Going in a Loop With Mixed Integer Linear Programming

Rohan Jhunjhunwala[1]

[1]Personal Project, BS, EECS Berkeley

Abstract

We present a Mixed Integer Linear Programming formulation of a recreational mathematics problem and discuss some practical limitations of the model.

The resulting formulation is able to find a simple cycle of length 50 kilometers running only on roads in Jersey City within ~1.25 kilometers of its center.

Keywords: Integer Linear Programming, Nonlinear Programming, Computational Geometry, Ultramarathon Running

1. Formal problem statement

Given the set of "Established roads and trails" in a given region find the route in this area with the best ratio of distance to diameter which is still a simple cycle.

This is the predicate of the "Long Tiny Loop" challenge. The challenge creator explains in further detail, "Long Tiny Loop is a fitness challenge that also tests your navigational and cartographic aptitude. Your goal is to traverse the longest possible non-self-intersecting loop within the smallest possible region, without revisiting any streets or intersections. If you love computational geometry and graph theory almost as much as you love outdoor workouts, this site is for you." [1]

To complete this challenge, we present an integer linear programming formulation of this problem.

- In the first section, we discuss the challenges of sourcing a dataset for the list of "established roads and trails" and techniques for reducing its size.

[1]https://longtinyloop.com/faq

- In the second section, we present a naive model along with successive refinements that improve performance, making it feasible to apply the model to large datasets, such as those of entire cities.
- Finally, we describe potential future work, including an exploration of the market for commercial optimization software.

2. Data Ingestion

Data ingestion for this problem presents several challenges. The phrase "established road or trail" is quite ambiguous. Most urban streets have a sidewalk on each side, but the challenge, as stated, is interpreted over a set of undirected edges. Therefore, it would not be permissible to run down the sidewalk on the right and return on the sidewalk on the left. However, OpenStreetMap generally presents these as separate roads.

To address this, we used several techniques to pre-process the data. These techniques are as follows:

- **R-tree algorithms**: We utilized R-trees and disjoint sets to merge nearby vertices into a single vertex, remove parallel roads and split roads with nearby vertices into multiple roads. These algorithms helped clean but often introduced a considerable amount of noise. We had to careful to tune the parameters to ensure we didn't oversimplify the graph and introduce spurious connections.
- **Pruning the graph**: We removed "Isolated Vertices", bridges, and degree 2 vertices to construct a simplified graph whose solutions are in one-to-one correspondence with our original graph.
- **Optional Manual validation**: Once the data was cleaned, we manually reviewed the resulting cycles from the model.

3. Model Formulation

Using basic nomenclature, we can begin to formalize a mathematical description of the problem. First, we will present the problem using abstract combinatorial notation (lists/sets), and then we will discuss how each component can be made linear.

$V \subseteq \mathbf{R}^2$	The coordinates of street intersections.
$l \in \mathbf{R}$	The distance I'm willing to run.
$E \subseteq V \times V$	The set of roads, as pairs of vertices.
$C \subseteq \mathcal{P}(E)$	The set of all cycles of edges.
$V_c \subseteq \mathcal{P}(V)$	The subset of vertices representing the path I run.
$W_e \in \mathbf{R}$	The length of a given edge e.

We can now formulate the model in the abstract. We intentionally leave the definition of some terms like *diameter* vague to make room for a more formal model later on.

$$\begin{aligned} \max_{c \in C} \quad & \frac{\sum_{e \in c} W_e}{\text{diameter}(c)} \\ \text{s.t.} \quad & \sum_{e \in E} W_e \leq l \\ & \forall s \in \mathcal{P}(V_c), \exists (u, v) \in E, \quad \text{s.t. } u \in s \text{ and } v \in V_c \setminus s \end{aligned}$$

Looking at this formulation, we see a challenge. Ostensibly, we have $O(2^{|V|})$ constraints. Fortunately, modern solvers allow us to insert these cuts only each time we notice a violation. This approach is known as the "DFJ" formulation for subtour elimination and is typically quite performant in practice. [2]

We still face challenges in converting this into a workable Mixed Integer Linear Programming (MILP) formulation. First, computing the diameter of C is difficult. Then, once we have both the diameter and the length of the cycle, we need to compute their ratio. There are several ways to handle this, but we chose the most maintainable and performant approach for our optimization software, Gurobi.

For the division, we have the optimizer maximize the difference between the logarithms of the length and the diameter. To compute this logarithm we utilize a piecewise linear approximation. Gurobi handles the details of the piecewise linear constraint for us. Constructing the diameter is less straightforward. Naively, the diameter constraint turns the problem into a Mixed Integer Second Order Conic Program (MISOCP). This destroyed our performance in practice. Instead, we added N constraints that define an N-gon and enforced that every selected point lies inside this N-gon.

The coordinates of a given vertex are represented as (v_x, v_y) in an approximately flat local Euclidean plane. We consider the case where N = 20 thus treating a circle as a

[2] https://pubsonline.informs.org/doi/abs/10.1287/opre.2.4.393

20-gon which should be a reasonable approximation.

Omitting the subtour elimination constraints, which we add lazily, we arrive at the following formulation.

$$
\begin{aligned}
\max_{c \in C} \quad & \text{loglength} - \text{logdiameter} \\
\text{s.t.} \quad & 0 \leq \text{length} \leq l \\
& 0 \leq \text{radius} \leq 2500 \\
& \text{loglength} = \log(\text{length}) \\
& \text{logdiameter} = \log(\text{radius} \times 2) \\
& \forall v \in V \quad (\text{hasvertex}_v \in \{0, 1\}) \\
& \forall e \in E \quad (\text{hasedge}_e \in \{0, 1\}) \\
& \forall v \in V \quad 2 \times \text{hasvertex}_v = \sum_{e=(v,i) \in E} \text{hasedge}_e \\
& \sum_{e \in E} \text{hasedge}_e \times W_e = \text{length} \\
& \text{center}_x, \text{center}_y \in \mathbb{R} \\
& \forall i \in \mathbb{N}, \theta = \frac{i \times 2\pi}{20}, \ v \in V : \quad (v_y - \text{center}_y) \sin(\theta) + (v_x - \text{center}_x) \cos(\theta) \\
& \leq \text{radius} + (1 - \text{hasvertex}_v) \times 2500
\end{aligned}
$$

This formulation allows for finding a collection of cycles with a total length of less than 50 kilometers and a radius of less than 2500 meters. The value 2500 is a "Big M" parameter, which should be tuned based on the problem context. A value greater than the diameter of the town suffices to guarantee correctness.

This formulation, without the diameter and scoring constraints, is "Totally Unimodular", which results in tight relaxations and excellent solver performance. Occasionally, the solver will emit a solution with subtours, which we rule out with an additional constraint generated by a cut-generation subroutine. In practice, we were able to solve for the best-scoring 50k loop in all of Jersey City, with a total length of 50 kilometers and a diameter of approximately 2500 meters.

4. Conclusion and Future Work

So far we have outlined a relatively flexible model for solving the "Long Tiny Loop" problem. Our algorithm scales well in practice [3] and invites some interesting follow-up problems.

[3] https://github.com/rjhunjhunwala/TinyBigLoop

- **Improved dataset preprocessing**: Further refinement of the data cleaning process could lead to more accurate models and faster solutions. High-quality trail datasets are also intrinsically useful to various efforts in the private and public sector to understand outdoor fitness.

- **City planning**: This paper shows that Mixed Integer Linear Programs can scale to the size of relatively large cities. Future work could use similar models to plan city infrastructure decisions.

- **Exploration of commercial solvers**: We plan to explore the capabilities of other commercial solvers such as CPLEX and examine their performance in handling large datasets and complex models. One solver we briefly attempted to use in this investigation was "Hexaly". While Hexaly allowed us to naively port our model using a handful of lines of code, directly handling our major nonlinearity, it struggled to even find a feasible solution.

- **Human route planning**: Investigating why humans are so effective at planning routes could lead to better heuristic approaches to similar problems.

We are excited to see where these explorations could lead and how they may contribute to the broader field of optimization and computational geometry. We are also grateful that this challenge serves to introduce a very diverse audience simultaneously to the disparate fields of "Computational Geometry", "Graph Theory", "Mathematical Optimization", "Cartography", and "Distance Running".

5. Acknowledgements

We would like to take a moment to thank Stuart Geipel [4] for help parsing the unstructured XML data from OpenStreetMap. We also acknowledge CBC [5] which is free software while Gurobi [6] and Hexaly [7] are commercial products that offered academic licenses to help me complete this investigation.

[4] https://github.com/pimlu
[5] https://github.com/coin-or/Cbc
[6] https://www.gurobi.com/
[7] https://www.hexaly.com/