CS221: Trip Planner Final Report

December 2013

1 Introduction

Visiting a new city. Going on a vacation. All of these require careful planning. And, planning a trip is often a lengthy and headache-inducing task. This is because our subconscious mind needs to process a large volume of information–checking feasibility of a certain set of events, considering other possibilities that might be *better* in some sense than others and revisiting what these *better* conditions mean. To ease up the pain, we employed artificial intelligence (AI) techniques from CS221 lessons to create a trip planner. Unfortunately, we didn't have the time to implement all the neat ideas we had, so we will document what we tried over the past few weeks and discuss remaining ideas as future work.

2 Task Definition

The trip planning problem can be formulated as follows:

$$\begin{aligned}\textsc{TripPlanner} = \text{“On input } &\langle \textsc{EventQuery}_{1:n}, \textsc{Database}_{1:m}, \mathcal{I}, f\rangle,\\ &\text{1. Return optimal schedule } \textsc{Itinerary}^*_{1:p} \subset \textsc{Database}_{1:m} \times \mathcal{I}\\ &\text{according to objective } f,\text{”}\end{aligned}$$

where

$$\begin{aligned}\textsc{EventQuery}_i = \{&K_i, && \text{(Query keyword)}\\ &T_i \in \{A, R, H\}, && \text{(Types } A\text{, } R \text{ and } H \text{ correspond to activity, restaurant and hotel)}\\ &w_i \in [1, 5] \cup \{5+\}, && \text{(degree of preference; 5+ means it must be selected)}\\ &\Delta t_i, && \text{(desired duration of event)}\\ &\mathcal{I}_i = [t_i^s, t_i^e], && \text{(time interval during which event } i \text{ should be scheduled)}\\ &L_i && \text{(optional location where events should preferrably take place)}\}\end{aligned}$$

and

$$\begin{aligned}\textsc{Database}_i = \{&N_i, && \text{(name of the event)}\\ &r_i \in [1, 5] && \text{(event rating)}\\ &L_i, && \text{(location of the event)}\\ &\{\Delta t_{ij}\} && \text{(adjacency list of travel time to other events)}\}.\end{aligned}$$

In this project, we have limited Database to businesses searchable on Yelp and used the Google API to populate the adjacency list. As will be discussed in Section 3, we have obtained results with approximation $\widehat{\textsc{Database}}$.

3 Approaches

There are several ways of formulating the trip planning problem as a classical AI problem. Since the goal of the trip planning task is to generate a trip proposal under specific user-defined conditions, one natural modeling option is the constraint satisfaction problem. The specifics of our design will be described in Section 3.3.

During this design process, one important challenge that we were faced with, was the estimation of transit time between two locations in a trip. This transit time estimation was made using Google's Distance Matrix API[1], which has a daily limit on the number of queries. These queries are also rate-limited so if we relied on this to make actual predictions, we would have to wait for a long time just to construct problem instances. To cope with this issue, we decided, on one hand, to limit the scope of our project in planning a trip for a user in a fixed city: Toronto, Ontario, Canada. On the other hand, we chose to approximate transit time instead of querying Google for the exact values. This capability was important as each event request results in many candidates from Yelp and the number of pairwise transportation time that we need to make quickly outgrows the 2500-query limit per day. The details of this implementation are laid out in Section 3.1.

3.1 Time estimation as a search problem

Because each IP address has a limited number of queries it can make from Google, it is necessary to make a reasonable estimate of transportation time offline. This was achieved by representing a city as a two-dimensional array of points, and predicting distances based on known pairwise distances between some nodes. Ideally, it would be best to sample the travel time of all possible pairs, but this would amount to $\binom{mn}{2} = O((mn)^2)$ queries where m and n are dimensions of the grid. With this approach, the size of mesh one can learn each day from Google is smaller than 10×10. Therefore, having a good coverage of a large city in a limited time span is difficult in practice. In this work, distances to 8-neighbors of each mesh point are collected instead and the uniform cost search is performed over the denser mesh. The search problem is formulated as follows:

- $s_{\text{start}} = (x_{\text{start}}, y_{\text{start}})$ where (x, y) is a satellite coordinate operated with an affine transformation to normalize to some bounding box.
- $\text{IsGoal}(s) = 1\{s = (x_{\text{goal}}, y_{\text{goal}})\}$
- $\text{Actions}(s) = \{\text{north-west}, \text{north}, \text{north-east}, \text{east}, \text{south-east}, \text{south}, \text{south-west}, \text{west}\}$
- $\text{Succ}(s, a) = (\text{new location by following direction } a)$
- $\text{Cost}(s, a) = \begin{cases} t_{\text{ground-truth}}(s, \text{Succ(s,a)}) & \text{if Succ}(s, a) \text{ is within the mesh} \\ \infty & \text{otherwise} \end{cases}$,

where $t_{\text{ground-truth}}$ is the time obtained by the Google API. Because each query does not need to compare itself with the rest, the number of queries made is strictly bounded by $8mn = O(mn)$. It is now possible to generate a much denser representation of a city of interest.

One problem with the mesh model is that it does not allow time estimation within a mesh cell. The prediction of intro-cellular distance is posed as a linear regression problem. Specifically let (x, y) be the coordinates of a location in a mesh cell specified by the four points $\{ (x_1, y_1) ; (x_2, y_2) ; (x_3, y_3) ; (x_4, y_4) \}$. Then we approximate the distance $d((x, y), (x_c, y_c))$ between (x, y) and its closest mesh point (x_c, y_c) by :

$$\hat{d}((x, y), (x_c, y_c)) = (1 \;\; x_c \;\; y_c \;\; x \;\; y) \cdot \theta$$

where θ is the least squares solution to the equation

[1] https://developers.google.com/maps/documentation/distancematrix/

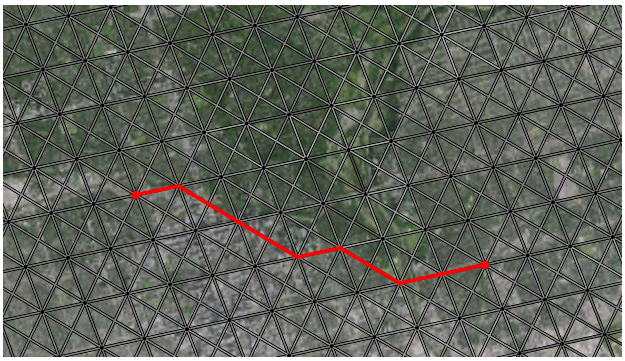


Figure 1: Distance approximation between two mesh points is casted as a search and linear regression problem in an 8-connected mesh grid.

$$Z\theta = d \quad \text{with} \quad Z = \begin{pmatrix} 1 & x_1 & y_1 & x_2 & y_2 \\ 1 & x_2 & y_2 & x_1 & y_1 \\ 1 & x_1 & y_1 & x_3 & y_3 \\ 1 & x_3 & y_3 & x_1 & y_1 \\ 1 & x_1 & y_1 & x_4 & y_4 \\ 1 & x_4 & y_4 & x_1 & y_1 \\ 1 & x_2 & y_2 & x_3 & y_3 \\ 1 & x_3 & y_3 & x_2 & y_2 \\ 1 & x_2 & y_2 & x_4 & y_4 \\ 1 & x_4 & y_4 & x_2 & y_2 \\ 1 & x_3 & y_3 & x_4 & y_4 \\ 1 & x_4 & y_4 & x_3 & y_3 \end{pmatrix} \quad \text{and} \quad d = \begin{pmatrix} d\left((x_1, y_1), (x_2, y_2)\right) \\ d\left((x_2, y_2), (x_1, y_1)\right) \\ d\left((x_1, y_1), (x_3, y_3)\right) \\ d\left((x_3, y_3), (x_1, y_1)\right) \\ d\left((x_1, y_1), (x_4, y_4)\right) \\ d\left((x_4, y_4), (x_1, y_1)\right) \\ d\left((x_2, y_2), (x_3, y_3)\right) \\ d\left((x_3, y_3), (x_2, y_2)\right) \\ d\left((x_2, y_2), (x_4, y_4)\right) \\ d\left((x_4, y_4), (x_2, y_2)\right) \\ d\left((x_3, y_3), (x_4, y_4)\right) \\ d\left((x_4, y_4), (x_3, y_3)\right) \end{pmatrix}. \tag{1}$$

3.2 Trip Planning as an interval scheduling problem

Our baseline system takes travel time into consideration, but assumes that all activities have uniform weights. With this relaxation, we were able to formulate a simple greedy algorithm. Specifically, based on the idea of the Earliest Deadline First (EDF) scheduling algorithm [3], the events with the earliest deadline (t_i^e) and the highest score ($r_i w_i - \Delta t_{ji}$) are scheduled first.

3.3 Trip planning as a constraint satisfaction problem

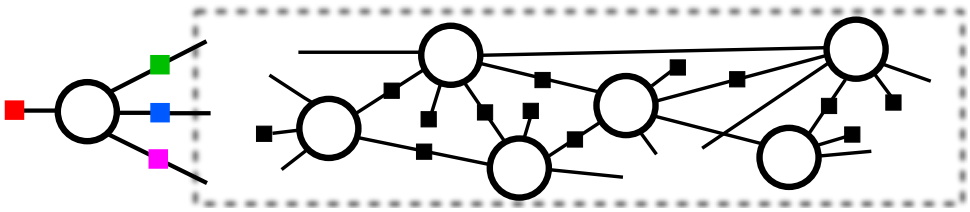
We reduced the TripPlanner problem instance to a factor graph where obtaining a valid assignment on this graph corresponds to a valid itinerary according to f in our task definition. Thus, in effect, optimizing the objective f is now finding the maximum weight assignment which can be performed by a backtracking search as covered in lecture.

The factor graph reduction consists of variables whose i^{th} domain is defined as follows:

$$\textsc{PrevEvent}_i \times \textsc{ValidInterval}_i = (\textsc{Database}_{\mathcal{M}_i} \cup \{\textsc{NotAssigned}, \textsc{FirstAssigned}\}) \times (\mathcal{I}_i \otimes \Delta t_i),$$

where $\mathcal{M}_i \subset \{1, \ldots, m\}$ maps to a set of events related to an $\textsc{EventQuery}_i$ and $\otimes$ returns all subintervals in $\mathcal{I}_i$ whose duration is Δt_i. We will interchangeably use PrevEvent to denote a working domain and also the actual assignment value when acted as an operator.

The idea behind this design is to assign a time interval and also what the previous event should be. This is necessary to enforce an ordering in the schedule and is implemented by binary potential 2 below. Originally, we attempted to enforce ordering by adding all possible pairs of activities as variables and a unary potential with a binary value to indicate whether this pair ordering is valid. However, we found that this led to a search space of $O(2^{n^2}) \supset O(2^{n \log_2 n}) \supset O(n^n)$; this is worse than the final formulation with a search space, $O(n^n)$.



Essentially, there are three types of potentials added to the graph to describe our TripPlanner instance.

1. Each event EventQuery$_i$ is given a unary potential

$$u\left(\text{EventQuery}_i\right) = \begin{cases} 0 & \text{if } [w_i = 5+] \wedge \neg \text{IsScheduled}(\text{EventQuery}_i) \\ 1 & \text{if } [w_i \neq 5+] \wedge \neg \text{IsScheduled}(\text{EventQuery}_i) \\ \exp\left\{2\frac{\frac{w_i r_i}{\sqrt{(w_i^2+r_i^2)/2}}-1}{}\right\} & \text{otherwise} \end{cases}$$

   whose role is to quantify the suitability of the activity for the user, given his preferences. The rationale behind this metric is to penalize activities with a high user-preference but with a low Yelp-rating or vice-versa when compared to activities with both average user-preference and Yelp-rating.

2. Between each pair of variables, a binary potential is added to evaluate whether ordering is enforced and, if so, the actual travel and wait time scores. This is performed by a series of checks.

   (a) If PrevEvent(e_2) = NotSelected, return 1 (e_2 is not scheduled; there is no need to enforce anything.)

   (b) If PrevEvent(e_2) = PrevEvent(e_1), return 0 (two events can't be scheduled simultaneously after some event.)

   (c) If PrevEvent(e_2) = FirstSelected, return 1 (the first event's assignment is a don't-care value)

   (d) If PrevEvent(e_2) $\neq e_1$, return 1 (e_2 doesn't back-point to e_1; no need to enforce anything.)

   (e) If PrevEvent(e_1) = NotSelected, return 0 (at this point, we know e_2 backpoints to e_1. e_1 should have been selected.)

   (f) If this step is reached, e_1 precedes e_2 and their ordering is valid. We now check whether the start time of the second event is at least the end time of the first event plus the travel time between the two. If not, we return 0. Otherwise, the binary potential evaluates to $\exp(-\log(\text{travel time}) + -\log(\text{wait time}))$. The actual time score is given by $-\log(t)$. This was to penalize a difference in the order of magnitude rather than small variations.

3. To ensure that one and only one restaurant is scheduled during a particular meal time, we use two n_{rl}-ary and n_{rd}-ary potentials where n_{rl} and n_{rd} are the numbers of restaurants that could potentially be scheduled for lunch and dinner respectively. The same issue for the hotels is solved using an n_h-ary potential with n_h being the number of hotels.

4 Results and Analysis

It was found that the uniform-cost search and linear regression model for intra-cellular time approximations overestimate the groundtruth value by a factor of about 1.25 (Erratum. It was mentioned to be 2 during poster presentation. This was approximated, but with more extensive data collection and analysis, 1.25 is a more representative factor than 2.) Our approximation method performed decently over small distances, but as locations became farther apart, the inability to model highways resulted in a larger error (Figure 2).

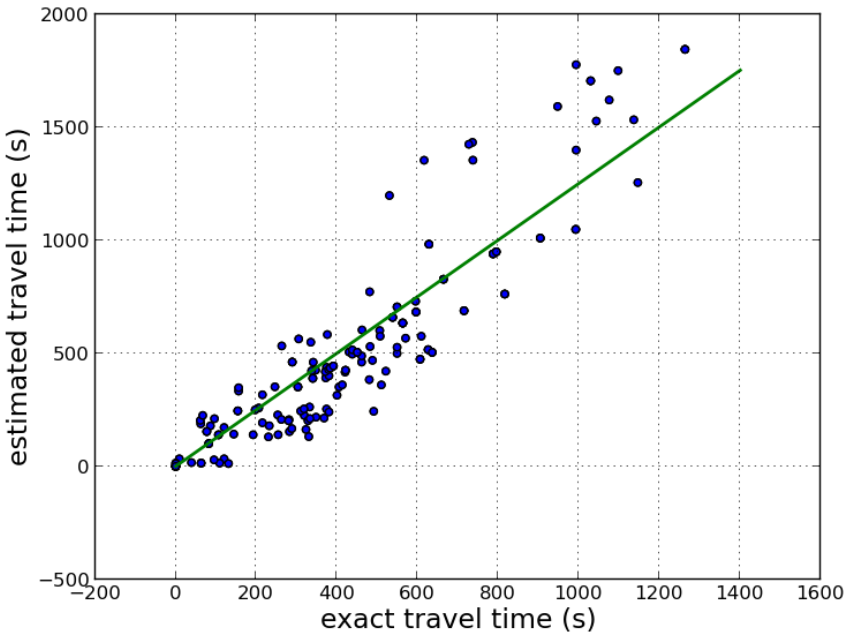


Figure 2: To verify correctness, we sampled numerous pairs of locations over multiple scales and plotted the approximate distances against the ground-truths from Google API.

We solved various trip planning problems, we constructed, with travel time approximations and groundtruth values, and found that the consistency of overestimation has allowed the approximate problem to yield the same set of solutions as the original problem. There was one case where the approximate problem returned an itinerary from one end of the city to the other end, but this was because the business of interest was out of mesh and had an unhandled return value of -1 for the travel time.

As an interesting comparison, we documented in Table 1 the results of our greedy baseline system, which naively chooses an event based on the earliest deadline. Due to its myopic nature, it doesn't realize that events 16-18 can be scheduled in the morning before lunch, while the optimal solution from backtracking search correctly schedules one of these events first. However, the major advantages of the greedy approach are that this runs in $O(n^2)$ time and its time interval domain is not limited to some predetermined time granularity.

The last column in Table 1 illustrates how our formulation of the trip planning problem allows the solver to discard suboptimal solutions. To easily visualize these itineraries, a labeled map is presented in Figure 3 in the appendix. Both the optimal and the sample suboptimal solutions schedule event 17 first. In the suboptimal case, the second activity scheduled is activity number 2 due to its shorter travel time than going to activity 1. The optimal solution (optimal according to a supplied f) instead schedules activity 1 because of the higher rating score. The same reasoning explains why 16 and 20 are scheduled instead of 18 and 22, respectively. Our objective turned out to favor user preference and activity rating scores very strongly, and in many cases overlooked travel time. Although this may not be the best itinerary to many users, our factor graph formulation correctly finds solutions according to the given objective function, as specified in the task

definition.

					Baseline		Optimal		Suboptimal	
Event #	Yelp rating	User pref.	Keyword	Time req.[1]	Order	Start time	Order	Start time	Order	Start time
1	3.50	4.00	Lunch	[11, 13], 1	1	11	2	12	-	-
2	2.50	4.00	Lunch	[11, 13], 1	-	-	-	-	2	12
3	2.50	4.00	Lunch	[11, 13], 1	-	-	-	-	-	-
4	2.50	4.00	Lunch	[11, 13], 1	-	-	-	-	-	-
5	3.00	4.00	Lunch	[11, 13], 1	-	-	-	-	-	-
6	2.50	4.00	Lunch	[11, 13], 1	-	-	-	-	-	-
7	4.00	4.50	Dinner	[18, 19], 1	-	-	4	18	-	-
8	4.00	4.50	Dinner	[18, 19], 1	-	-	-	-	-	-
9	4.00	4.50	Dinner	[18, 19], 1	-	-	-	-	4	18
10	3.50	4.50	Dinner	[18, 19], 1	-	-	-	-	-	-
11	2.50	4.50	Dinner	[18, 19], 1	3	18	-	-	-	-
12	3.50	4.50	Dinner	[18, 19], 1	-	-	-	-	-	-
13	3.50	5.00	Theater	[15, 22], 2	-	-	-	-	-	-
14	3.50	5.00	Theater	[15, 22], 2	4	19.27	-	-	-	-
15	4.00	5.00	Theater	[15, 22], 2	-	-	5	20	5	20
16	5.00	5+	Shopping	[8, 19], 3	2	12.24	3	14	-	-
17	5.00	5+	Shopping	[8, 19], 3	-	-	1	8	1	8
18	3.50	5+	Shopping	[8, 19], 3	-	-	-	-	3	13
19	4.00	5+	Karaoke	[21, 28], 2	5	21.52	6	23	6	23
20	4.00	5+	Karaoke	[21, 28], 2	-	-	7	26	-	-
21	3.50	5+	Karaoke	[21, 28], 2	-	-	-	-	-	-
22	4.50	5.0	Pub	[23, 28], 3	-	-	-	-	7	26
23	4.00	5.0	Pub	[23, 28], 3	-	-	-	-	-	-
24	4.00	5.0	Pub	[23, 28], 3	6	24.02	-	-	-	-

Table 1: [1] Time requirement consists of a valid time window and the desired duration for that activity. When constructing the factor graph representation, we have assumed that the finest granularity of time is an hour. We found that in practice with a smaller unit of time, the domain size becomes too large and search becomes practically intractable.

5 Summary and Future Work

In this project, we formally defined the trip planning problem, and casted it as a constraint satisfaction problem with factor graph representation. Then, the backtracking search algorithm with most-constrained variable, least-constraining variable and arc consistency heuristics was employed to solve the abstracted version of trip planning. Due to pressing time, we only focused on problem-casting and evaluated the correctness of solutions, but we plan to make further improvements to make this more useable in everyday lives. First, probabilistic methods such as Gibbs sampling will be used to make an approximate inference to cope with the exponential search space. Then, we will try to taylor the trip planner for an individual user. This can be formulated as a machine learning problem where the objective is to learn optimal f^* given training data $\left\{\left((\textsc{EventQuery}^{(i)}, \textsc{Database}^{(i)}, \mathcal{I}^{(i)}), \textsc{Itinerary}^{(i)}\right)\right\}$.

6 Acknowledgement

We would like to thank Roy Frostig, Pokey Rule, Arun Chaganty and Sushobhan Nayak for helpful discussions and advices on how we can tackle some of the problems we faced.

References

[1] Google. *Google Maps API*. Nov. 2013 URL : `https://developers.google.com/maps/`

[2] Yelp. *Yelp API*. Nov. 2013 URL : `http://www.yelp.com/developers/documentation`

[3] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson. *Introduction to Algorithms* (2nd ed.). 2001. McGraw-Hill Higher Education.

Appendix : Map of solutions

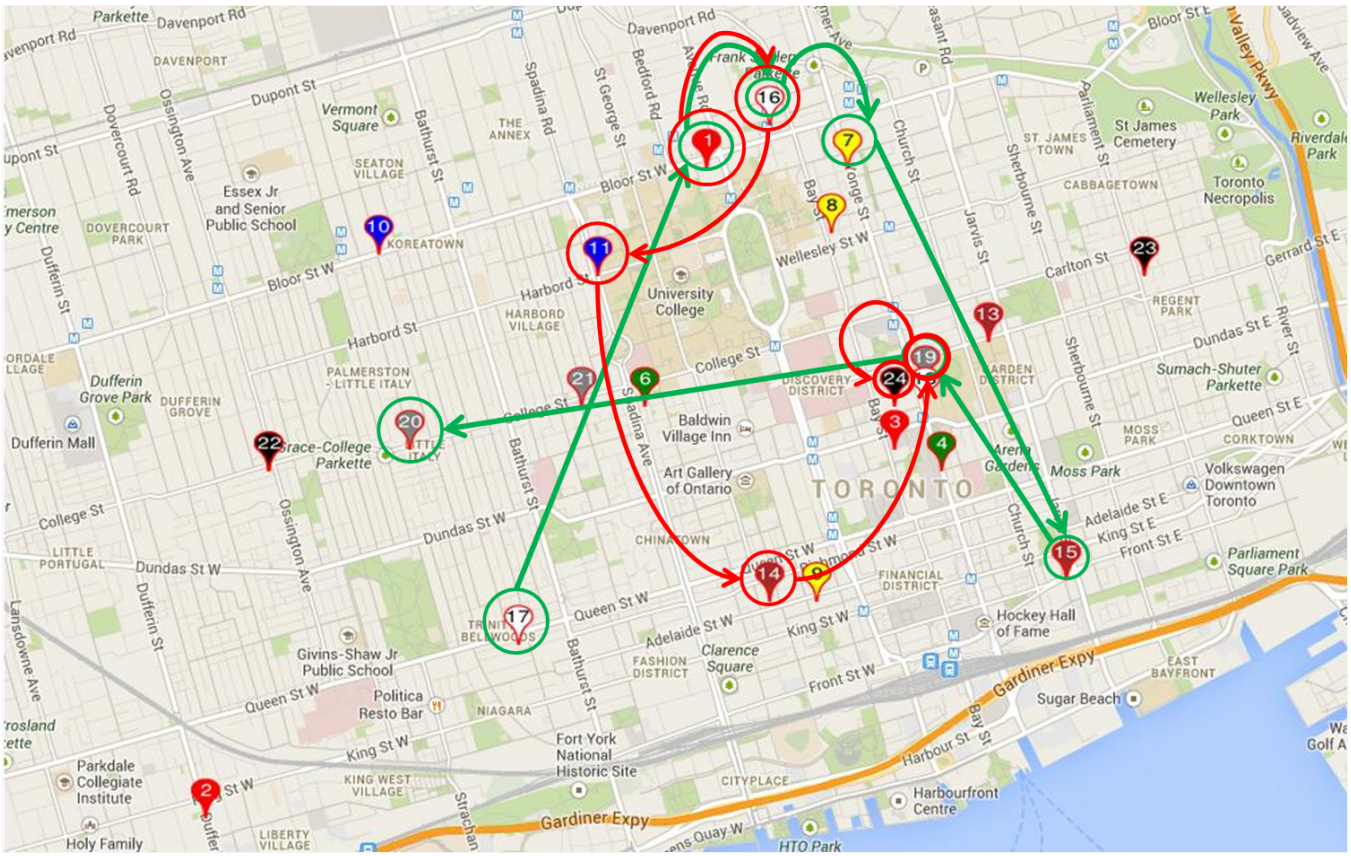


Figure 3: By scheduling the events by decreasing order of deadline, the baseline solver (in red) fails to find the optimal solution (in green) according to our objective function. Indeed seven events can be scheduled in the optimal case, whereas the baseline solver manages to schedule on 6 events.