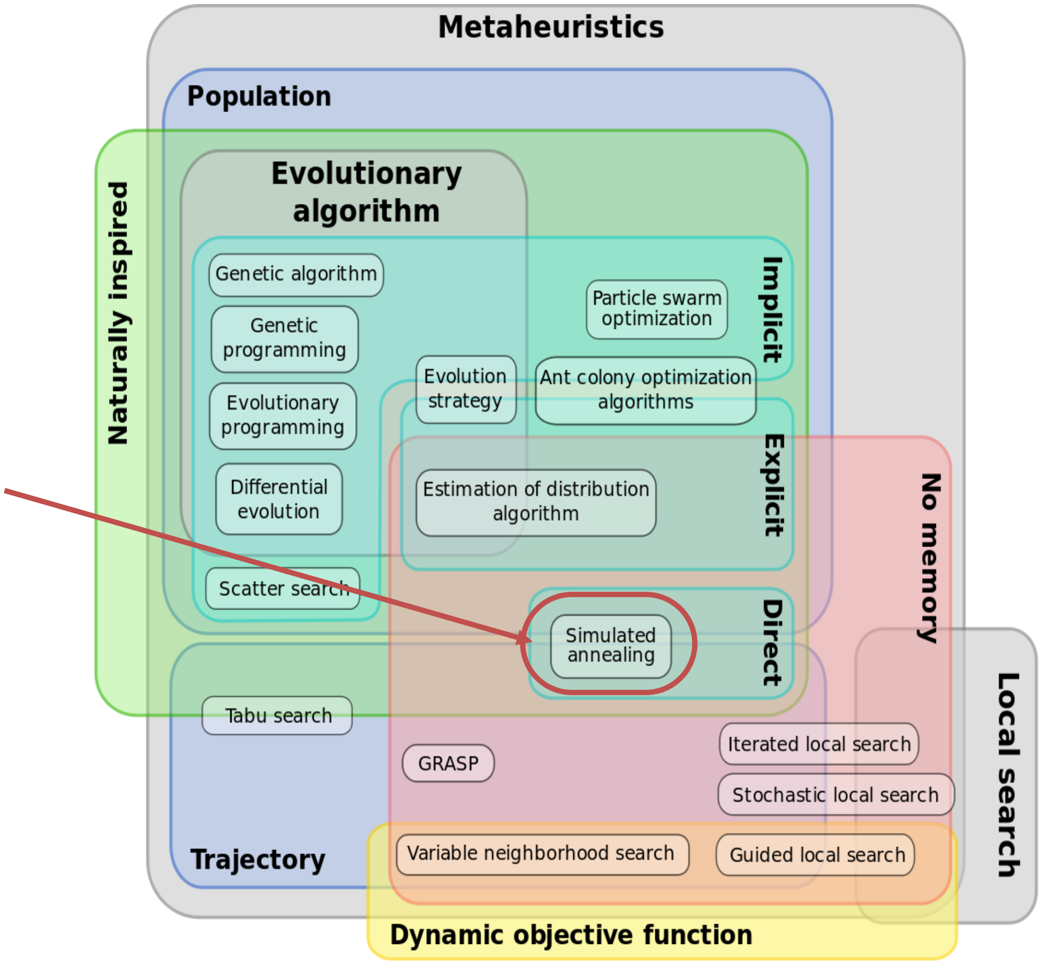
Diagram of how a genetic algorithm can get stuck in a local maxima, showing generations 1-8

Simulated Annealing

Simulated annealing is another metaheuristic algorithm that tends to be a bit faster than a genetic algorithm and is much better at avoiding local minima and maxima.

It is also a nature inspired algorithm. In this case, it is inspired by the process of annealing a metal, which is explained further on the next page.

Simulated annealing also tends to be a bit more customizable than a genetic algorithm, provided you with more variables, functions, and options to adjust to tweak it to your needs.



“Different classifications of metaheuristics shown as a Euler Diagram” by Johann "nojhan" Dréo, 28 August 2011 , with the simulated annealing algorithm circled.

https://commons.wikimedia.org/wiki/File:Metaheuristics_classification.svg

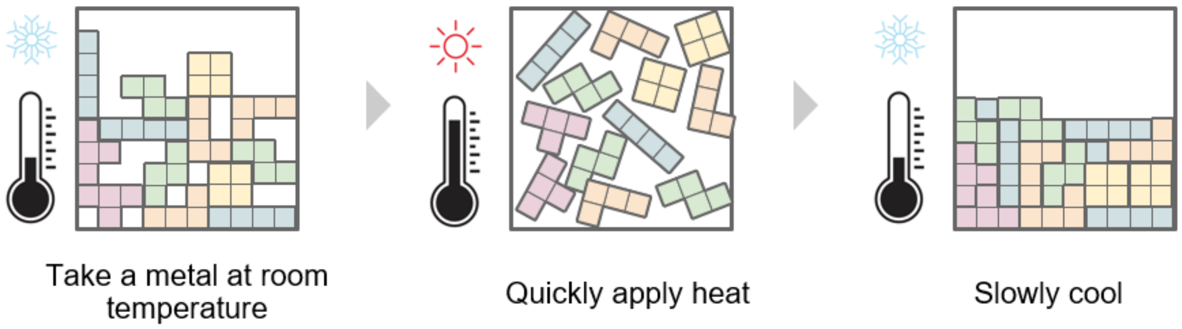
Annealing Diagram

Strengthen a metal by quickly heating and then slowly cooling

While most people are familiar with the natural process of evolution, not everyone outside metallurgy is familiar with annealing.

When you anneal a metal, you quickly apply heat, and then slowly let the metal cool. The idea being that as you heat up the molecules of the metal, they will have more energy and start moving around. As the metal slowly cools, the molecules will slowly settle and fill in any gaps and inconsistencies, ultimately creating a stronger material once the cooling process is complete.

In the diagram below, the annealing process is represented as Tetris pieces:



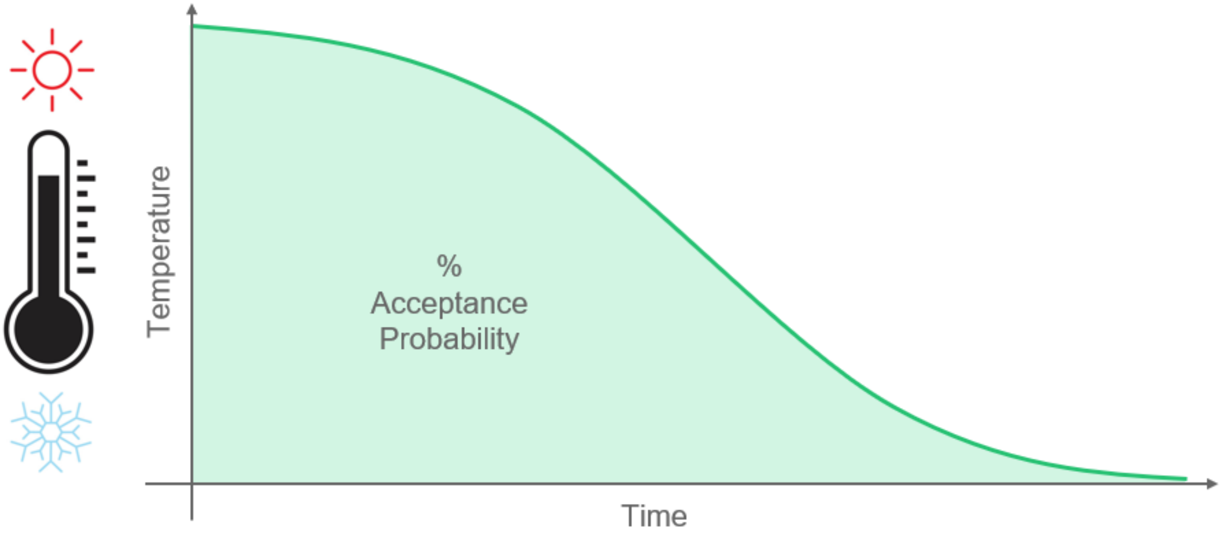
A diagram of the annealing process using Tetris pieces

‘Simulated’ Annealing Diagrams

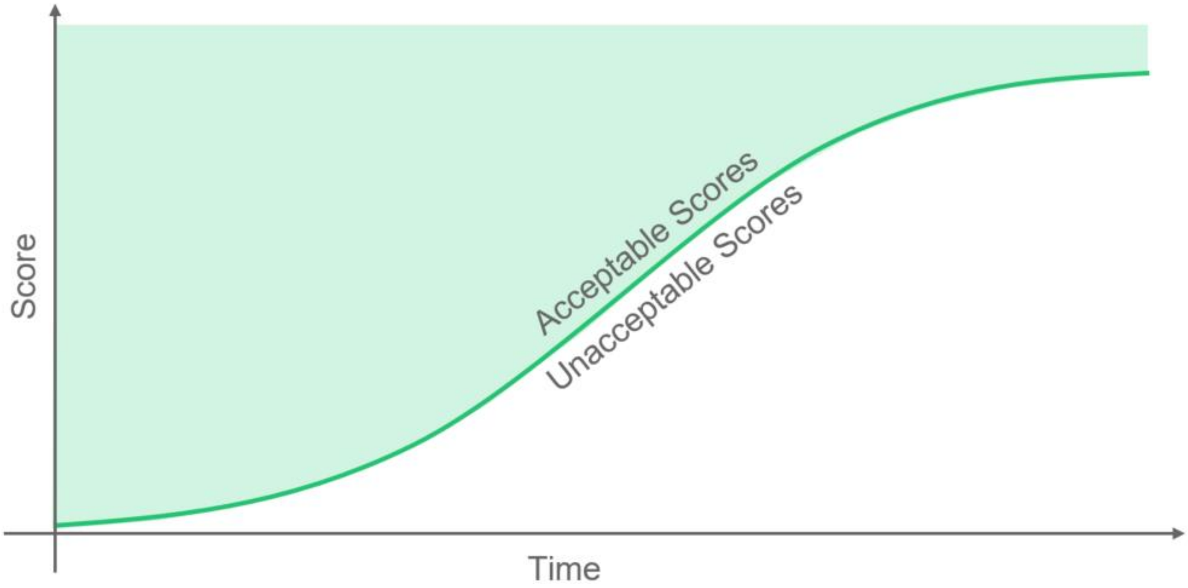
A ‘cool’ way to find a solution

Similar to the natural process, the ‘simulated’ annealing algorithm also uses a temperature, which starts high and is slowly decreased.

The temperature is correlated to what is called an acceptance probability. The higher the temperature, the more likely that the algorithm will temporarily accept a bad solution.



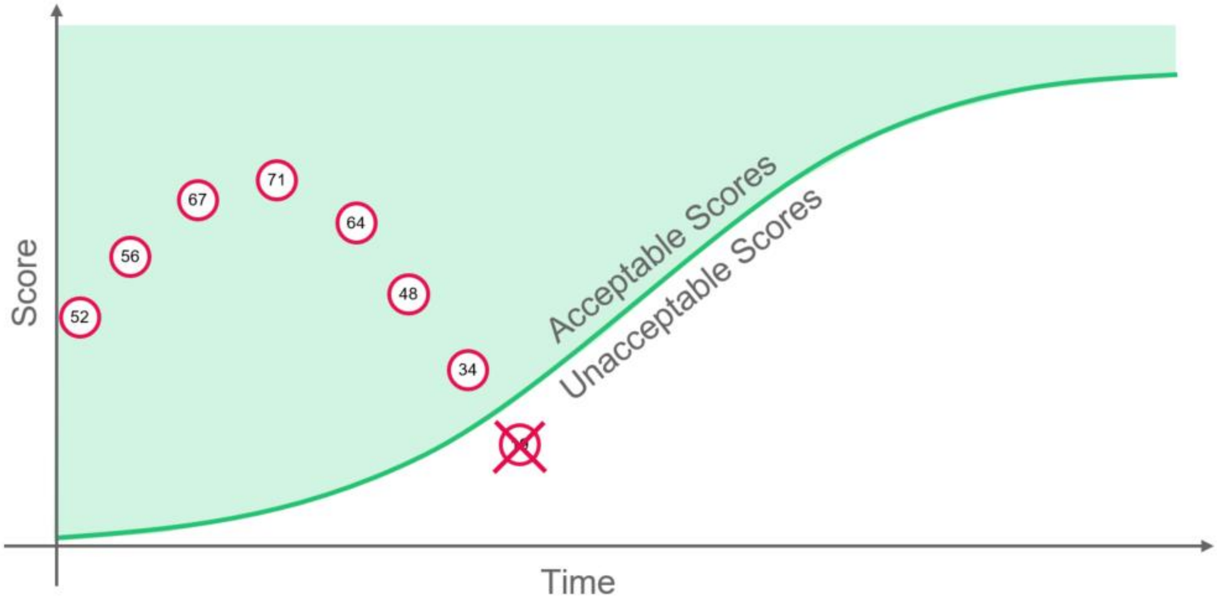
If we invert the above graph, and place score, not temperature, on the left, you can see that as time goes on, a solution's score needs to be higher in order to be acceptable.



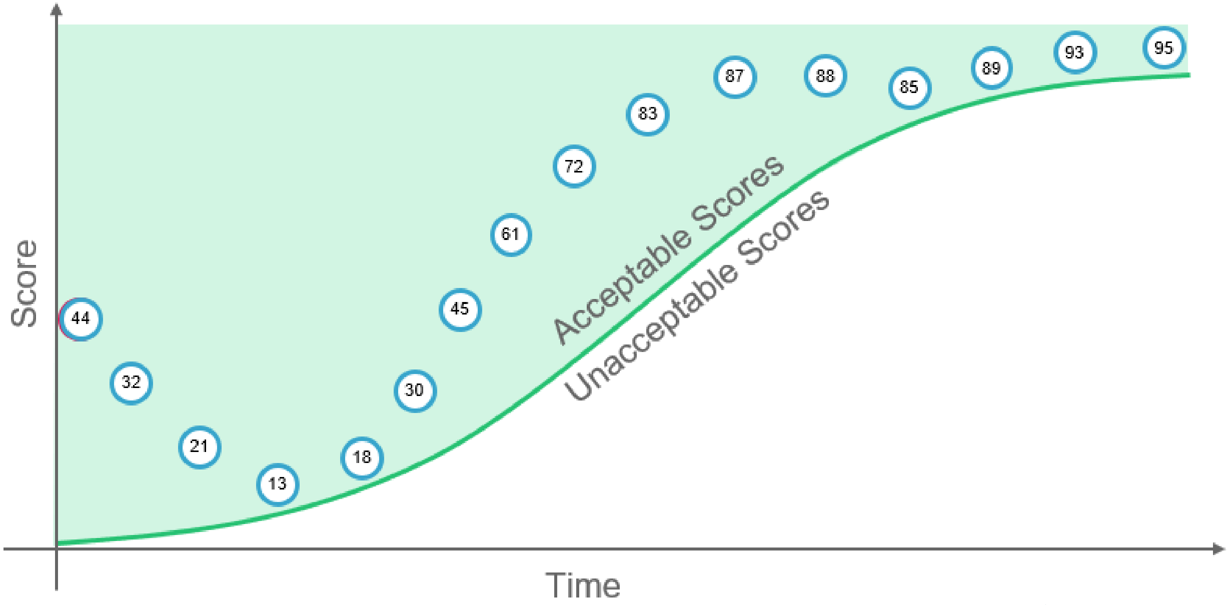
Instead of 'generations,' a simulated annealing algorithm continues to mutate, or iterate on, the same design. The algorithm will take a design, and continue to change it until either:

(1) **(red)** The designs score becomes unacceptably bad (which is relative to the current temperature), or
(2) **(blue)** The design becomes the best solution that's been found yet!

Consider the following two diagrams. In the 'red' one, the score gets up to 71, but then continues to get worse until it's unacceptable.



A diagram of a simulated annealing algorithm iterating on an idea until it becomes worse than the current temperature (AKA unacceptable solution score) will allow.



A diagram of a simulated annealing algorithm iterating on an idea that eventually becomes the new 'best' solution, because it stays within the 'acceptable score' zone, even as the temperature decreases as time goes on.

The key points here are that sometimes, a solution needs to be allowed to get worse before it can get better, like what happens in the blue graph. In the red graph, even though a new high was reached (71), it's quite possible that from that idea, it would be impossible to iterate to anything better – and only worse solutions would be produced from that idea. In a genetic algorithm, 71 would have been a local maxima that the algorithm got stuck at.

Sometimes you have to accept a worse solution to get to a better one. *We couldn't have invented the car by simply continuing to iterate on the horse.* Sometimes you need to take a step back!

To understand this concept further, I recommend checking out this section of the class via the video recording, as the explanation along with the animated diagrams makes things a bit more clear.

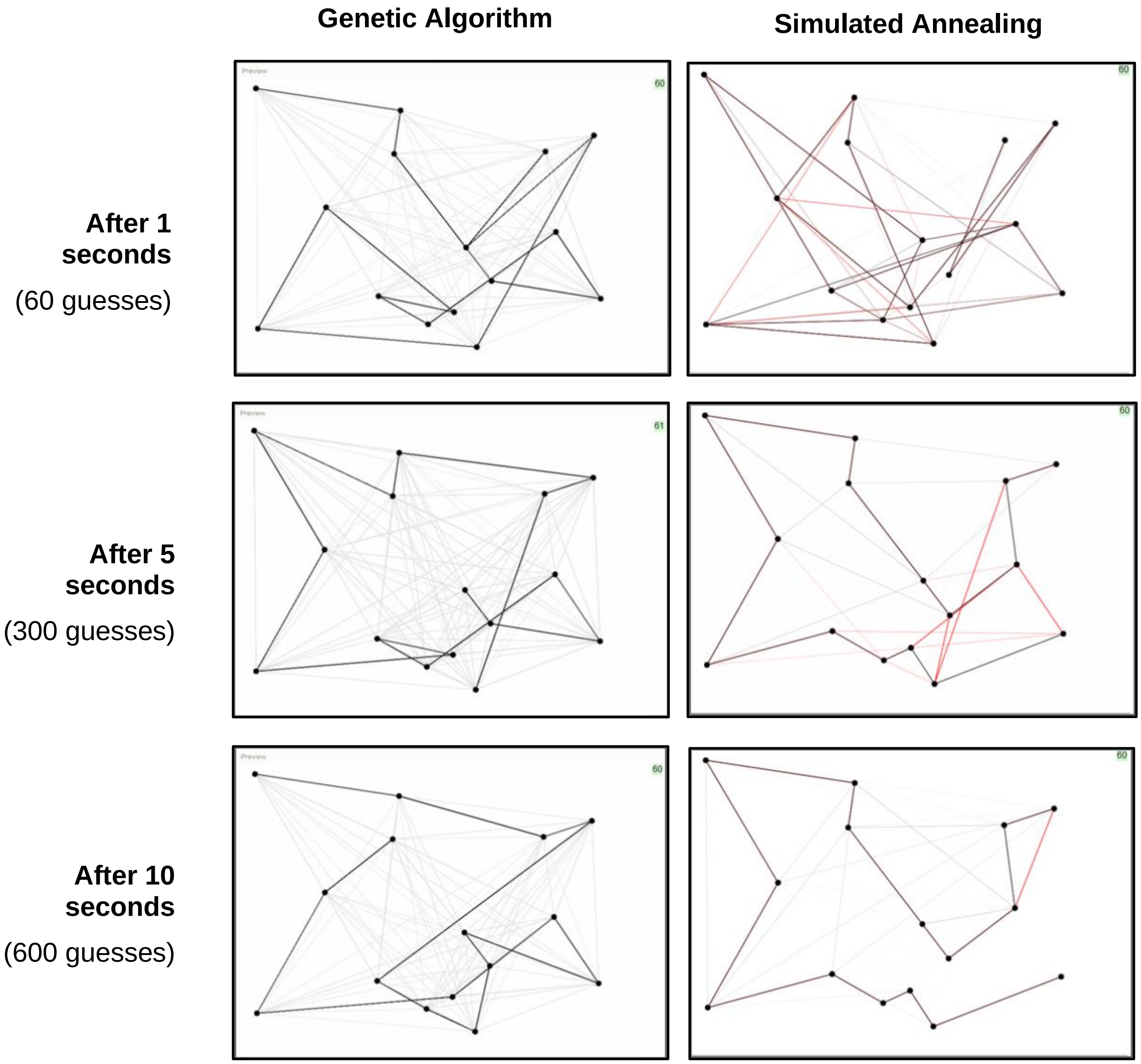
Simulated Annealing applied to TSP

Traveling Salesperson Problem reprise

Looking back at the TSP again, let's compare the Simulated Annealing (SA) to the genetic algorithm from earlier. As we saw before, the Genetic Algorithm has 1 generation's worth of light gray 'offspring' paths in the background. The SA, however, has three types of lines:

- **Black** – the shortest path found so far
- Gray – the last path that was tried
- **Red** – what I will call the 'currently annealing' path. This is not the best path, but it is the one that is currently being iterated on.

Note how, in the SA column, early on, the **red** line can be very different from the **black** 'best' path. But as time goes on, and the temperature drops, the red line deviates less and less from the current black 'best' solution. Again, this becomes extremely clean when looking at the animated diagrams in either the PowerPoint file or the class video.



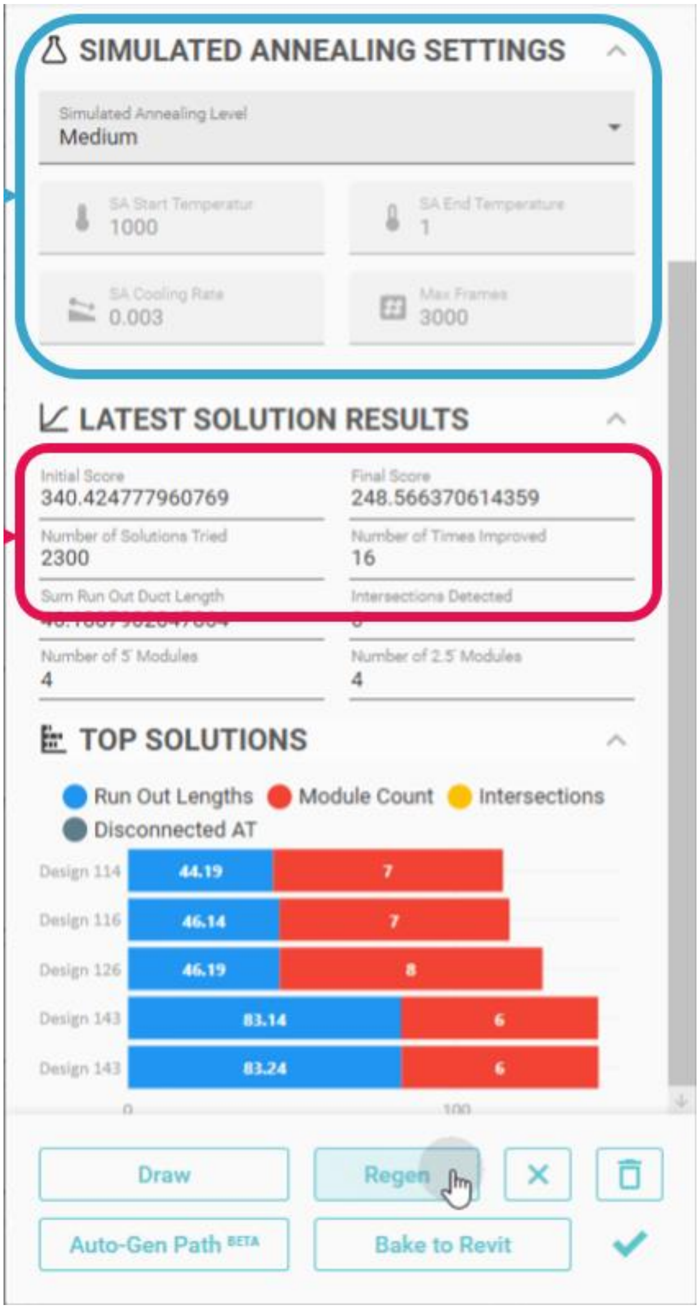
Screenshots from the EvolveLAB TSP animated solver highlighting differences between a genetic and simulated annealing algorithm.

In this example, we used 15 cities, instead of the 12 we used in the previous lexicographic / genetic TSP comparison diagram, which as you now know adds quite a bit more complexity. In this case, the left genetic algorithm got stuck in some sort of local max and did not improve much from 5 to 10 seconds. Notice how in the SA algorithm, as time goes on and the temperature drops, the **red** currently annealing solution that is being explored (even though it is not the current best) changes less and becomes less different form the ‘best’ **black** solution as time goes on and the temperature decreases.

Simulated Annealing for Duct Layouts

What was this class about again?

Bringing it back to Air Duct layouts – a simulated annealing algorithm is what is running in the background, creating, and iterating on various layouts. We don't draw all those light gray and **red** line options – we only draw the current best solution (the **black** line from the previous diagram) to the screen.



SIMULATED ANNEALING SETTINGS

Simulated Annealing Level: Medium

SA Start Temperatur: 1000

SA End Temperature: 1

SA Cooling Rate: 0.003

Max Frames: 3000

LATEST SOLUTION RESULTS

Initial Score: 340.424777960769

Final Score: 248.566370614359

Number of Solutions Tried: 2300

Number of Times Improved: 16

Sum Run Out Duct Length: [illegible]

Intersections Detected: 0

Number of 5' Modules: 4

Number of 2.5' Modules: 4

TOP SOLUTIONS

If we revisit some of the settings we talked about in section '*3 – The Solution'*, you might recognize some of the terms now! The *Simulated Annealing Settings* have a dropdown of presets – or there's a custom option that lets you adjust all of the simulated annealing values individually. Notice there's a starting and ending temperature – yup, that's the same temperature we discussed above that controls an SA algorithm. You can also set the *SA Cooling Rate,* which is how fast the temperature drops, as well as a *Max Frames* to add a safety time/iterations cap (you can always hit *Regen* to iterate further!).

The *Latest Solution Result* second contains information about the current / latest algorithm run. You can see the initial and final 'score,' as well as the number of guesses (*Number of Solutions Tried*) and the number of times a tried solution was better than and replaced the current 'best' solution (*Number of Times Improved)*.

These stats and settings allow for a lot of flexibility. You can let the algorithm run for 10 minutes to find a super optimized solution, or limit it to about 5 seconds, and just get a good-enough draft.

Notice that the '*Number of Solutions Tried'* is only 2,300. There have got to be millions, or possibly vigintillions of possibilities here, but a pretty decent solution was able to be found with only this many guesses!

Again, if you watch the animated gifs in the video or PowerPoint, you'll notice that the duct layout updates a lot as you draw. The application is always drawing the best solutions it has found so far (the **black** line from the previous TSP diagram), but constant iterations are still happening in the background – it's just not all rendered to the screen for clarity.

We found that having the current best solution drawn, even if it is not that good, is extremely helpful from a UI/UX perspective. The designer can get some insight into how the algorithm is thinking, and a preview of if their current branch duct will end up being a good or bad path. It produces much better results, from a user adoption standpoint, to have instant feedback, rather than waiting a few seconds after the designer finishes drawings to see only the best / final duct layout. We'll dive further into this in section '*5 – User Adoption'*.

Simulated Annealing Recap

Here's a quick summary of section *'4 – Simulated Annealing'*, because this section covered a lot. Some of the things we discussed includes:

- What Generative Design is, in the context of this application
- You might now be sick of the Traveling Salesperson Problem (TSP)
- Metaheuristics – a big new word to impress your friends, meaning basically large-scale educated guessing to find pretty good solutions to huge problems fairly quickly.
- Genetic algorithm – a nature-inspired way to generate designs, inspired by Darwin's theory of evolution
- Local minima / maxima – one downside of genetic algorithms is that they can get stuck in these
- Simulated Annealing – inspired by annealing metal, this algorithm uses a temperature to allow the exploration of bad ideas early, in hopes of exploring a wider solution space, and finding overall (globally) best solution.

Now that we understand how this application works, we are going to give into what is probably an even more important topic – how we actually got people to use it!

5 – User Adoption

Without adoption, your tool is just expensive marketing material

This is a point we'd like to emphasize, because we've seen many tools that had incredibly clever or useful functionalities, but ended up ultimately failing because nobody wanted to actually use them. In this section, we will highlight a few aspects of this application that we feel helped to foster strong user adoption. Let us dive in!

Integrated within Revit

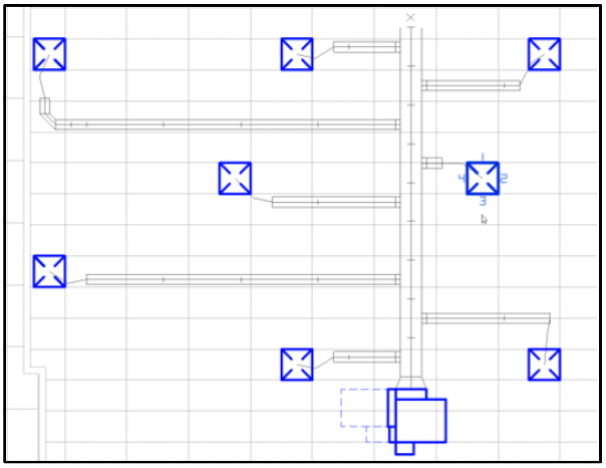
Minimize Learning New Behaviors

This app was built to work directly in the Revit canvas, to minimize the amount of new behavior a user would have to learn. For the most part, if you are familiar with Revit, you will be able to use this app with little to no training.

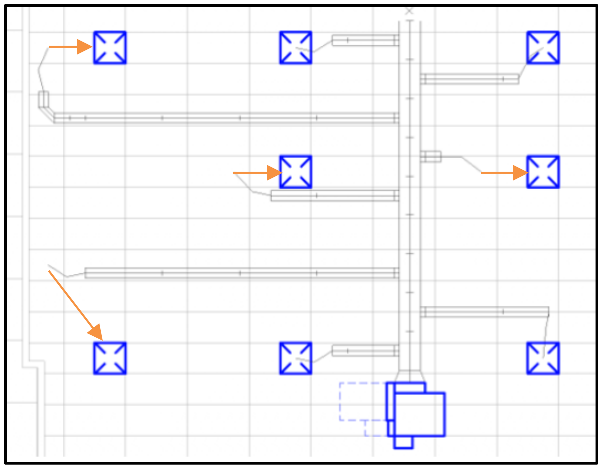
The air duct layouts are drawn directly in your current Revit view, so you can use all the same Revit navigation controls and keyboard shortcuts to get around. The drawing of the main branch duct also mimics the same click-release pattern that most native Revit drawing tools use.

The application also creates fully native Revit geometry (ducts, duct fittings, duct accessories, etc.) so that, once baked to Revit, the user can take the duct layout and modify or document it however they normally would with any other Revit content. This allows the application to integrate with other design methods, so it can be used for just a small part, or a majority, of the design!

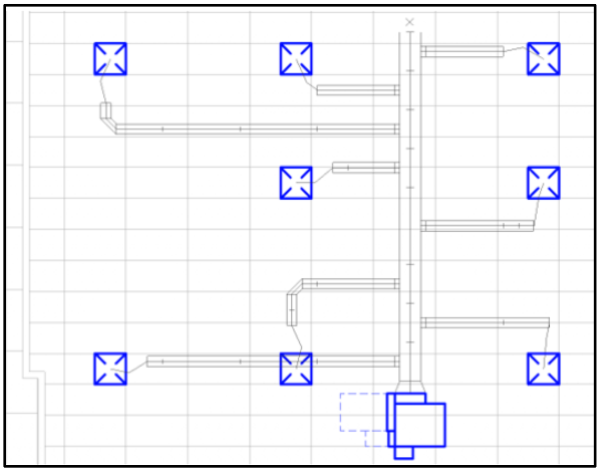
Working fully within Revit reduces friction and allows for a seamless modular duct designing experience with a low barrier-to-entry.



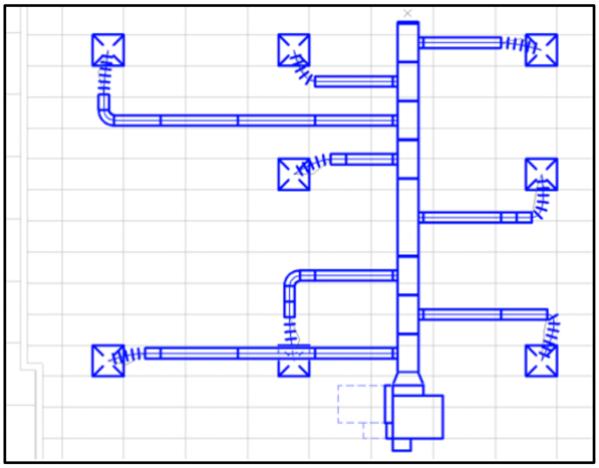
1. An initial air duct layout, with a VAV Box and Air Terminals creating using Revit elements, and temporary geometry showing the branch and run out paths created using the Modular Duct Generator app.



2. Some of the Air Terminals have been moved slightly by the user. The moves are depicted using the orange arrows. This is done using the native Revit move command.



3. The Modular Duct Generator App was run again, and the new branch and run out duct paths updated and adjusted to the new positions of the moved native Revit air terminals.



4. After baking to Revit, the temporary duct layout was converted into native fully connected Revit mechanical elements, including ducts, duct fittings, duct modules, and flex ducts.

App Architecture Tradeoffs

Supporting Co-Authoring

To really promote user adoption of this application, there were a number of tradeoffs that had to be made during the design and development of the Modular Duct Generator. These were all made with the consideration of useability and adoption sometimes at the cost of functionality or performance.

Interactivity over Speed

There's nothing more frustrating than waiting for an application with no visual feedback. We wanted to app to never feel like it was frozen, even if the application was still 'thinking.' The largest decision we made to achieve this was to always display results or feedback to the user, even if those results were yet incomplete.

As soon as you start drawing a new branch duct, the app starts trying out solutions. The current best solution, even if it's the app's terrible first guess, is always displayed, so the user knows that the app is indeed working. As soon as a better layout is found, it is immediately drawn to the screen instead. This provides lots of instant feedback at the start, but the longer the mouse hovers in the same spot, the better the currently displayed solution is, and therefore the less frequently the app updates. It gives you the feeling that the app is slowing down, as if it is 'settling' on a solution – which it actually is!

It is ok to have partial success, as opposed to seeing nothing until the algorithm is completely finished. The current best solution is rendered to the screen at 30 frames per second, so the animation appears super smooth. There is, of course, a performance cost to this – always display the current best at this speed results in the simulated annealing algorithm running a bit slower – but that's the tradeoff with this technique.

We also made sure to include loading animations on 'active' buttons, so it's easy to tell if the app is still thinking and might come up with more or better solutions.

Speed over Accuracy

As we touched on in the Simulated Annealing section, the choice of metaheuristic solver was to prioritize speed over accuracy. We would rather get a decent result super fast, then wait a long time to get a near-perfect result. The idea being, since everything is integrated with Revit – you can always tweak and edit the layout after baking it to Revit.

The 80-20 rule says that often times, the final 20% takes 80% of the work. In this case, we let the app take care of the lower-hanging fruit and automate most of the process, but instead of spending a ton of extra effort to cover all edge cases that would make up the final 20% - we allow the user to fill any inaccuracies. Again, since everything is right in Revit, you can use the app as far as you want, and take over with manual design wherever it makes sense to maximize efficiency.

Simplicity over Customizability

In general, the app does not expose that many settings. They are all saved externally, and loaded every time the application is open. The idea is to minimize clicks and make drawing a duct network as simple as drawing a wall.

There are tons of settings, that can be edited by an administrator, if needed, in a backend config file, but most of these are hidden from the average user. Presenting too many options can be intimidating, and actually decrease user adoption, so we tried to make an application that appears as simple as possible – sacrificing some customizability for the sake of simplicity.

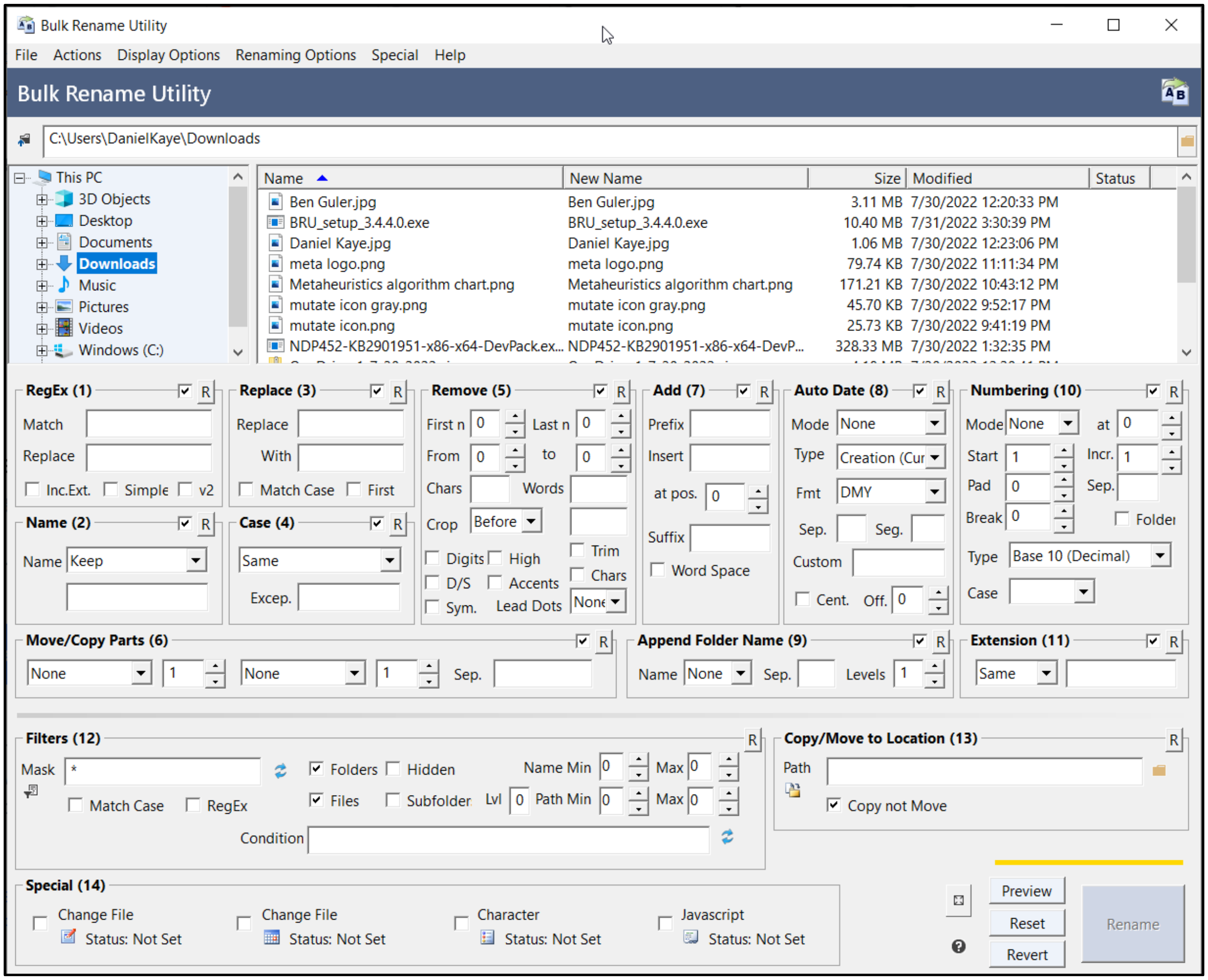
Specific over General

When trying to create custom tools to automate your workflows, starting from a well defined problem could make all the difference. Typically, trying to create a tool that is too general, but does not cover any specific task that well, ends up creating a tool that isn’t that useful to users – and ultimately hurts user adoption.

In this case, the problem was very modularized – a fixed VAV Box and Air Terminal Layout, and pre-designed modular pieces. Once you solve your specific problem well, you can slowly branch out to more specific problems to widen the tools scope. But going too broad, too soon, can hurt initial user adoption. Often times, once someone has tried a tool and didn’t like it, it somewhat taints their view of the tool for future tests, and could hurt adoption potential in the future, even if the tool did improve a lot!

UI Design

The UI design of any application is always extremely important, and often undervalued. Take a look at the below example, from the *Bulk Rename Utility* tool for Windows:



C:\Users\DanielKaye\Downloads

Name	New Name	Size	Modified	Status
Ben Guler.jpg	Ben Guler.jpg	3.11 MB	7/30/2022 12:20:33 PM	
BRU_setup_3.4.4.0.exe	BRU_setup_3.4.4.0.exe	10.40 MB	7/31/2022 3:30:39 PM	
Daniel Kaye.jpg	Daniel Kaye.jpg	1.06 MB	7/30/2022 12:23:06 PM	
meta logo.png	meta logo.png	79.74 KB	7/30/2022 11:11:34 PM	
Metaheuristics algorithm chart.png	Metaheuristics algorithm chart.png	171.21 KB	7/30/2022 10:43:12 PM	
mutate icon gray.png	mutate icon gray.png	45.70 KB	7/30/2022 9:52:17 PM	
mutate icon.png	mutate icon.png	25.73 KB	7/30/2022 9:41:19 PM	
NDP452-KB2901951-x86-x64-DevPack.ex...	NDP452-KB2901951-x86-x64-DevP...	328.33 MB	7/30/2022 1:32:35 PM	

RegEx (1) ☑ — Match: ; Replace: ; □ Inc.Ext. □ Simple □ v2

Replace (3) ☑ — Replace: ; With: ; □ Match Case □ First

Remove (5) ☑ — First n: 0; Last n: 0; From: 0; to: 0; Crop: Before; □ Digits □ High □ D/S □ Accents □ Sym. Lead Dots: None; □ Trim □ Chars

Add (7) ☑ — at pos.: 0; □ Word Space

Auto Date (8) ☑ — Mode: None; Type: Creation (Cur; Fmt: DMY; □ Cent. Off.: 0

Numbering (10) ☑ — Mode: None; at: 0; Start: 1; Incr.: 1; Pad: 0; Break: 0; □ Folder; Type: Base 10 (Decimal)

Name (2) ☑ — Name: Keep

Case (4) ☑ — Same

Move/Copy Parts (6) ☑ — None; 1; None; 1

Append Folder Name (9) ☑ — Name: None; Levels: 1

Extension (11) ☑ — Same

Filters (12) — Mask: *; ☑ Folders □ Hidden; Name Min: 0; Max: 0; □ Match Case □ RegEx; ☑ Files □ Subfolder; Lvl: 0; Path Min: 0; Max: 0

Copy/Move to Location (13) — ☑ Copy not Move

Special (14) — □ Change File Status: Not Set; □ Change File Status: Not Set; □ Character Status: Not Set; □ Javascript Status: Not Set

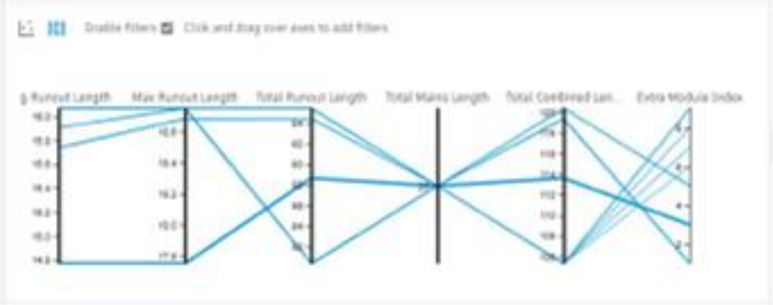
Screenshot of the Bulk Rename Utility tool, a small application for Windows that allows users to rename multiple files at once using a complex array of settings.

At first glance, the above UI may seem quite overwhelming! Even though I personally love this tool and it is extremely feature-rich, trying to visually parse the UI to understand how it works can be quite difficult and time consuming at first.

The Modular Duct Generator App UI was designed around some basic UI principles, the first of which is its vertical organization. Many applications, and most webpages, follow this format these days – so you can tag onto more already-learned user behaviors. The app was designed to be horizontally compact, so that you could easily use it while still being able to see the Revit view you are working inside of.

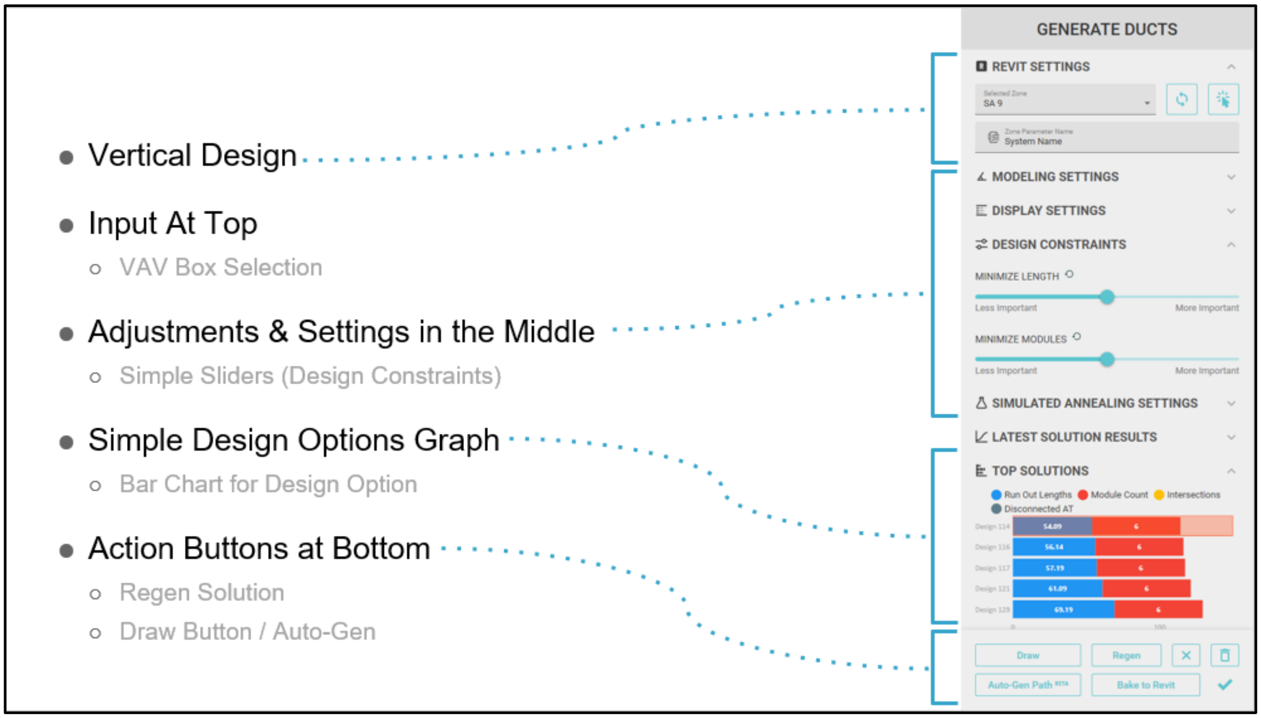
It's not just the orientation that is vertical, but the app is also used from top to bottom. All inputs are placed at the very top, followed by the settings in the middle. As you continue to scroll down, the results are in the bottom portion. The very bottom of the UI is what we call an action bar – a static horizontal band that is always visible, and contains the buttons that trigger the main app features (Draw, Delete, Bake to Revit, etc).

All settings were kept as simple as possible – in this case they are all either sliders or on/off toggles. If you need to know a bit more, all the tools have tool tips that display more detailed information when you hover the mouse over them. This allows the UI to remain generally uncluttered, but still have a depth of information when it is needed.



An example of a 'parallel coordinates plot' from Revit Generative Design

The bar graphs in the *Top Solutions* section are an adapted version of the 'parallel coordinates plot' you may be familiar with from Revit GD or other tools. The overall bars are always sorted in order from shortest (AKA best) at the top, and longest (AKA worst score) at the bottom. This gives you a quick overall ranking of each solution. However, the lengths of each individual bar show the portion of the score determined by each scoring component. As noted earlier, hovering over each bar will provide even more detailed information above each potential solution.



- Vertical Design
- Input At Top
  - VAV Box Selection
- Adjustments & Settings in the Middle
  - Simple Sliders (Design Constraints)
- Simple Design Options Graph
  - Bar Chart for Design Option
- Action Buttons at Bottom
  - Regen Solution
  - Draw Button / Auto-Gen

Screenshot of the UI of the Modular Duct Generator App, broken down into sections, each labeled with their general function.

UX Design

Having a thoughtful User Experience design is also extremely important for user adoption. This is what helps a user build trust in the application, and without trust it's very hard to get people onboard with a new technology.

One main goal was to never make the user wait because in general, people are impatient, especially when it comes to technology. That's why, as describer earlier, we are always constantly displaying the latest and greatest duct layouts to the user – even if they aren't fully resolved yet.

We used very minimal lightweight geometry to display the duct layout output (before you've baked it to Revit), so that we could technically achieve what feels like an instant response from the app. As soon as you move the mouse, the layout adjusts, and the algorithm starts running on and improving the design based on your new position. On the backend this is actually quite difficult to do, but worth the effort when you see the result – it really inspires what you can do within Revit.

I recommend checking out the ortho mode on/off gifs in either the PowerPoint or the class Video, as it is hard to describe with static images how smooth the app actually feels when you are drawing with in!

Workable Output

We have mentioned this a few times before, but it's so important we have a whole section on it – whatever the result of your tool, it really does need to be easily malleable by the user. If they can't get exactly what they want from the tool, and they can't tweak it afterwards, they will become frustrated which leads to much lower user adoption.

This app created a fully connected mechanical system using Revit families. Once it's in Revit, the user can tweak, change, or even delete the whole thing if they want. This allows for partial success – even if the tool's proposed solution doesn't get you 100% of the way there – it likely still saved a lot of time, and you can just do the last bit manually. And the best part is that, since everything is connected, you can move one Duct Module, and the whole branch and flex duct will also adjust accordingly to keep everything connected. That is all just native Revit behavior! You can also run any analysis, takeoffs, or scheduling just as you normally would.

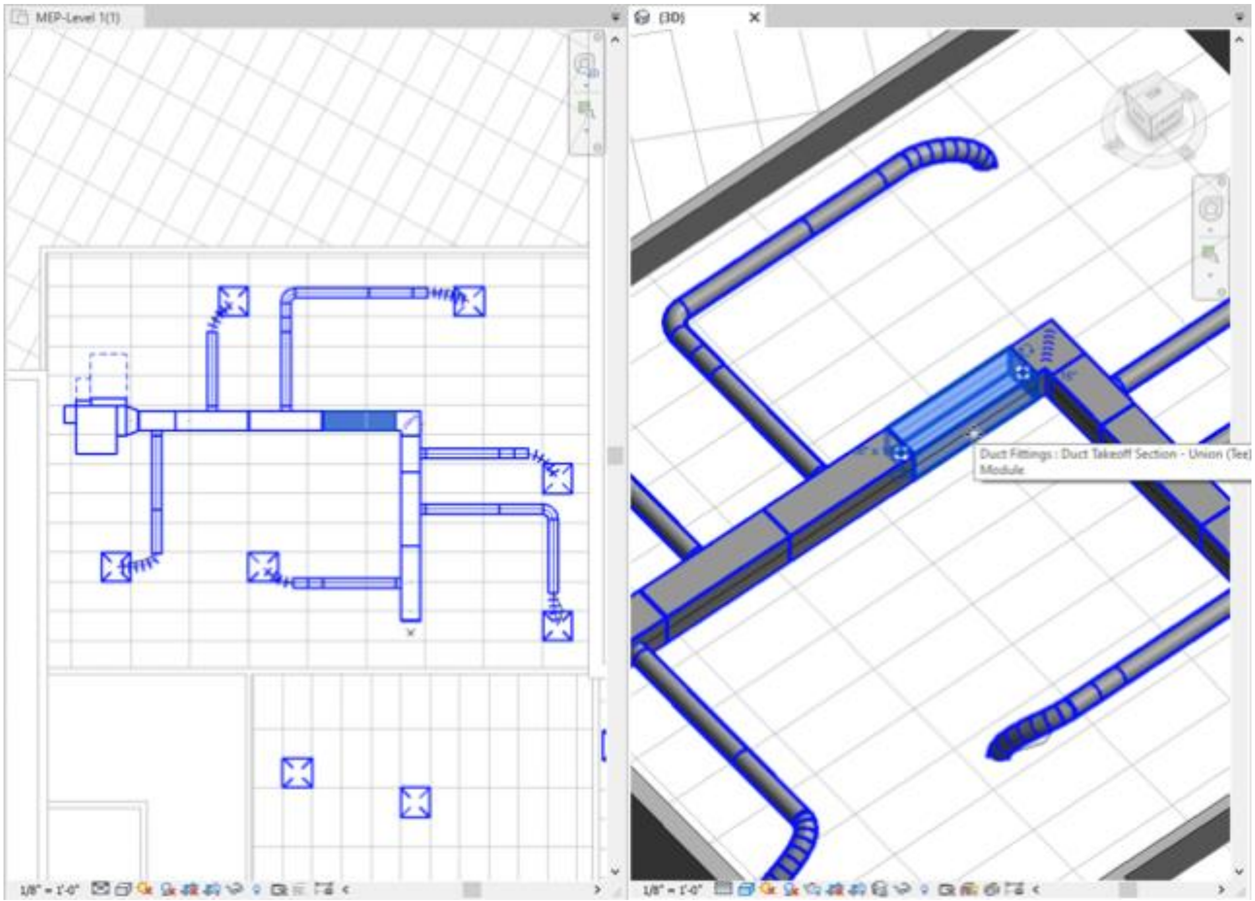
Of course, we'd like the app to get it right every time, but developing a tool to cover all possible edge cases is way more work than covering the most common cases, and letting the users fill in the gaps. This way, we can always improve the tool to handle more situations when needed, but we won't waste resources by writing a lot of code coverage that ends up having a diminishing return-on-investment.

Make It Fun

It sounds cheesy, but it's true! The more fun and satisfying an app is to use, the more likely users will continue to utilize it. It's really rewarding to a mechanical designer when they can complete their task faster, and the interactivity between the designer and the app really does make it a joy to use. There's something really satisfying about pressing the *Bake to Revit,* and having all those Revit components generate almost instantly in front of your eyes.

There is always a correlation between how much effort is required to learn a new tool, and how much automation it offers. The point being – if nobody wants to use your tool (AKA low user adoption), then it really doesn't matter how good the algorithm is. Our choice of metaheuristic solver isn't the fastest or most accurate possible, but the UI/UX benefits of the interactivity and instant feedback in a way make up for that.

Again, you don't want to invest a lot of time and money in the perfect algorithm if the tool that allows the users to interface with it is frustrating. It's a balancing act, where the useability of the app is just as important, or perhaps more-so, than the features that it packs.



A screenshot of the Modular Duct Generator App output inside 2 Revit views – a plan and a 3D view – after a generated layout has been baked to Revit.

6 – Beyond Ducts

Other potential applications for this technology

Now we are going to take a step back from the world of modularized air duct layouts for a bit and try to look at the bigger picture. What else could the ideas and technologies used in the Modular Duct Generator App be used for? Could you apply them to lighting layouts? Egress path optimization? Schematic building design?

Below, we are going to take a look at just a few examples of where these ideas could potentially be used, explore the possibilities beyond just ducts.

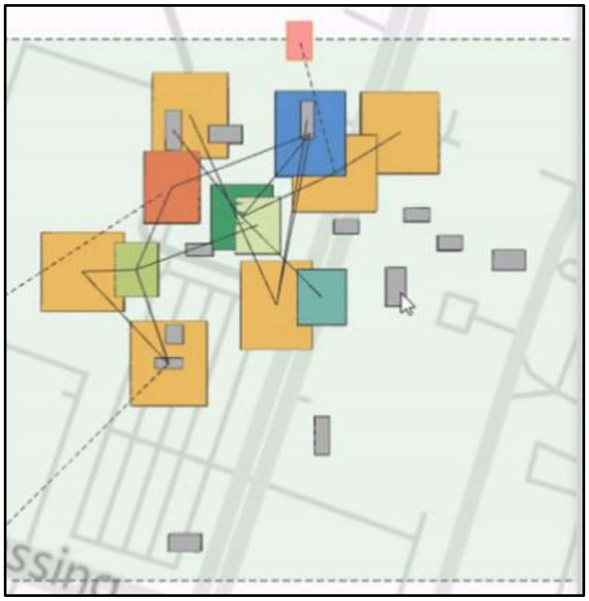
Schematic Plan Generation

Using physics to create designs

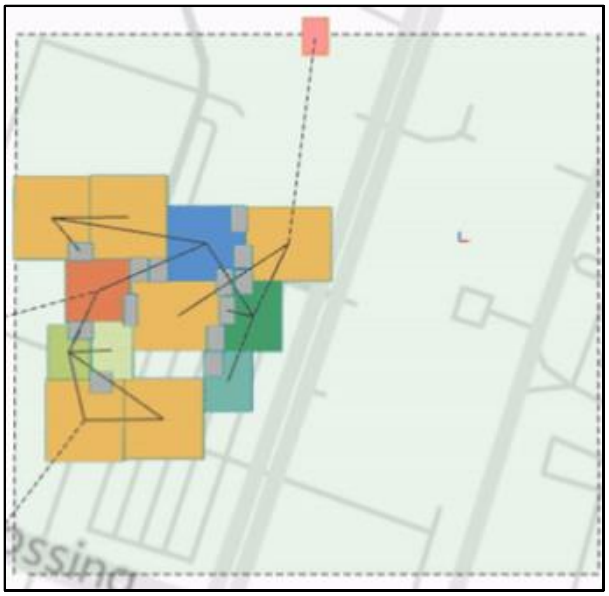
This was a project for an architectural client that wanted to generate interactive and scored schematic floor plans. It used a physics-based metaheuristic generative design algorithm, not too dissimilar from the Simulated Annealing one used for the Modular Duct Generator App.

The tool employs similar co-authoring techniques. Even though it runs on a very different platform, it still exports to Revit. The main difference might be that this project had over 30 different optimization targets, with a design constraint slider (similar to this app) for each one!

It uses very similar underlying principles, but the results are much more schematic and in the architectural layout space instead of a fully-designed modular system! I could give a whole talk just on the algorithm(s) used in this project – maybe at a future Autodesk University!



Screenshot of a schematic plan generator tool, in the earlier stages of generating a layout.



Screenshot of a the same schematic plan generator tool in the later stages of generating a layout.

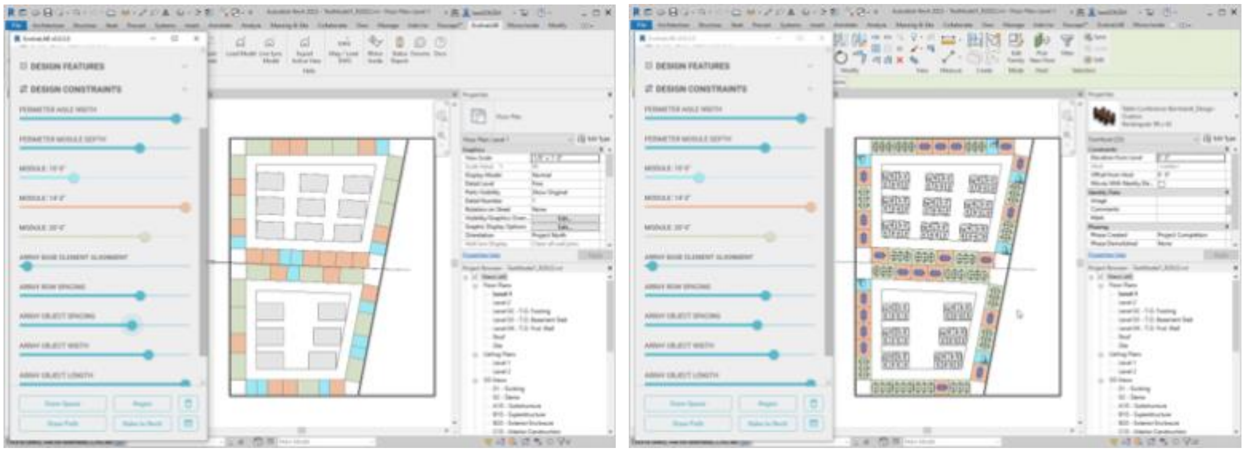
Morphis

A space layout and array tool gone wild

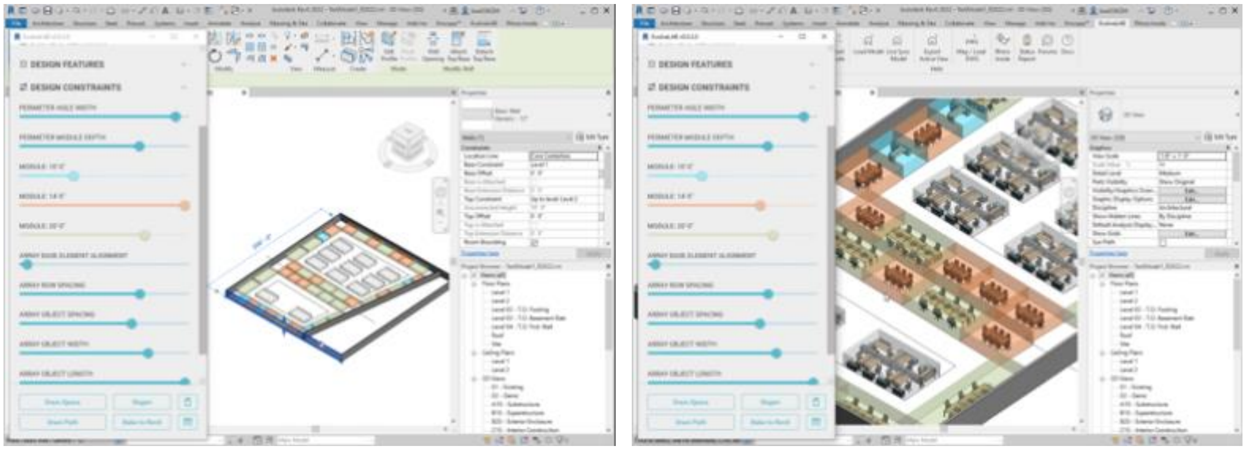
Morphis is a generative space layout and complex array tools that is currently being developed by EvolveLAB. This one, similar to the Modular Duct Generator App, operates completely inside of Revit.

It uses similar techniques to create a very responsive and snappy UI with immediate feedback and results to the designer. Currently, it lays out perimeter spaces and central arrays, and uses a metaheuristic generative design engine to optimize the results based on the UI slider inputs.

Another similarity is that it bakes the completed geometry into chosen native Revit elements – like rooms or model groups! It is essentially a tool built on similar ideas, but generates space arrays instead of ducts. Keep an eye out for the beta release in the near future from EvolveLAB.



Screenshots of the Morphis tool in Plan View, showing various design constraint UI sliders, and a generated layout that gets baked into Revit spaces and Model Groups.



Screenshots of the Morphis tool in a 3D View, showing again the various design constraint sliders in the UI, and a generated layout that gets baked into Revit spaces and Model Groups.

Endless Possibilities

What else can you do with this technology?

Thinking back to section *3 – The Solution*, I can easily see this tool continuing to improve on that beta auto-generate branch duct feature. I could also see this tool expand to layout and optimize the Air Terminal Placements, move the VAV Boxes around, and even create optimized mechanical zones for the building.

The point being, there are really endless possibilities for similar tools, in other disciplines (structural, electrical, fire protection, etc.) and even beyond MEP (architectural design, construction optimization, etc.).

Below is a list of a few of the other ideas we've had. Some of these we have worked on in the past, or are working on now, and others are just a few of the ideas for other similar tools that we could imagine in the future.

- Expand on Mechanical Duct Generator
  - Air Terminal Layouts
  - Branch Duct Paths
  - VAV Box Locations
  - Zoning Efficiency
- Electrical
  - Lighting layouts
- Fire Protection
  - Sprinkler Plans
  - Gravity-drained Pipe Layouts
- Plumbing
  - Fixture arrangements
  - Pipe Routing
- Architectural
  - Schematic plan layouts
  - BOMA rental unit optimization
  - Stadium, auditorium, and conference hall seating and booth layouts
  - Facade design and optimization
  - Curtain wall layouts
- Construction
  - Panel board placement optimization
  - Modular wall panel segmentations

Looking at the above list, do you have any ideas we missed? Can you see where some of these ideas could be implemented in your particular area of expertise?

Recap

[What] did we learn?

First, we talked about the problem that we were trying to solve – generating modular air duct layouts right inside Revit in an intuitive way that Henderson Engineer’s staff would be willing to try.

Then, we gave an overview of the Modular Duct Generator App with a simple 6-step explanation of how to use it. Basically, you draw a line, and it generates modular duct layouts.

In the middle, we dove into generative design, metaheuristic algorithms (genetic, simulated annealing), and talked about how they work and their pros and cons.

Most importantly, we covered how to promote user adoption of a tool like this through various design choices and UI/UX techniques.

If there’s one takeaway form this, it is understanding that the concepts discussed above are not at all specific to modular ducts – they could be applied to a whole bunch of problems. Let’s let the computers do the optimization they are good at, and let the humans do the higher-level thinking, design work, and tweaks we are good at. With this approach, we can create ever more tools that help make all sorts of architectural, engineering, and construction tasks more efficient – to get more work done more efficiently and better in less time!

Have any questions on this course? Want to geek out even more about simulated annealing? Are you inspired to try making your own tool?

Feel free to reach out to Daniel and Ben anytime! We love to talk about this stuff: