Introduction to Trusses

A truss is a structure of axial members (elements) connected at pinned joints (nodes), such that no moment may be transferred between members. Trusses are designed to efficiently distribute load. Examples of trusses structures range from bridges and cell phone towers to internal and external supporting components in additively manufactured devices.

One method of efficient truss design that is particularly relevant for additive manufacturing applications (where engineering complexity is not an issue) is weight minimization.

Genetic Algorithm

A Genetic Algorithm imitates natural selection by producing iterative generations of increasingly fit populations until converging on a near optimal solution. The algorithm evaluates the fitness of each member of a population of possible solutions and "breeds" the most-fit members to generate the subsequent population.

Each truss is encoded in a binary chromosome, a series of binary digits representing the connected members of the truss. The breeding step includes crossing over (cutting and recombining two fit trusses to blend their attributes) and mutation (random switching of binary values during reproduction by producing iterative generations of individuals). A "1" represents an element connecting the two nodes. The pairings progress in sequential order:

\[
1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1
\]

The binary chromosome above represents an encoding of a truss with 9 nodes in the design space. The numbers in red signify the nodes to be connected by an element if the slot immediately above is a "1". While the fitness certainly moves toward a more optimal value, even visual inspection of the resulting truss diagrams suggests that the program does not find an absolute optimum.

Key Features of Genetic Algorithm

- **MASTAN2 Integration:** Users define geometry of problem in MASTAN’s graphical interface. Four outermost nodes define the available design space, and nodes with support and load conditions establish the optimization problem.

- **MATLAB Interface:** Algorithm uses text-based interface to allow users to input key parameters, including number of generations, population size, and various parameters defining random truss generation and mutation.

- **Grid of Possible Nodes:** The algorithm randomly generates an initial population of trusses and ensures complete interconnectivity of all essential nodes (support and load locations). It then randomly adds and removes elements based on a mutation probability.

- **Uniqueness Encoding:** Trusses are stored in a binary chromosome used in the breeding step, and are converted to and from matrices for fitness evaluation. Each spot on the chromosome represents a pairing between two possible nodes in the design space. A "1" represents an element connecting the two nodes. The pairings progress in sequential order:

\[
1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1
\]

The binary chromosome above represents an encoding of a truss with 9 possible nodes in the design space. The numbers in red signify the nodes to be connected by an element if the slot immediately above is a "1".

- **Elitist Selection:** The top ten trusses from each generation are passed identically (unmutated) to the subsequent generation, guaranteeing that the top fitness will not deteriorate from one generation to the next.

- **Element Removal Preference:** In order to encourage the program to find lighter (more fit) solutions, the algorithm is 60 times more likely during the mutation step to randomly remove an element than to randomly add one. This significantly increases the rate of fitness improvement and prevents premature convergence.

- **Population Size:** 2000. Number of iterations: 1000

Results

The fitness plot shows that convergence is obtained after approximately 750 runs.

Summary and Acknowledgements

- The algorithm effectively converges towards a more optimal truss design.

- More strongly guided method for generating the initial population may allow for an improved final result.

- Thank you to Emma Lejeune and Ren Gibbons for mentorship, Christian Linder for sharing his lab and resources, Berkin Dortdivanlioglu for assistance in using the remote workstations, and Stanford CEE VPUE for funding this project.

References