A Model for Developing A Ski Resort with Greedy Strategy

H170

丁子豪　信息与软件工程学院　2015220302019
臧宇航　信息与软件工程学院　2015220302023
邵昊　信息与软件工程学院　2015220302024
（按学号升序，排序不分先后）

2017 年 11 月 20 日

Signature Sheet

Dear Ms. Mogul,

As one of the teams you entrust to help inspect the Wasatch Peaks Ranch, we have identified potential ski slopes and trails on the property in order to develop it to a potential future Winter Olympics location. Our team have figured out a systematic approach trying to develop the Wasatch Peaks Ranch to make it one of top ski resorts among its counterparts in North America.

Provided with a partial list of North American ski resorts with comparison data, and other information available on the web, we carefully inspected all the material and derived huge volume of useful information.

In the first part, we realize data acquisition with respect to points by uniformly doing random sampling on Google Earth to acquire detailed longitude and latitude data, making full use of the topographical information on the map.

Then, we figure out a solution to use the collected coordinates data to produce a path which can be used as ski slope. Subsequently, based on the path we just produce, we have to figure out an approach to produce more paths under specific constraints, trying to meet the entire criteria in a macroscopic perspective.

The key features of our model are as follows:

Skiable Acres	Slope Total (km)	Green (km)	Blue (km)	Black (km)	Width (m)
3348.8	167.443	32.500	62.523	72.420	20

Simple	Intermediate	Difficult
19.41%	37.34%	43.25%

We hope that you will take our suggestion into consideration.

Yours sincerely,
H170 Team

A Model for Developing A Ski Resort with Greedy Strategy

H170

Summary Sheet

Abstract

In this paper, we proposed a systematic approach to develop the Wasatch Peaks Ranch, trying to make it one of top ski resorts among its counterparts in North America.

In the first part, before we start our modeling, we have to acquire the geographical coordinates of the Wasatch Peaks Ranch. The coordinates is extremely vital and play a role of foundation during our process of modeling. We realize data acquisition by uniformly doing random sampling with respect to points on Google Earth to acquire detailed longitude and latitude data, making full use of the topographical information on the map.

Then, we figure out a solution to use the collected coordinates data to produce a path which can be used as ski slope. For the path we just generated, we have to guarantee its quality, such as decent width, moderate slope and angle. Hence, introduction of a judging method is undoubtedly necessary. We build a generative model of multi-trail, which repeatedly use generative model of single-trail. After that, we derive several randomly generated trails which constitute a group of trails, endeavoring to avoid intersections.We roughly obtain a construction plan of the entire ski resort when we build the two models above. And, finally, we analyzed the result of our model, guaranteeing the plan can meet the initial criteria.

Finally, we will analyze the model 1 and model 2, the strengths and weakness of out models. And we compare our final plan with existing ski resorts listed in SkiSlopeComparision.xlsx.

Keywords
Random Sampling Greedy Strategy Generative Model of Ski Trail

A Model for Developing A Ski Resort with Greedy Strategy

1 Problem Restatement

In order to develop it as one of the top ski resorts in North America, we have to identified potential ski slopes and trails on the property, meeting the following criteria:

1. Determine the slopes and trails distribution.
2. Make sure that we construct plenty of trails.
3. Guarantee the distribution of trails of varying difficulty, the most ideal ratio is **beginner : intermediate : difficult = 2 : 4 : 4**. Strictly control the deviation.
4. Guarantee the total length of trails at a minimum requirement of **160 km**.
5. Rank the final outcome of our proposal and endeavor to make it stand out among existing ski resorts in North American.

2 Preliminary Analysis

We have figured out a systematic approach trying to develop the Wasatch Peaks Ranch to make it one of top ski resorts among its counterparts in North America.

Before we start our modeling, we have to acquire the geographical coordinates of the Wasatch Peaks Ranch. The coordinates is extremely vital and play a role of foundation during our process of modeling. Provided with the information available on the website, we have to figure out a fast, effective, accurate way to realize random sampling.

Then, we must figure out a solution to use data of collected coordinates to produce a path which can be used as ski slope. For the path we just generated, we have to guarantee its quality, such as decent width, moderate slope and angle. Hence, introduction of a judging method is undoubtedly necessary.

Subsequently, based on the path we just produce, we have to figure out an approach to produce more paths, trying to meet the entire criteria in a macroscopic perspective. For example, the approach should be able to help our design meet the criteria of total length and distribution of trails in varying difficulty, ratio 2:4:4.

And, furthermore, we have to evaluate our model and make sure that the result is consistent with our expectation.

3 Assumption

In order to alleviate the calculation workload, we would like to put forward a few assumptions after we have inspected the topographic features of the whole ranch. The postulations are as follows:

1. **Ignore** all the influences that **rivers/creeks** and **primary roads** give rise to.

2. All the ski trails running from **starting points** to **destinations** must correspondingly wind their way from **high elevation** to **low elevation**, where the opposite conditions will be given no consideration.

3. All the ski trails are of the same width.

4. A few intersections are allowed.

5. In order to conveniently compute the distance of two points on a trail, we calculate their (two points) **projective distance** and their **difference in altitude** from starting point to destination, assuming the height and horizontal distance are uniformly changing.

6. In order to conveniently compute the distance of two points on a trail, we assume that two points are connected by a **straight line**, where smooth curve (shown in Fig.1.) will be given no consideration.

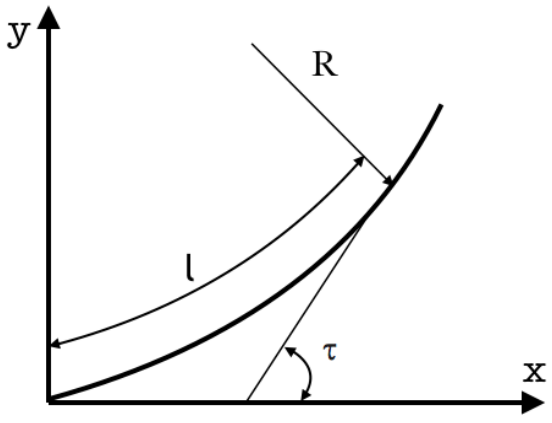

Fig.1. the smooth curve of a road

4 Notation Interpretation

Notation	Interpretation
p	point coordinate, (x, y, h)
v	a vector in three-dimensional space
N	partition the dataset into N groups according to different elevations
Z_i	the point that the path go through on the i-th layer
L	the set of candidate points in the waiting list
dist(p_1, p_2)	distance between two point p_1 and p_2
slope(p_1, p_2)	the slope between two point p_1 and p_2
angle(v_1, v_2)	the intersection angle between vector v_1, and v_2
V	set of vectors,V_i is the vector constituted by the point Z_i and Z_{i+1}
S	set of slope value, S_i = Slope(Z_i, Z_{i+1})
A	set of angles, A_i = Angle(V_i, V_{i+1})
X	candidate of set of trails
Y	set of decisive trails

5 Specification: Modeling and Solution

In this section, we will introduce the approach we use to acquire the geographical coordinates of the Wasatch Peaks Ranch (Data Acquisition).

After we acquire the data, we build a generative model of single-trail at first trial, which can randomly generate a ski trail. In order to examine the quality of this generation, we design a score mechanism to evaluate the trail just be generated. A trail will get a higher score if it is macroscopically smoother (less sharp turns and the turns are relatively gentle).

Subsequently, we build a generative model of multi-trail, which repeatedly use generative model of single-trail. After that, we derive several randomly generated

trails which constitute a group of trails, endeavoring to avoid intersections.

We roughly obtain a construction plan of the entire ski resort when we build the two models above. And, finally, we analyzed the result of our model, guaranteeing the plan can meet the initial criteria.

5.1 Data Acquisition and Preprocess

5.1.1 Data Acquisition

Before we make modeling analysis, we have to acquire the latitude, longitude and altitude of many points in the ranch, to realize the quantization of the map information. We successfully acquire the geographical data of ranch area via Google Earth.

Firstly, we downloaded the file[1] in the appendix and get the coordinates of the resort boundary in Google Earth, which reveal when we visit Google Earth(shown in Fig.2.).

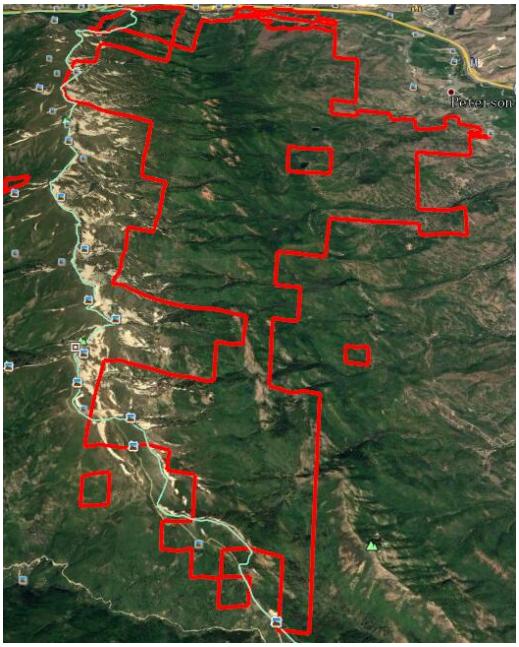

Fig.2. Resort Boundary in Google Earth

Subsequently, we do random sampling on the entire ranch in Google Earth, obtaining the latitude and longitude of the sample points. Furthermore, we use TCX Converter to obtain the corresponding altitude of the sample points. Totally, we acquire 44851 sample points over an area of 13000 acres in the ranch, and output the dataset which are coordinates of three-tuple (x, y, h) for each point. The dataset format is included in the appendix.

The sample points are the foundation of the whole modeling, and will be repeatedly used in modeling computation later.

To visualize the three-dimensional dataset in 3D form, we use the Matplotlib, and the effect taking on is shown in Fig.3..

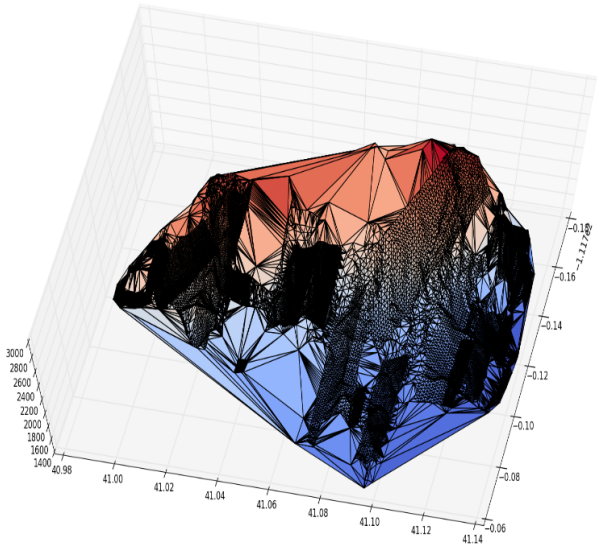

Fig.3. Visualization of Dataset in 3D form by Matplotlib

For further visualization, we use QuickGrid the draw the contours corresponding to the dataset. (shown in Fig.4.)

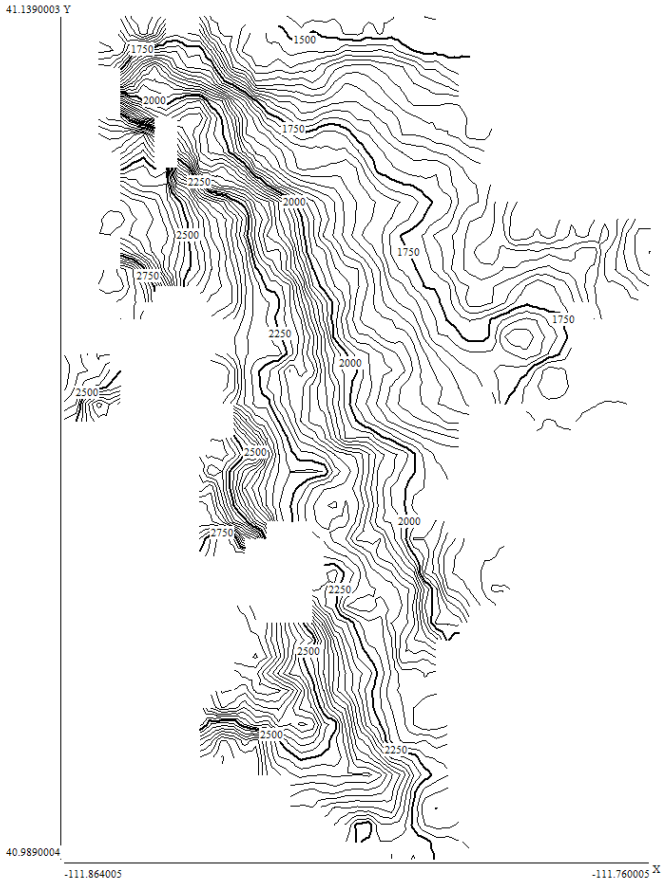

Fig.4. Contours

In Fig.4., we can easily derive the following information:

1. The whole ranch has higher elevation in west and lower elevation in the east, hence, we have a preference to initialize the starting points of ski trails at locations in the northwest.
2. There exist some outparcel and enclaves in the ranch, we will consider these complex conditions later.

5.1.2 Data Preprocess

The data preprocess section mainly functions to unify the unit.

Unit unification: Dataset unit in [1] is foot, while unit in Google Earth is kilometer. We only have area coordinates in [1], while we have latitude and longitude coordinates in [2]. We uniformly mark the coordinates according to the longitude and latitude, and use km as unit for longitude.

5.2 Model 1: Generative Model of Single-trail

According to the second assumption that all the ski trails running from **starting points** to **destinations** must correspondingly wind their way from **high elevation** to **low elevation**, where the opposite conditions will be given no consideration. We analyze part of the above acquired data. Among these sampling points, the one with lowest elevation is **1467.255 km**, the one with highest elevation is **2901.771 km**.

We derive the contour according to the dataset, and we firstly uniformly partition the elevation interval **[1467, 2901]** to N layers, the difference between each two layers is **40 feet**, guaranteeing each data point will be partitioned into a specific layer.

We set **N = 20**, and give the proof in **6.1** that our model is **robust** in that when value of N slightly get changed, the influence on the final result is trivial.

In order to examine the quality of the randomly generated ski trails, we define a loss function, considering the following factors: **slope between two adjacent points**, **the intersection angle of vector constituted by three adjacent points**, the **change rate** of slope and intersection angle.

For those trails whose **slope**, **intersection angle** are **within the reasonable range** and **relatively small change rate**, their corresponding **loss function** values are also lower.

Before we build model, we firstly define the distance of two points and intersection angle constituted by two vectors in three-dimensional space:

For each sample data point, it can be uniquely identified by a 3-tuple coordinates (x, y, h). Assume that for arbitrary two points $\mathbf{P_1}$ $\mathbf{(x_1, y_1, h_1)}$ and $\mathbf{P_2}$ $\mathbf{(x_2, y_2, h_2)}$.

The slope between two points can be defined as **formula 1**:

$$\frac{h_2 - h_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

Meanwhile, we can compute the distance between the two points, which is defined as **formula 2**:

$$dist(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (h_2 - h_1)^2}$$

Compute the horizontal distance between the two points, which is defined as

formula 3:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

For arbitrary three data points P1, P2, P3, we can derive two vector if from P2, which are defined as **formula 4**:

$$\mathbf{a} = (x_2 - x_1, y_2 - y_1, h_2 - h_1) = (a_1, a_2, a_3)$$

$$\mathbf{b} = (x_3 - x_2, y_3 - y_2, h_3 - h_2) = (b_1, b_2, b_3)$$

We can subsequently compute the intersection angle constituted by two vectors, shown in **formula 5**:

$$\theta = arccos(\frac{a_1 b_1 + a_2 b_2 + a_3 b_3}{\sqrt{a_1^2 + a_2^2 + a_3^2} \cdot \sqrt{b_1^2 + b_2^2 + b_3^2}})$$

The generative model of single-trail can be converted to: select some random sampling points in different layers, connect the selected points together, and finally derive a path which go over across the contours from high elevation to low elevation. The algorithm 1 can be expressed in following steps:

***Step 1**: Randomly select a layer from layer **[0, N-5]**, and mark it as **i-th** layer. Randomly select a point on this layer, and mark it as $\mathbf{Z_i}$ point, meaning start from point $\mathbf{Z_i}$ in **i-th** layer.*
***Step 2**: Add all the points in layer **i + 1**, and layer **i + 2** to the waiting list $\mathbf{L_1}$.*
***Step 3**: Successively traverse the points in the waiting list $\mathbf{L_1}$, mark the traversed points as point **P**, compute the horizontal distance between point **P** and point $\mathbf{Z_i}$ according to **formula 3**. If the horizontal distance exceeds the **threshold**, give it up, else add it in waiting list $\mathbf{L_2}$. Finally we get a new waiting list $\mathbf{L_2}$.*
***Step 4**: For each point in $\mathbf{L_2}$, successively compute the slope between **them** and selected points in last layer.*
***Step 5**: For each point in $\mathbf{L_2}$, select a point at different possibility (simple: intermediate: difficult = 0.667 : 0.167 : 0.164), the possibility depends on the range of slope that the point is within, mark the selected point as $\mathbf{Z_{i+1}}$, add it to the set of selected points. The set is to contain the points $\mathbf{Z_{i+1}}$ in **i+1-th** layer that the paths would possibly go through.*
***Step 6**: For selected points $\mathbf{Z_{i-1}}$, $\mathbf{Z_i}$, $\mathbf{Z_{i+1}}$, compute the intersection angle constituted by the three points. If the angle is less than **45** degree, discard $\mathbf{Z_{i+1}}$, goto Step 4 to reselect point **P**.*
***Step 7**: Set the point $\mathbf{Z_{i+1}}$ as starting point, let **i = i + 1**, goto step 2, repeat the process until all the layers have been selected points, or no available point in current layer, return the path.*

We again use the approach of random sampling, repeated the algorithm 1, and derive several paths. For each derived path, consider the slope between two points, and the intersection angle constituted by the three points, we define a score function. A path is more likely to be adopted if its score function value is higher (which means the path is more reasonable). The score function can be express as **formula 6**:

$$l = \sum_{i=1}^{n-1} [\lambda_1 (\max(|A_{i+1} - A_i|, 120)^2 + \lambda_2 (\max(|S_{i+1} - S_i|, 0.2)^2]$$

5.3 Model 2: Generative Model of Multi-trail

When we call model 1 once, we can get one random path. The model 2, however, is to generate many random paths over the ranch so that it finish the entire ski resort's skiable area design. The model 2 must meet the criteria of total length and distribution of trails in varying difficulty, ratio 2:4:4.

After we call model 1 and generate a path, we can compute the set of intersection angles **A**, the set of path points **Z**, the set of slope values **S**. Furthermore, for this path, we compute its maximum slope value. If the maximum slope value does not exceed the limit, we can classify it into beginner, intermediate, hard (three difficulty modes).

We have several plans for the combination of many trail, so we also have to consider the corresponding loss function (partial optimization object, to minimize it) of each plan. We give the following definition on loss function: the value of loss function gets higher if the trails have more intersections, the ratio of trails difficulty among **beginner : intermediate : difficult** are best at 2:4:4 or the value of loss function gets higher if the deviation is more conspicuous.

According to ideas above, for a set of trails which contain several trails, its corresponding loss function can be expressed as formula 7:

$$l = \sum_{i=1}^{n-1} \lambda C_i + (k_1 - 0.2)^2 + (k_2 - 0.4)^2 + (k_3 - 0.4)^2$$

$$k_1 = \sum_{i=1}^{n} 1(0.6 < S_i < 0.25)/n$$

$$k_2 = \sum_{i=1}^{n} 1(0.25 < S_i < 0.4)/n$$

$$k_3 = \sum_{i=1}^{n} 1(0.4 < S_i)/n$$

The algorithm of model 2 is:

***Step 1**: Randomly generate 200,000 paths, derive the set of candidate paths **X**, **X_i** denotes the **i-th** path. The set **Y** contains all the paths in the final path construction plan, initialized as empty set.*
***Step 2**: Randomly select a path in **X** and add it to **Y**.*

***Step 3**: Traverse the candidate paths in path set **X** which have not been added to path set Y, compute the score function value according to formula 6 if the path is added to set **Y**. Finally select a path that has the greatest score function value to set **Y**. The algorithm to compute number of intersections of paths are included in appendix 2.*
***Step 4**: Repeat step 2, until it meet the criteria of total trail length or is hard to find a new path that can bring about positive score.*
***Step 5**: Compute the loss function value of set **Y** according to the formula 7.*

5.4 Analysis of Model Effectiveness

We run the model 1 and model 2, and get a final path plan of the ski resort, it is shown in Fig.5.

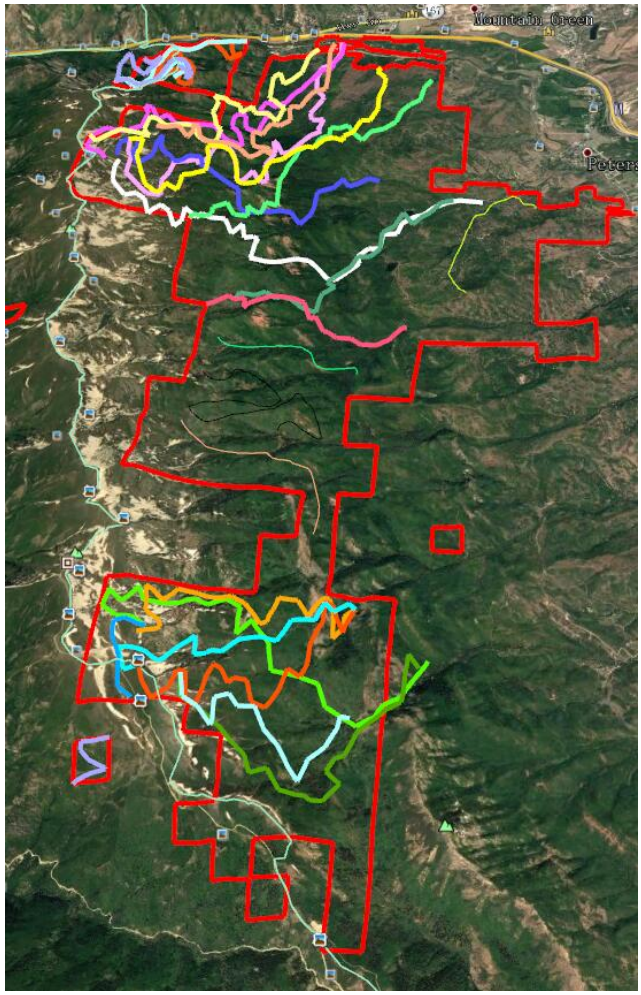

Fig.5. Satellite View after modeling

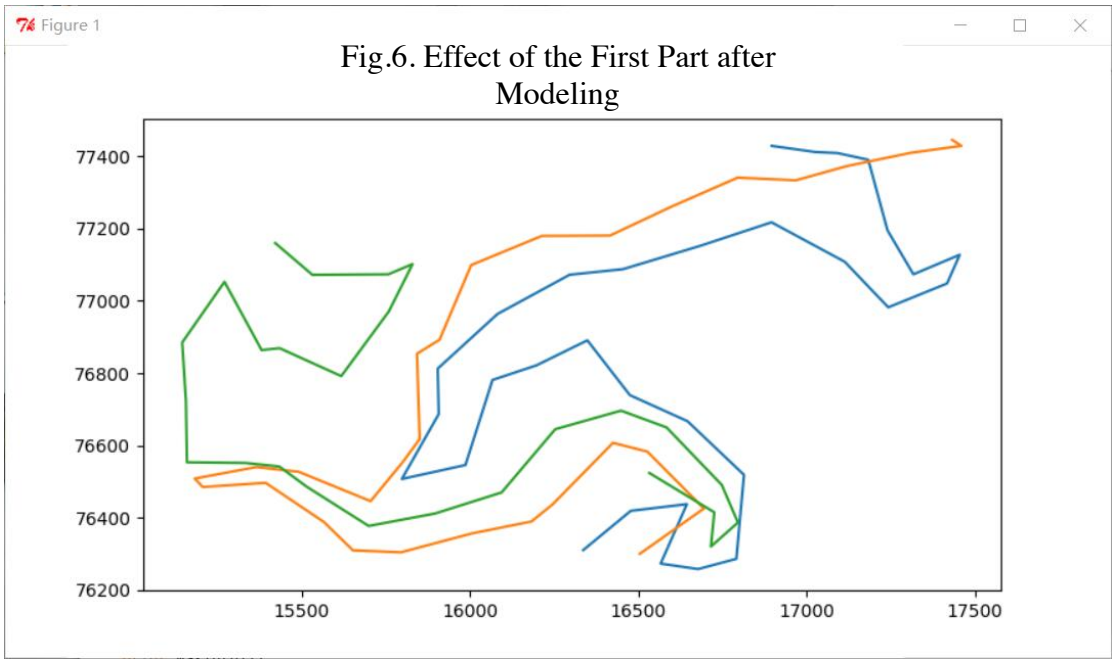

Fig.6. Effect of the First Part after Modeling

6320.91968208
5808.53045638
5029.14660685

Total 24.4 km

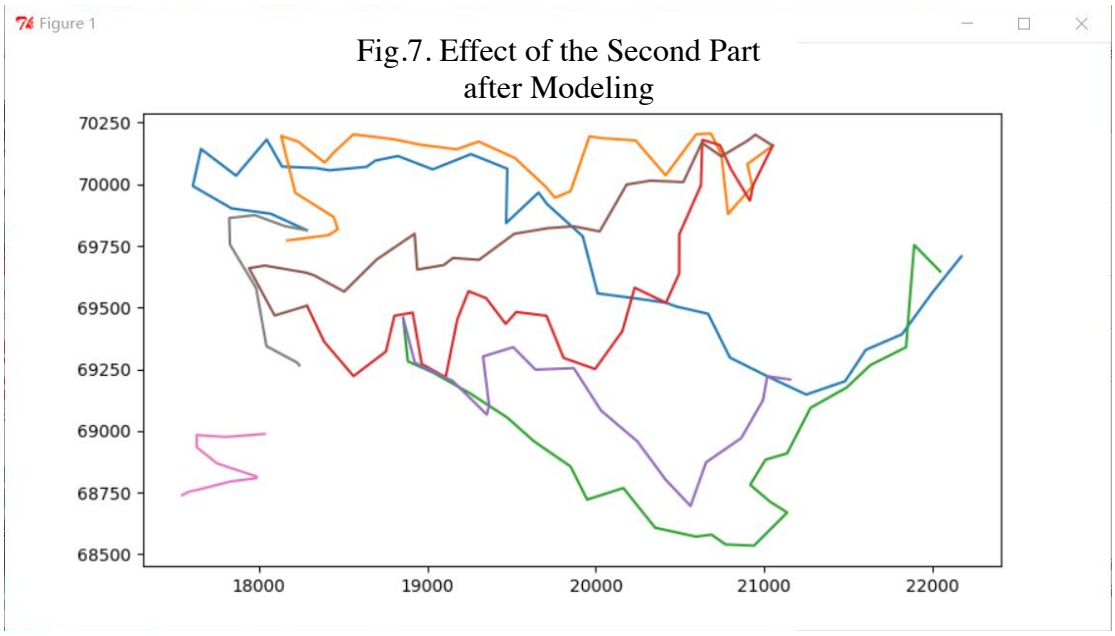

Fig.7. Effect of the Second Part after Modeling

8347.76594265
6085.63204204
6177.66856022
5997.0777292
4297.4517025
5324.91898318
1724.47136879
1663.0354622

48 km

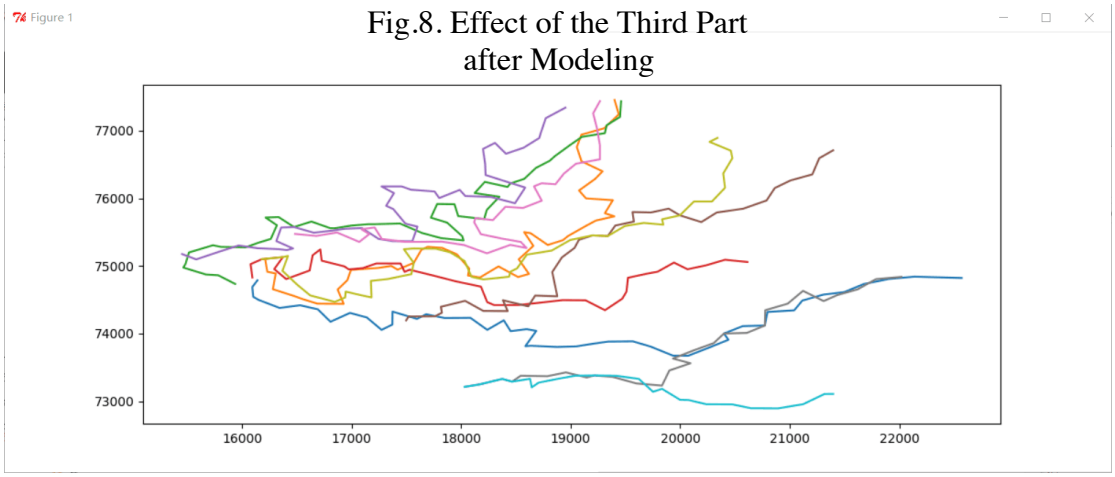

Fig.8. Effect of the Third Part after Modeling

11962.522273
11349.524976
11178.5762016
9063.12819023
10521.1991481
8438.50304166
8386.11322515
7402.734987
10341.3157574
5321.81161427

105 km

6 Model Evaluation

In 6.1, we will analyze the model 1 and model 2, the parameter change's impact on model will be given consideration.

In 6.2, we compare our final plan with existing ski resorts listed in SkiSlopeComparision.xlsx.

In 6.3, we discuss the strengths and weaknesses of our model.

6.1 Robustness of Our Model

Although our ski slopes are randomly generated, we apply a series of systematic constraints to guarantee it meet the key criteria. We firstly defined a score function to examine the quality of a single ski trail, the less slope between key points, the smaller change in angle, the higher in value of score function. Hence, our model theoretically has decent stability.

In practice, when the parameter changes within ±10%, the value of score function changes within ±8%, proving the robustness of our model.

6.2 Comparison with Other Ski Resorts

We compare the scheme we designed with the other ski slopes in SkiSlopeComparison.xlsx with following metrics:

$$\frac{SkiableAcres}{PeakElev - BaseElev} + (P_{simp} - 0.2)^2 + (P_{interm} - 0.4)^2 + (P_{diffi} - 0.4)^2$$

According to the comparison index, the result is as follows:

Name	State	Country	Score
Vail	Colorado	USA	50.6008
Silver Star	British Columbia	Canada	41.0472
Wasatch Peaks Ranch	**Utah**	**USA**	32.6499
Breckenridge	Colorado	USA	28.466
Lake Louis	Alberta	Canada	28.0054
Squaw Valley	California	USA	26.3793
Big Sky Resort	Montana	USA	17.1486
Jackson Hole	Wyoming	USA	14.0467
Beaver Creek	Colorado	USA	13.4581
Park City Mountain	Utah	USA	11.7391
Sugarloaf Mountain	Maine	USA	7.26617
Whistler Blackomb	British Columbia	Canada	5.0981
Breckenridge	British Columbia	Canada	2.99824
Killington	Vermont	USA	2.15294
Sun Peaks	British Columbia	Canada	0.64885
Winter Park Resort	Colorado	USA	-7.0965
Steamboat Springs	Colorado	USA	-23

We designed the Wasatch Peaks Ranch Ski Resort program ranked No. 3, indicating the rationality of our design.

6.3 Strengths and Weaknesses

Strengths:

(1) **Decent robustness**: our model is of high stability, the impact on the result will be trivial when the parameters slightly get changed.

(2) **The reasonable distribution of slopes over elevations**: we have develop a reasonable algorithm to select key points on contours when generate trails.

(3) **Optimization**: We have constructed the score function and loss function to find approximate optimal solution from huge space of candidate solution.

(4) **User-friendly**: Out model has inspected several perspectives including angle change and slope change, making trails to be gentle and user-friendly.

(5) **Quantization**: We uniformly do random points sampling on Google Earth to acquire detailed longitude and latitude data, making full use of the topographical information on the map.

Weaknesses:

(1) **Ideal assumption**: we ignored the effect of rivers/creeks and primary roads during the construction.

7 Reference

Prop-Map-for-Wasatch-Peaks-Ranch

8 Appendix

8.1 Format of GPS Coordinates

	A	B	C
1	LONG	LAT	ALT
2	-111.836	41.13689	1587.474
3	-111.85	41.13661	1576.055
4	-111.85	41.13282	1759.664
5	-111.855	41.13276	1634.326
6	-111.855	41.11822	2378.56

8.2 Fundamental Code for Experiment

```
import math
import pickle
import collections
import numpy
import random
import time
import traceback
from multiprocessing import Pool

import matplotlib.pyplot as plt

beginnerSlope = 0.06
intermediateSlope = 0.25
difficultSlopes = 0.4

class Point():
    def __init__(self, x, y, h, _id):
        self.x = x
        self.y = y
        self.h = h
        self._id = _id
class Layer():
    def __init__(self, h, points=[]):
        self.h = h
        self.points = points
        self.pointsNum = 1
class Solution():
    def __init__(self, path, degs, slopes, distance):
        self.path = path
        self.degs = degs
        self.slopes = slopes
        self.distance = distance

def computeslopes(point1, point2):
    return abs(point2.h - point1.h) / (math.sqrt((point1.x - point2.x) ** 2
+  (point1.y - point2.y) ** 2))

def computeDistance(point1, point2):
    return math.sqrt((point1.x - point2.x) ** 2 + (point1.y - point2.y) ** 2
+ (point1.h - point2.h) ** 2)

def computeHorizontalDistance(point1,point2):
    return math.sqrt((point1.x - point2.x) ** 2 + (point1.y - point2.y) **
2)

def computeDegree(point1, point2, point3):
    a = [point1.x - point2.x, point1.y - point2.y, point1.h - point2.h]
    b = [point3.x - point2.x, point3.y - point2.y, point3.h - point2.h]
    vecProduct = a[0] * b[0] + a[1] * b[1] + a[2] * b[2]
    norm_a = math.sqrt(a[0] ** 2 + a[1] ** 2 + a[2] ** 2)
    norm_b = math.sqrt(b[0] ** 2 + b[1] ** 2 + b[2] ** 2)
    return math.acos(min(1, max(-1, (vecProduct / (norm_a * norm_b))))) *
180 / math.pi

def intersection(line1, line2):
    (x1, y1), (x2, y2) = line1
    (x3, y3), (x4, y4) = line2
   if x3 == x4:
        (x1, y1), (x2, y2) = line2
        (x3, y3), (x4, y4) = line1
```

```
        if x1 == x2:
            if x3 == x4:
                if x1 == x3:
                    y1, y2 = min(y1, y2), max(y1, y2)
                    if (y4 >= y1 and y4 <= y2) or (y3 >= y1 and y3 <= y2):
                        return 1000
                    else:
                        return 0
                return 0
            else:
                x3, x4 = min(x3, x4), max(x3, x4)
                if x1 >= x3 and x1 <= x4:
                    k2 = (y4 - y3) / (x4 - x3)
                    b2 = y4 - k2 * x4
                    y0 = k2 * x1 + b2
                    y1, y2 = min(y1, y2), max(y1, y2)
                    if y0 >= y1 and y0 <= y2:
                        return 1
                    else:
                        return 0
                else:
                    return 0
        k1 = (y2 - y1) / (x2 - x1)
        k2 = (y4 - y3) / (x4 - x3)
        if k1 == k2:
            b1 = k1 * x1 + y1
            b2 = k2 * x2 + y2
            if b1 == b2:
                x1, x2 = min(x1, x2), max(x1, x2)
                if ((x4 >= x1 and x4 <= x2) or (x3 >= x1 and x3 <= x2)):
                    return 1000
                else:
                    return 0
            else:
                return 0
        x0 = (k1 * x2 - k2 * x4 - y2 + y4) / (k1 - k2)
        x1, x2 = min(x1, x2), max(x1, x2)
        x3, x4 = min(x3, x4), max(x3, x4)
        if x1 <= x0 and x0 <= x2 and x3 <= x0 and x0 <= x4:
            return 1
        else:
            return 0


def intersectionOfMany(lines1, lines2):
    num = 0
    for t_line in lines1:
        for tt_line in lines2:
            num += intersection(t_line, tt_line)
    return num

pointsNumForEachHeight = collections.defaultdict(int)
pointsDict = collections.defaultdict(list)
heightList = []
layers = []


def loadData(filename):
    heightSet = set([])
    with open(filename) as f:
        lines = [x.strip() for x in f.readlines()]
```

```
          for line in lines:
            points = [x.strip() for x in line.split()]
            trueheight = float(points[0])
            height = int(trueheight) / 20 * 20
            heightSet.add(height)
            for point in points[1:]:
                x, y = [float(x) for x in point.split(',')]
                newpoint = Point(x, y, trueheight,
pointsNumForEachHeight[height])
                pointsNumForEachHeight[height] += 1
                pointsDict[height].append(newpoint)
    for h in heightSet:
        heightList.append(h)
    heightList.sort(reverse=True)
    for h in heightList:
        layers.append(Layer(h, pointsDict[h]))

def preProcess():
    pass

visitedPoint = set([])
def generatePath(randomseed=None):
    random.seed(randomseed)
    isStart = 1
    startLayer = random.choice(range(len(layers) - 30))
    lastLayer = startLayer
    lastPoint = random.choice(range(layers[lastLayer].pointsNum))
    path = []
    for i in range(lastLayer):
        path.append(None)
    path.append(lastPoint)
    degrees = []
    slopes = []
    totalDistance = 0
    while lastLayer < len(layers) - 2:
        toBeChoosedPoints = []
        probs = []
        t_slopes = []
        for point in layers[lastLayer + 1].points:
            toBeChoosedPoints.append((lastLayer + 1, point._id))
        for point in layers[lastLayer + 2].points:
            toBeChoosedPoints.append((lastLayer + 2, point._id))
        toBeChoosedPoints = [point for point in toBeChoosedPoints if
computeHorizontalDistance(layers[point[0]].points[point[1]],
layers[lastLayer].points[lastPoint]) < 250]
       if isStart == 1:
           isStart = 0
           for point in toBeChoosedPoints:
                slope = computeslopes(layers[lastLayer].points[lastPoint],
layers[point[0]].points[point[1]])
                t_slopes.append(slope)
                if slope <= intermediateSlope:
                    probs.append(6.66)
                elif slope <= difficultSlopes:
                    probs.append(1.64)
                elif slope <= 0.6:
                    probs.append(1.6)
                else:
                    probs.append(0)
```

```
                probTarget = sum(probs) * random.random()
            if probTarget == 0:
                return Solution(path, degrees, slopes, totalDistance)
            for i, prob in enumerate(probs):
                probTarget -= prob
                if probTarget < 0:
                    if toBeChoosedPoints[i][0] - lastLayer == 1:
                        path.append(toBeChoosedPoints[i][1])
                    else:
                        path.append(None)
                        path.append(toBeChoosedPoints[i][1])
                    totalDistance +=
computeDistance(layers[toBeChoosedPoints[i][0]].points[toBeChoosedPoints[i]
[1]], layers[lastLayer].points[lastPoint])
                    lastLayer = toBeChoosedPoints[i][0]
                    lastPoint = toBeChoosedPoints[i][1]
                    slopes.append(t_slopes[i])
                    break
            continue
        t_degs = []
        for point in toBeChoosedPoints:
            if path[-2] != None:
                deg = computeDegree(layers[len(path) - 2].points[path[-2]],
layers[lastLayer].points[lastPoint], layers[point[0]].points[point[1]])
            else:
                deg = computeDegree(layers[len(path) - 3].points[path[-3]],
layers[lastLayer].points[lastPoint], layers[point[0]].points[point[1]])
            t_degs.append(deg)
            slope = computeslopes(layers[lastLayer].points[lastPoint],
layers[point[0]].points[point[1]])
            t_slopes.append(slope)
            if deg <= 45:
                probs.append(0)
            elif slope <= intermediateSlope:
                probs.append(6.66)
            elif slope <= difficultSlopes:
                probs.append(1.64)
            elif slope <= 0.6:
                probs.append(1.2)
            else:
                probs.append(0)
        probTarget = sum(probs) * random.random()
        if probTarget == 0:
            break
        for i, prob in enumerate(probs):
            probTarget -= prob
            if probTarget < 0:
                if toBeChoosedPoints[i][0] - lastLayer == 1:
                    path.append(toBeChoosedPoints[i][1])
                else:
                    path.append(None)
                    path.append(toBeChoosedPoints[i][1])
                totalDistance += computeDistance(layers[toBeChoosedPoints[i]
[0]].points[toBeChoosedPoints[i][1]], layers[lastLayer].points[lastPoint])
                lastLayer = toBeChoosedPoints[i][0]
                lastPoint = toBeChoosedPoints[i][1]
                degrees.append(t_degs[i])
                slopes.append(t_slopes[i])
                break
    return Solution(path, degrees, slopes, totalDistance)
def score(solution):
    s = solution.distance
    for deg in solution.degs:
        s -= 0.5 * max(0, 120 - deg) ** 2
    return s
```

```
def showGraph(graph, savePath=None):
    for s in graph:
        x = []
        y = []
        for i, p_id in enumerate(s.path):
            if p_id == None:
                continue
            x.append(layers[i].points[p_id].x)
            y.append(layers[i].points[p_id].y)
        plt.plot(x,y)
    if savePath:
        plt.savefig(savePath)
    else:
        plt.show()
    plt.clf()

def showPath(solution):
    x = []
    y = []
    for i, p_id in enumerate(solution.path):
        if p_id == None:
            continue
        print '---->', layers[i].h, layers[i].points[p_id].x / 1000 * 94,
layers[i].points[p_id].y / 1000 * 77
        x.append(layers[i].points[p_id].x)
        y.append(layers[i].points[p_id].y)
    plt.plot(x, y)
    plt.show()


def trainSave(filename):
    loadData(filename)
    solutions = []
    for i in range(10000):
        if i %100 == 0: print i
        try:
            solutions.append(generatePath(int(time.time()) + i))
        except Exception as e:
            print e
    pickle.dump(solutions, open(filename.split('.')[0] + 'solutions.pkl',
'wb'))

def t_generatePath(seed):
    try:
        if not layers: loadData('center.txt')
        return generatePath(seed)
    except Exception as e:
        print e
        traceback.print_exc()

def trainSaveF(filename):
    loadData(filename)
    solutions = []
    tasks = [i + int(time.time()) for i in range(200000)]
    pool = Pool(8)
    solutions = pool.map(t_generatePath, tasks)
    pool.close()
    pool.join()
    solutions = [x for x in solutions if x is not None]
    print solutions[0].path
    pickle.dump(solutions, open(filename.split('.')[0] + 'solutions.pkl',
'wb'))
```

```
def path2coor(path):
    lines = []
    for i in range(len(path) - 1):
        if path[i] is None:
            continue
        if path[i + 1] is not None:
            j = i + 1
        else:
            j = i + 2
        lines.append(((layers[i].points[path[i]].x,
layers[i].points[path[i]].y),
(layers[j].points[path[j]].x ,layers[j].points[path[j]].y)))
    return lines

def generateGraph(solutions, randomseed = 0, targetDistance = 20000):
    random.seed(randomseed)
    linesPool = []
    graph = []
    choosedSolutions = set([])
    totScore = 0
    currDistance = 0
    ix = random.randint(1,100)
    linesPool.extend(path2coor(solutions[ix].path))
    currDistance += solutions[ix].distance
    totScore += solutions[ix].distance
    while currDistance <= targetDistance:
        maxscore = -1
        maxi = 0
        for i in range(ix, ix+8000):
            if i in choosedSolutions:
                continue

            score = solutions[i].distance - 150 *
intersectionOfMany(linesPool, path2coor(solutions[i].path))
            if score > maxscore:
                maxscore = score
                maxi = i
        print maxscore
        if maxscore <= 0:
            print 'Warning'
            return graph, totScore, currDistance
        else:
            choosedSolutions.add(maxi)
            print maxi, solutions[maxi].distance,
intersectionOfMany(linesPool, path2coor(solutions[i].path))
            currDistance += solutions[maxi].distance
            totScore += maxscore
            graph.append(solutions[maxi])
            ix = ix + random.randint(1,2000)
            linesPool.extend(path2coor(solutions[maxi].path))
    return graph, totScore, currDistance

def main():
    loadData('center.txt')
    solutions = pickle.load(open('centersolutions.pkl', 'rb'))
    for solution in solutions:
        solution.score = score(solution)
    solutions.sort(key = lambda x: x.score, reverse = True)
    pickle.dump(solutions, open('centersolutions.pkl', 'wb'))
    maxInfo = None
```

```
    infoList = []
    for i in range(10):
        seed = i + int(time.time())
        g, s, dis = generateGraph(solutions, seed, 100000)
        infoList.append((g, s, dis))
        print i, seed, i, s, dis
        showGraph(g, str(seed) + '.png')
        pickle.dump(g, open(str(seed)+'.pkl', 'wb'))
        if s > maxscore:
            maxscore = score
            maxInfo = (g, s, dis)
    pickle.dump(infoList, open('result.pkl','wb'))
    print maxInfo[2]
    print maxInfo[1]
    showGraph(maxInfo[0])

if __name__ == '__main__':
    main()
```