Project 1:

Context-aware Intelligent Tour Guide

CS420: Artificial Intelligence

Lafayette College

— Submitted to

Prof. Chun Wai Liew

A. Introduction

Lafayette College wants to upgrade their existing college tour system by introducing a self-guided tours for visitors. This recommender/planner system will incorporate the visitor's' personal academic, athletic, extracurricular and cultural interests, along with their desired campus events, in order to produce an optimal itinerary for that visitor. The system will produce a suggested list of places to visit and/or events to attend, as well as the reasons for visiting and/or attending. Specifically, the system and algorithms used should be adaptable and efficient enough to address all use cases and to provide the users with relevant and enjoyable itineraries.

B. Parameters

I. Input

The input is the information that the visitor (user) provides the system with upon scheduling their tour. This input is a form which inquires about a visitor's personal information, academic, athletic and extracurricular interests, desired tour events and tour details. The potential user interface is shown in *Figure 1.* and it includes a sample of the form that will collect user information. A visitor could fill this form out online or on the spot at the admissions office when scheduling a tour.

The first section of the form is personal information, shown in the top section o*f Figure 1*, which is defined as identifiable information regarding an individual. For our purposes, personal information will inquire about name, email, address, parent contact details, predicted entry term, student type (first-year or transfer), high school name, location and class year. This personal information will aid our system, as well as the admissions office, in keeping track of all interested visitors.

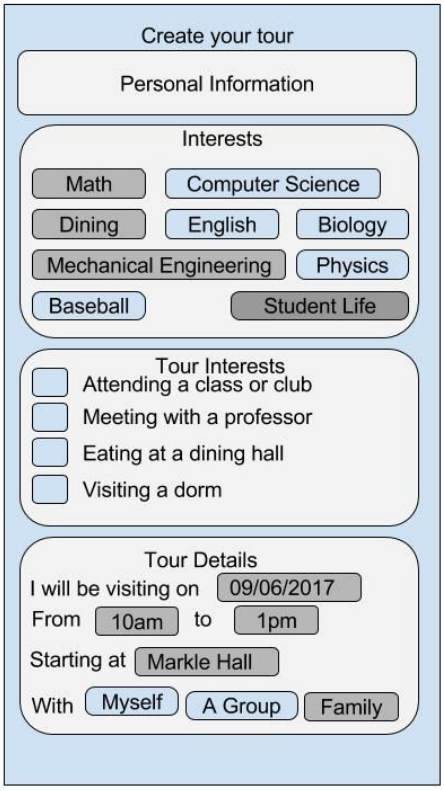

Figure 1. Potential user interface to collect information prior to creating a tour

The next section of the form is interests, shown in the top half o*f Figure 1.*, which will inquire about the academic, athletic, cultural and extracurricular passions of the visitor. This section will be in the form of a pool of predefined tags, such as [Math], [Computer Science] and [Baseball] in the figure. The visitor may select as few or as many as they feel apply to them. These tags will include majors, minors, clubs, sport and any other campus interests such as student life or the quad. These tags will be used to choose buildings to visit and possibly classes or club meetings to attend, if the visitor intends on doing so.

The next section of the form is desired tour events, shown in the bottom half o*f Figure 1.* as Tour Interests, which inquires about the desired tour activities of the visitor. This includes whether they would like to attend an introductory class in one of their areas of interest, attend a club or sport meeting, meeting with a professor in one of their areas of interest, eating at a dining

hall or visit a dorm. The introductory classes that will be available to sit in on and this will include classes where the professors have already approved visitors for.

The final section of the form is tour details, shown in the bottom section o*f Figure 1*. This section inquires about the specifics of the tour such as the date of the visit, the start and end time, the start location, who will be joining them on a tour and any disabilities. The data of the visit, start and end time are vital components for scheduling the tour because it will narrow down the possible buildings to visit and/or events to attend. Additionally, we think it would be beneficial to the visitor if we asked them about their time flexibility for the tour. This includes asking the visitor if they could spend an extra 10 or so minutes on the tour if the system finds a building or event of high interest to them that will not be able to fit without this time flexibility. The visitor will be provided with a drop down menu of possible buildings to start the tour from. If not specified, the starting point will be defaulted to Markle Hall because that is the admissions building and it provides visitor parking. It is important that the system is aware of any disabilities of the visitor or anyone visiting with them in order to allow more time between buildings and to ensure that the suggested itinerary has the appropriate accommodations.

After the user finishes filling out the form, the system will extract input data from the user's responses. This input will be used to initialize system variables that will be used to determine the personalized and optimal itinerary for the visitor. Their input data will be used in two main ways. First, our system will immediately use the data to generate and determine the personalized and optimal itinerary for the visitor. Second, we will also store the input to a database on a server managed by the school for recording and analyzing purpose. The admissions office have access to all this information: the visitors information, selected interest data, the generated tours and their selected tours. Admissions counselors will have the authorization to query the data directly from the database. The possible tours will be generated shortly after filling out this form and the user will have an option of choosing a prefered itinerary, display on mobile devices, printing the tour out, or have it sent to their email for future use.

The tour details section of the tour asks about who is going on the tour with the visitor. The form gives them three choices of companions on the tour: themselves, their family or as a group. If the visitor selects themselves, then they are visiting the school on their own and that single form will be the only input used by the system to determine the itinerary. If the visitor selects their family, then they are visiting the school with some numbers of people in their family. In this case, the form will ask for a number of family members (excluding themselves) that will be on the tour with them. The tour will still be catered only to the visiting student and their interests but by indicting the number of family members that will be accompanying the student, the system will be able to respect constraints such as building capacity. If there are more than one interested student, who want take the tour as a group, the system will prompt the user for additional information, shown in *Figure 2*. Each visitor will fill out the form with their individual interests and desired campus events but they will list the names of other members in their tour group. The system will be able to collect all the forms for each individual in the group by matching the names listed in the form with the names listed in one's personal information section. Once all the forms for each visitor in a group has been collected, the system will average the interest tags over all the group members by tag frequency. Depending on the number of students on the tour and the time allocated for a tour, the top x most frequently selected tags will be chosen. The system will then be able to generate a tour that applies to a majority of the individuals in the group while still respect building and event capacity, as well as tour duration.

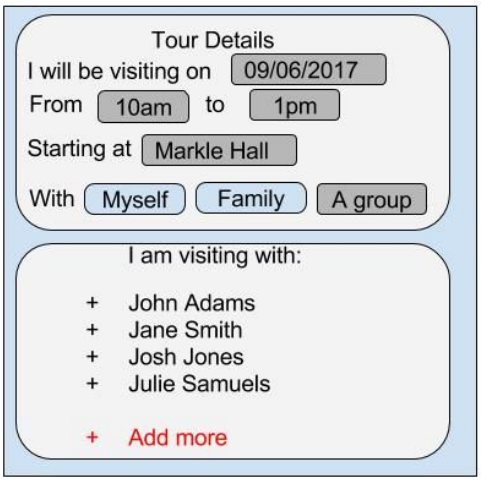

Figure 2. Additional section in user interface to handle group tours

User Input Summary

Personal Information: ● Name, email, address. ● High School (name, location, class year). ● Predicted entry term ● Student type (first-year or transfer) ● Parent contact details	Tour interests/activities: ● Attending a class, club or sport meeting. ● Meeting with a professor. ● Eating at a dining hall. ● Visiting a dorm or the student center
Academic, athletic, and extracurricular interests: ● Listed out via predefined tags	Tour Details: ● Date of visit. ● Start and end time. ● Start location. ● Who they are visiting with.

II. Knowledge

The knowledge is the information that the system will already have and will use, combined with the visitor input, to generate a tour. This includes campus information, schedules of events, classes, coaches and faculty and building information. This can be broken down into knowledge pertaining to buildings/landmarks and events.

For each building and landmark on campus, the system will have its name (full name and nickname), its use, its physical location on campus and its capacity (how many visitor can the building handle at any given time). It is important for the system to have information regarding its purpose. For each building, this includes knowing which majors and minors are housed there, as well as which professors have offices there, which clubs meet or practice there. If there are several buildings that serve the same or similar purposes, the system has statistical information

on which building is preferred to be shown. The times that a building or landmark is accessible is important combined with tour start and end time. Not all buildings are open 24/7 therefore the system must know what hours each building is accessible to avoid sending a visitor to a closed building. The system will have an average base time for how long one might spend visiting a building but this time might increase or decrease depending on the interests of the user and if they pertain to the buildings purpose. Using the physical locations of the buildings and landmarks, the system will have information on the distance between one and any other building on campus.

For every possible event a visitor could attend on their tour, the system will have its name, its use, its location on campus and its capacity. This will include information regarding classes (class names and basic descriptions), club meetings (club names and basic description), professor office hours and any information sessions. For each event, the system will have information on the average time a visitor spends there. The schedule of these events will be known to the system so that it can eliminate events given the user's date and time of visit. Additionally, the weather forecast can be known using the user's date and time of visit. Using this, the system will be able to suggest more appropriate events.

This data is gathered from various sources which will be explained in the pre-processing phase of the system. It is important to note how much the combination of fixed system knowledge and user input will help produce a personalized list of buildings and events a visitor can visit and attend, respectively. For example, the visitor's academic interests will help decide which building to visit, which introduction class to attend and which professors to talk to while the visitor's extracurricular interests will help decide which club meetings to attend. The user input helps weed out buildings and events that the user is not interested in.

System Knowledge Summary

Buildings and landmarks: ● Name, use, location, capacity	Events: ● Name, use, location, capacity

• Times accessible • Average time spent • Distance between one and any other building on campus	• Times they run from • Average time spent Weather forecast (via user input)

III. Output

Overall, our system will produce a list of possible itineraries/tours that best fit the user's interest and time range. These tours will be ranked by the interest points generated by the system, where a tour's total score is the sum of the scores of its locations/events. We will provide highlights of notable, distinguishing features of each tour, such as its strengths and its trade-offs. This way, the student can choose the one that matched their the majority of their preferences. As soon as the user decides on a final tour, the system will provide them with specific details of the tour and how to carry it out. Below is the details of our system's output.

- A list of itineraries:
    - Each listed tour includes interesting features which helps the user identify differences between the tours to allow them to form their decision. Ex:
        - One tour might allow the student to attend an information session while the other might allow the student to talk to a professor instead.
        - One tour could finish strictly in the allotted tour time while another tour that goes over the ending time by 5 minutes but allows the student to visit another building of interest.
    - User's final decision on a tour will be used to teach the system about user preferences for future improvement of the system.
- Details of selected tour:
    - A list of places/events to visit/attend in order.
    - Visual instructions for carrying out the tour.

    - Reasons to go to each building or event
        - Each building and/or event is justified by available descriptions collected from sources such as Lafayette's directory or Lafayette's virtual tour, as well as our internal scoring system, which quantify how relevant a location is to the user's interest.
        - For example, a visit to Pardee Hall being important if a user if interested in mathematics and selected the mathematics tag in the user input.
    - These information can be provided in a form of a campus map with visual details of the tour
- The user will have an option of viewing their selected tour directly on their mobile devices or exporting the itinerary as a static printout.

After a decision is made, beside providing the user with specific tour details, our system will also record the chosen itinerary and use it to improve future iterations. It will be inferred that highlighted distinctive features possessed by the tour are more favorable in compared to those of other available options. Our algorithm can then utilize such preferences to strengthen its future recommendations for buildings and events with regards to different interests.

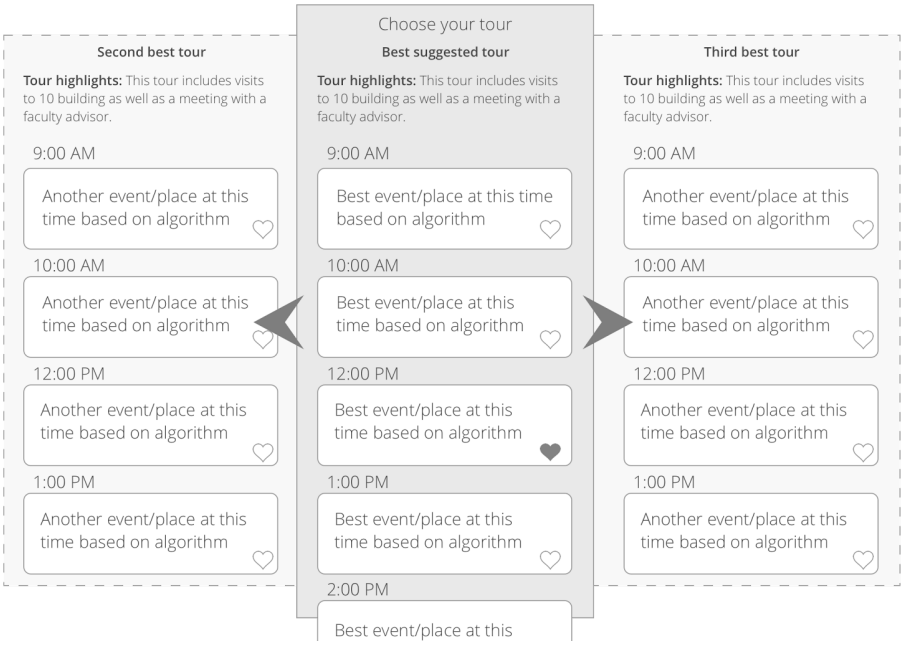

Figure 3. Potential user interface for choosing from multiple tours

IV. Assumptions

Assumptions are advantageous because they help the system remain constant and reduce the amount of different situations to consider. It is important to make assumptions about the external conditions of the system and how the system will be used in order to describe the best possible circumstances:

- Tours will be generated as if for a single user.
    - If the visitor is a student and their family, only the student will input their information.
    - If the visitor is a group of students with similar interests, they can add each other to their ‘group’ and allow the system to cumulate their interests.
    - If the visitor is a group of students with a wide range of interests, they probably should not take the tour as a group because the goal of this system is to map an individual’s interests to a custom tour.
- The user will expect a tour starting at their requested start time and last as long as they specified in their form. However, they may appreciate a particularly valuable event/location, according to their interests, even if it causes a slight overage in time.
    - If an event/location exists which would significantly improve the tour, then, realistically, it should not be excluded if it only caused the tour to be a very minor amount of time longer than specified. For example, in a tour scheduled for 4 hours, it would be expected that the user would not mind an extra 5 minutes if it meant a 15 minute faculty meet could be optimally added at the end.
- The user wants a tour which starts and ends at the same location.
    - For someone unfamiliar to the campus or for returning to the means of entry/exit to/from campus (parking spot).
- The user will be walking relatively slow because they are unfamiliar with campus.

    - As opposed to concerning variable travel times, we assume that people unfamiliar to their surrounding walk at a speed less dependant on their normal walking speed. This allows us to use a constant travel speed.
- The user spends the expected amount of time at events/locations.
    - This assumption is required because our output is static. The system completes its execution before the physical tour starts and we are constrained by the capacity of buildings.
- The user will be able to find a classroom intuitively in a building using the room number.
- The user knows the physical locations of buildings/event
    - Building names are given in the output and can be navigated with a campus map.
    - The user can navigate to the starting location independently.
- User input and feedback are honest

V. Constraints

As with any real-world project, there must be constraints drawn out in order to appropriately design the system:

- The starting location must be an actual node in the eventual graph representation.
    - This can be solved by assuming that the student can move a short distance to the nearest node at the start, if they are selecting to use the current location and time.
    - When scheduling a tour, the user would be only offered to start at locations known to and represented by the system.
- The end location must be the same as the start location in order to simplify use.
    - This requires the route through campus to be a circuit.
- Some buildings have restrictive access.
    - This means that the system cannot use these buildings at certain times of the day or of the year unless otherwise indicated.

- At least one itinerary must be produced as output for any combination of student input.
    - No input and full input (ex: all interest tags are selected) will fallback on the current state of the neural network's knowledge, yielding a default itinerary.
- The time span defined in the student's input will loosely constrain the length of the tour.
    - The intelligence will choose if better location may be visited in exchange for minor overages in time, proportional to the original time block.
    - The average overage can be tuned or learned by feedback.

VI. Use Cases

Use cases define the range of parameters and situations that our system must be able to handle. We defined use cases based upon the different parameters that the system will need handle . Different types of user, as well as number of users, require different forms of handling by the system. One student is handled by inputting their interests and information into the system standardly. A family is handled by setting the capacity of the tour to the size of the family but the interest values are only those of the student interested in the school. Groups of students are handled by each student entering their interests, grouping by names, and then averaging their interest tags. The number of students (and parents) is used as the capacity when considering events. The interests are then averaged across the group, then utilized in calculations for events as a single meta-student.

Time is handled in time chunks and implicitly through the system. During standard business hours, most buildings are accessible by students and events such as classes, information sessions, and or club meetings are available and accessible by the user. In the evening and mornings, buildings are not accessible and there are not many events that the user can attend; the system will then limit the tours to external visits of buildings and touring campus, accessing buildings when possible. Late night times are handled in a similar way, limiting the user to external tours of buildings. If the user misses their tour or decides to spontaneously tour the

campus, they can re-input their information online or at a kiosk at Markle to generate a tour starting immediately. Handicapness is accounted for through building accessibility information available in our knowledge base.

The number of interest tags input by the user results in different outputs. If a user inputs no interest tags, then the system will generate a tour as if all tags were selected, outputting the most general tour of campus possible, accessing a variety of events and buildings. If the user inputs very few interest tags, then the system will prioritize events and stops based upon those tags when generating the tour. If the user inputs many interest tags, then the system will generate a many tours based on those interests, each slightly prioritizing a different group of interests. If all possible interest tags are handled by the user, then the system handles tour generation in the same manner as no tags; the system generates a general tour.

VII. Evaluation Criteria

For evaluation, we take into consideration how different factors such as use cases or time are handled by the algorithms. With each factor, a list of relevant questions is composed to thoroughly assess the performance of the system. Details of these criteria are provided below:

- Use cases
    - Can the algorithm handle different use cases?
    - Are different use cases handled separately or is the algorithm generic enough to handle multiple ones in the same fashion?
    - Does it account for extreme cases such as busy hours, rainy day, or late evening tours? How well does it deal with these cases?
- Time
    - Can the algorithm generate tours of different lengths?
    - Can the algorithm generate tours for different times of the day?
    - Does the algorithm always satisfy user's input preferences for time? If not, why?

- Physical path
    - Does the algorithm produce reasonable paths which minimize its physical length?
    - Are the output paths the shortest ones to travel through given locations?
- Interests
    - Do generated tours include locations and events relevant to the user's field(s) of interest?
    - Are these locations/ events prioritized? If yes, how? If no, why?
    - Does the algorithm weigh locations/ events that are not directly of the user's interest? Does it rule them out entirely (and why?) or include certain ones (and how does it do that?) ?
    - How does the algorithm account for interests in the schools versus interests in specific major(s)?
- Others
    - How does the algorithm account for building capacity?
    - Is the algorithm scalable/ flexible?
    - What are the strengths of the algorithm?
    - What limitations does the algorithm have?

C. Algorithms

I. Previous Attempts

1. Dynamic tours

We have discussed a potential alternative approach consisting of generating a single tour and providing multiple choices at each time block when appropriate. The choices would be ranked by our recommender algorithm with the highest ranking event/building being the default choice. When a user selects an alternative event/building (*Figure 4.*), the part of the tour happening after the change can be invalidated (alternative choice could have been scheduled to be visited later that day, creating a repetition, or the new choice's end time could conflict with the next programmed event). In this case, we re-run our algorithm and generate the part of the tour happening after the new change is considered as the current state.

However, dynamically generating the tour at every node may introduce considerable stress on the system, and does not scale well with a bigger graph or larger number of users at the same time. Thus, we decided to focus on static tour generation.

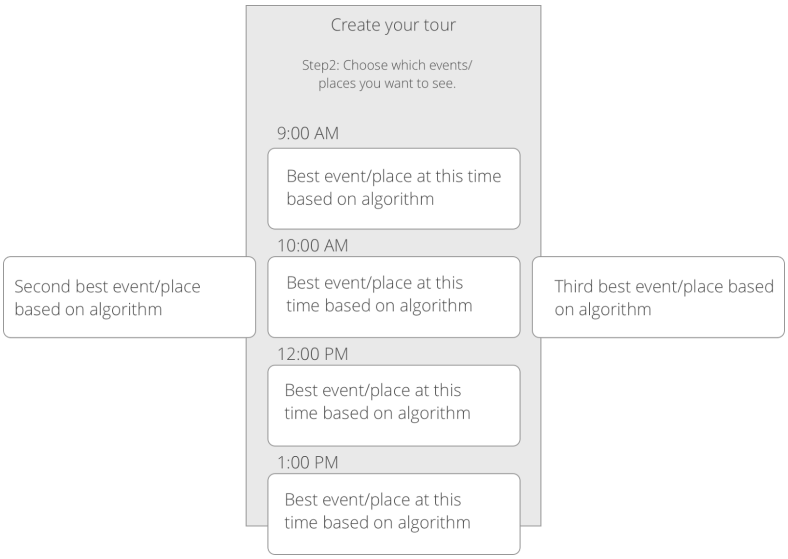

Figure 4. Alternative user interface approach for dynamic tours

2. Bayesian Networks

A Bayesian Network is a directed, probabilistic graphical model that represents the states of a situation that is being modeled. It is a way to encode real world knowledge in a representation that a computer can understand. It does so using random variables, values and nodes. The nodes represent random variables and the edges represent a node 'causing' another. For example, if a node A has an edge that points to node B, then node A causes B and therefore the probability that B occurs is P (B | A). [1]

In our case, the model would be the buildings and events that a user could visit or attend during their tour. The nodes would be any possible building a user could visit and the edges would indicate that one node directly influences another. We would have used Bayesian Networks to input a set of buildings and events based on their interest and obtain the graphical representation, including the probabilities, of those buildings and events in a graph that represents the campus.

While they are efficient in modeling reality and helpful in making prediction about how a model behaves, we determined that a Bayesian Network would not be the most applicable graph that we could use within our system. In fact, we have determined that a Bayesian network is very tied to conditional statements (ex. If a user liked Acopian then they will like Hugel with a probability X) that would have otherwise needed to be manually established. This would both bias the algorithm and tie it to our assumptions about buildings while at the same time rise the complexity of the setup of the system. On the other side, a bayesian network would not be able to represent events within buildings but would have to rely on the building's score to rate events.

3. A* Search

We have also considered using an informed heuristic search strategy to find the optimal tour path in the graph. A heuristic search strategy uses defined goals and problem specific knowledge in order to find solutions more efficiently. A* Search algorithm is a type of heuristic search strategy. This algorithm valuates nodes using a heuristic function which is a way to inform the search about the direction to the goal. It does by combining g(n) which is the cost to reach the node with h(n) which is the cost to get from the node to the goal into a single function: f(n) = g(n) + h(n). The result f(n) is the estimated cost of the cheapest solution through node n. This estimate of the total path cost from a start node to a goal node is constrained to start at that node. [1] A* is essentially a combination of lowest cost first and best first search algorithms because it considers both path cost and heuristic information in its selection of which node to go to next.

In order to use A* search as one of our AI techniques, we had considered implementing a sort of point system. We had talked about making every building and event that were available during the time slot selected by the visitor a single point. We realized that visitors are not going to want to see or attend every building or event, thus we explored the idea of giving buildings and events different points depending on the interests of the user. For example, if the user selected [engineering] tag, then Acopian Engineering Center would be given 5 points while Kirby Hall of Civil Rights would be given 1 point. This sort of system would need to be trained and/or defined so that the system would know how to properly assign points. This way, A* would try and gather as many points as possible while remaining under the time limit. The cost of the algorithm would be time (walking distance, time spent at buildings and events, etc) and the goal would be to attain at least x amount of points. We discussed having this algorithm run several times to see if, by choosing another node along the way, it could generate a path that has a higher total score while still remaining under the time requirement.

In the end, we decided to not use A* search algorithm because its function could not easily be implemented within our system. Time is not considered by the search. Also, our problem works to maximize interest under various constraints with no defined end location, whereas A* minimizes paths to a defined destination node. One particular aspect of our representation of the problem suggests zigzags may not be as detrimental as in traditional path finding. If our greedy algorithm zigzagged the user around the Hugel, Acopian, Kunkel area, they could simply, and may normally, choose those building in an order they prefer.

4. Shortest Path with Complex Weight

We explored an alternative planning agent to the ABC algorithm for generating an ordered list of locations after interests are mapped to building scores. The algorithm first involved generating generating a graph representing usable locations and events on the campus. The graph would be directed and fully connected to allow movement from one location to any other, and to represent a unique edge weight for every direction of movement. Nodes of the graph would be locations selectable for the itinerary, and where events existed within the location, they would be represented as subnodes of the location supernode. The starting location would be represented twice, and a path generate would start at one and end at the copy, to create a circuit on the physical campus. The duplicate starting node would also not be connected to each other like the rest of the graph.

Edge weights between supernodes would be calculated with an equation which would attempt to properly balance the importance of the various aspects of the planning problem. The user interest probabilities generated by a neural network would be algebraically inverted to represent higher interests with lower weights. In this algorithm, events would be outputs of the neural network aswell, and the resulting interest scores would be inverted and combined in some way into the ‘location’ super node's interest score. This resulting sum would then add the distance for each edge between location supernodes with some normalization to properly balance

between the importance of distance and value in the itinerary. Higher distances would be represented as higher weight.

$$Weight(A \rightarrow B) = (\Upsilon / Interest(B)) - (\varkappa / Interest(B.event)) + (б * Distance(A \rightarrow B) * T)$$

with

A: origin node

B: destination node

Υ = Location Interest gain

$\varkappa$ = Event Interest gain

$б$ = Travel time gain

T = Travel speed (constant)

With the weighted, fully connected, and directed graph representing the campus, a shortest path algorithm would be applied to generate the lowest cost path between the two duplicate starting nodes. This shortest path algorithm would need to constrain the minimum path to be at least the length specified by the user. Shorter/lower weight edges on the graph represent better choices in terms of an ideal tour itinerary because of the quantification and normalization of the inverted interest scores and the distance.

This algorithm was not used because of various implementation problems. Events are dynamically appearing choices which may only be available some days. Using the neural network requires extensive training of the network to make proper choices on outputs. Static objects like location work well, because the list of available locations changes only with construction and demolition on campus. Events, however, would need to be added and removed from the neural network as they open and close. An algorithm for properly weighing the distance and event interests alongside the location interest was never realized. It was also unclear how a shortest path algorithm could be modified to constrain the shortest path to at least the length of the requested time block.

II. Overview of the Final Algorithm

Before jumping into the technicalities of algorithms, a top level examination of our system will be beneficial in understanding its entirety. *Figure 5.* is a high-level overview of the steps our system executes in order to process a user's request. The system comprises of a pre-processing step to gather data and set up the system, and two algorithm phases putting different Artificial Intelligent techniques into use to match the user's preference and plan their itinerary.

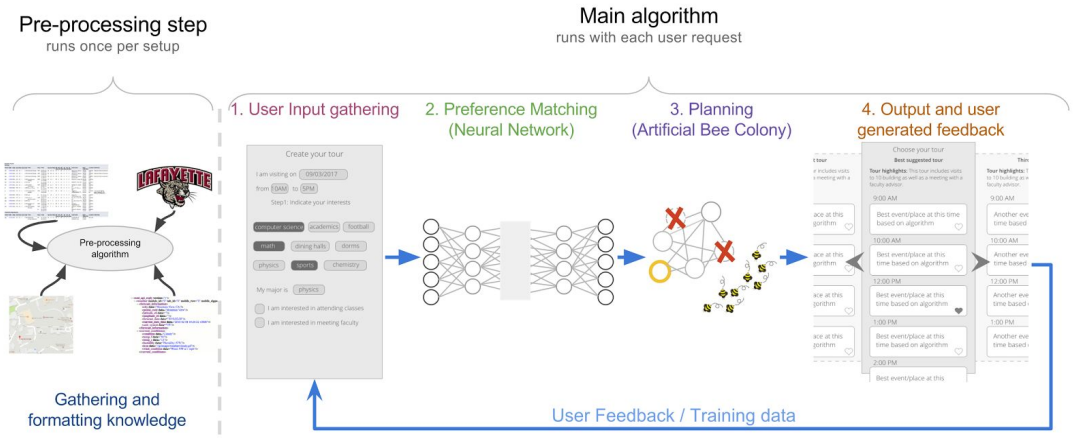

Figure 5. High level overview of system

The pre-processing step collects knowledge about buildings, landmarks, events from various sources. The knowledge is then streamlined, formatted and stored in a database. This knowledge, in combination with the user input, fuels our main algorithm. The main algorithm is comprised of two main functional parts solving two different sub-problems: the recommender sub-problem (user preference matching) and the planner sub-problem (itinerary generation and constraint satisfaction). The first functional part is achieved through a Neural Network inspired by the Word2Vec algorithm. It takes as input the user's preferences (chosen from a list of given keywords) and outputs a ranking of the building by probability of the user's interest in them (as a probabilistic value from 0 to 1). The neural network's output (building interest scores) is then used as input to the second functional part of our system. Building interest scores, combined with building distances computed in the pre-processing step, are used to generate a fully connected

graph of building, events and information about both. The graph becomes the search domain of the Artificial Bee Colony algorithm employed in the second functional part of the system.

III. Phase I : Pre-processing And Knowledge Discovery

In this phase, we will collect all the knowledge the algorithm will need to fully build context and make the correct decisions at later stages. We focused our efforts on utilizing as much already available information as possible without asking for system-specific knowledge or specialized formats of data. We have identified potential sources of existing data as well as ways to parse the information and recycle it into useful output for our system.

Looking back at our list of potential required knowledge, we have identified three major areas where we would need college-specific knowledge for the algorithm to function properly. These areas are:

1. Creating a map of buildings, their distances, walk time required to travel between every two buildings and the description of each building.

2. Establishing a list of tags that pertain to each of the buildings, which will double as the domain for user interest (ex. “science”, “dining”, etc.)

3. An up-to-date schedule of events/classes and faculty meetings that potential visitors may be interested to attend.

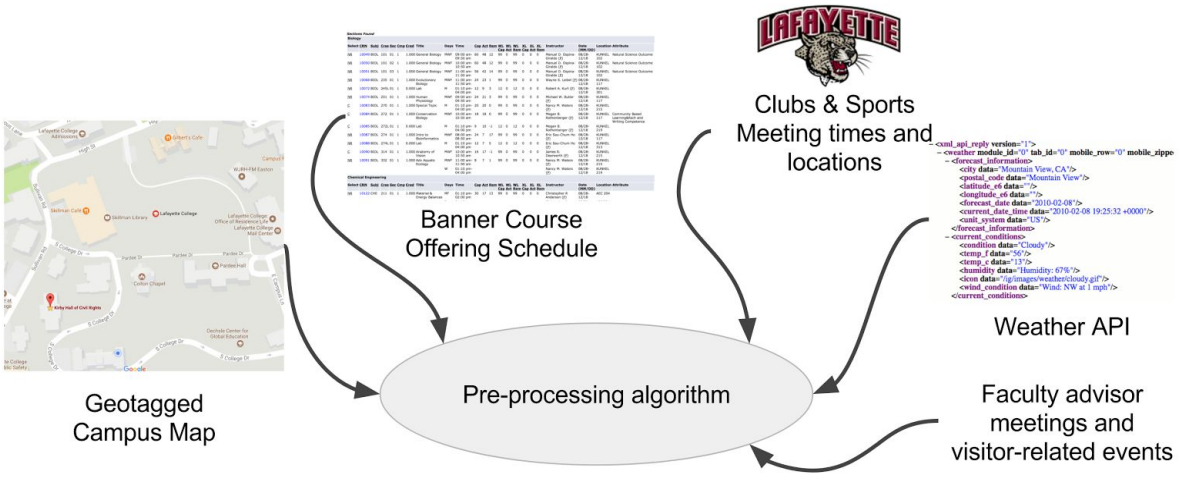

Figure 6. Overview of major sources of knowledge of the system

1. Pre-processing knowledge about building's geographical location

At the end of this step, we aim to output a fully connected graph of buildings with edge weights being the travel distance between every two buildings (*Figure 7*). This graph would then be augmented with more knowledge at each subsequent step of the pre-processing stage and the aggregate resulting graph would serve as input to the main algorithm.

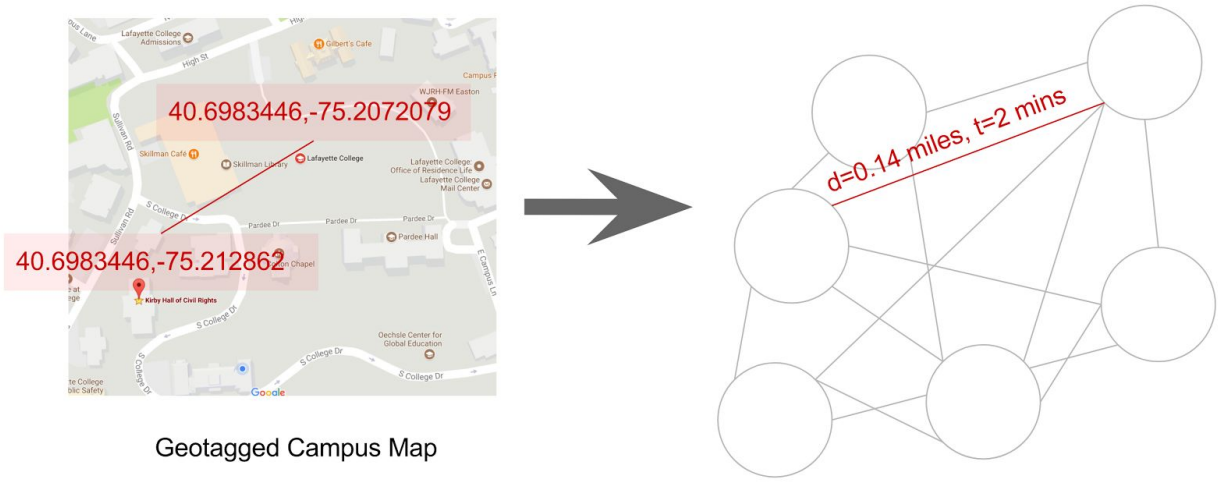

Figure 7. Overview of the inputs/outputs to the first step of the pre-processing stage.

We begin by considering a geotagged map of Lafayette's campus (in implementation, this would be easily obtained through the Google Maps API combined with Lafayette's campusmaps.lafayette.edu website), we then loop through the buildings and for each two building's latitude and longitude coordinates, we calculate the euclidean distance between them. If θ is the latitude and ϕ is the longitude then the distance between the two buildings is calculated as:

$$d(\vec{r_1}, \vec{r_2}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

with

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} R\cos\theta\cos\phi \\ R\cos\theta\sin\phi \\ R\sin\theta \end{pmatrix}$$

and $R \approx 6371\ km$ [4].

Through experimentation, we can then determine a static average walking speed of a touree (which is provided as knowledge to the algorithm) and therefore we can easily compute the travel distance between every two buildings.

2. Tagging buildings based on interest themes

At the end of this step, we would like to obtain a list of tags that pertain to each of the buildings we have in our system, which will eventually double as the domain for user interest (ex. “science”, “dining”, etc.).

To obtain this information, we wanted to harness as much available information sources as possible without asking for new information from the system administrators. To this end, we found multiple sources that can be used to match buildings to keywords/tags to finally obtain a list of tagged buildings (e.g. **Acopian** = [Engineering, Science, ..], **Hugel** = [Science, ..], **William’s** = [Painting, Art, ..], **Farinon** = [Dining, Student Life ..]). We first start by building a static list of tags/interests then use various Artificial Intelligence algorithms to match the buildings to their inferred tags. The initial list of tags/interests can be compiled by combining majors into their respective categories (e.g. Chemical Engineering, Civil Engineering, Mechanical Engineering $\Rightarrow$ Engineering, Science) as well as adding general keywords/interests such as “Sports”, “Fitness”, “Dining”, “Student Life”, etc. Once this list is compiled, the challenge that this pre-processing step tries to solve is guessing which tags belong to which building. We use various sources to infer this knowledge:

a. Source 1: Banner Self-service Course Offering Schedule

In addition to using the course offering schedule to obtain a list of first-level courses that campus visitors might be interested in attending (see section 3), we can cleverly use the course information to infer relationships between buildings and their associated majors/areas of interest since most buildings are usually associated with one or more disciplines. These relations can be inferred by running a frequency counting algorithm on the meeting locations of every course in the Banner-Self-service system then reverse-matching them with majors/tags on our predefined list. (e.g. *Figure 8.* Since most Economics courses are taken in Simon, we can infer the interest tags for the Simon building without any additional output. In other words, this example would give us the output tuple **Simon** = [Economics, Social Science, ..]).

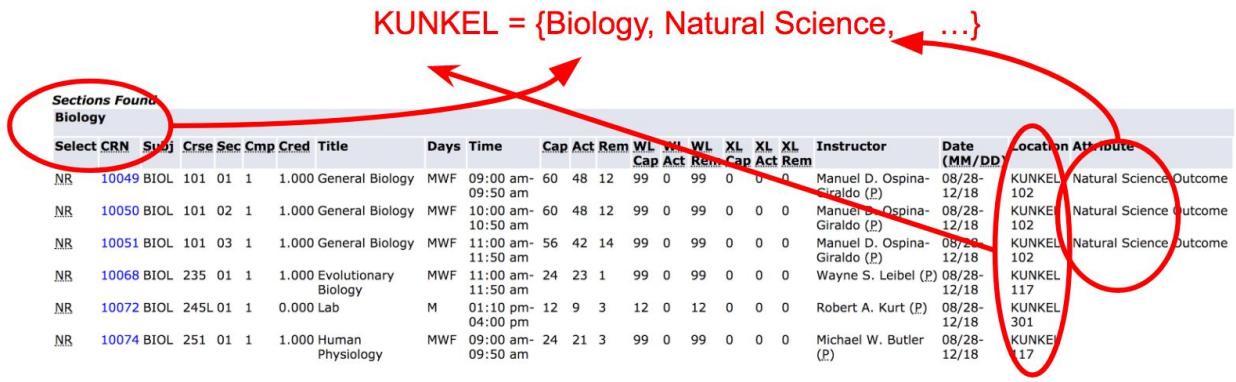

Figure 8. Inferring building tags through the course offering schedule

b. Source 2: Lafayette Directory, Online Virtual Tour and Campus Map

While the course offering schedule might give us information about academic buildings, other sources such as the Lafayette Directory web page might give us general information about all buildings while Lafayette's Virtual Tour might focus more on the rather "touristic" buildings. However, with these sources we are presented with a new challenge: While banner presented data in an easily parseable tabular format, these sources are destined to be consumed by visitors and students and therefore are bodies of text (paragraphs) that contain both keywords and other words that do not concern our algorithm (in particular, stop-words such as "a", "the", "we", etc.).

Although a simple frequency map – generated by counting how many occurrences of each word of our tag list appears in the text corpus – might suffice, this approach would ignore words that are closely related to tags on our list but are not exact matches. E.g.: If the description of a building contains the sentence "this building houses **scientific** majors", a simple frequency counting algorithm would not be able to identify that the word **scientific** relates to the tag **"Science"**. To solve this problem we recur to Natural Language Processing as an artificial intelligence approach that would help us match closely related words to each other. In particular, we would a modified version of the **Word2Vec** algorithm – detailed in the "User Preference Matching" section – coupled with a Lemmatization pre-processing and trained for relevance rather than proximity of words as detailed in Zamani's 2017 paper titled "Relevance-based Word Embedding" [6].

Figure 9. shows an example run of this algorithm where the input is the description of the building taken from the Lafayette College Directory and the output being the "relevant" tags from our list of tags/interests. Note how words such as "research" and "high-tech" do not appear in the output (since they are not part of our tags list), but were identified by the algorithm as relevant to other words in our tag list such as "technology" and "engineering".

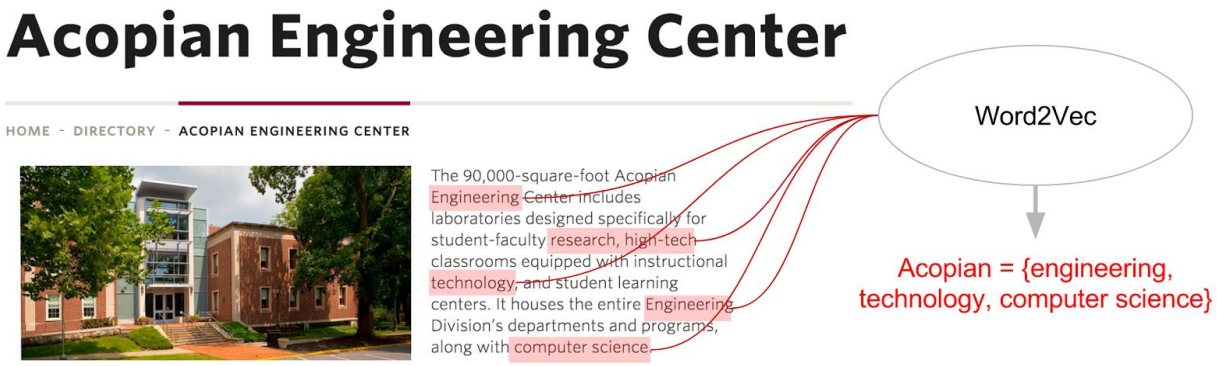

Figure 9. A building's page on the Lafayette Directory and the resulting tags matching the building through the relevance-based word embedding algorithm.

3. Compiling knowledge about event/class schedules

In addition to building visits, our algorithm is able to recommend events and classes to attend within a building to visitors. For this purpose, we need to gather information about the schedules of 100-level courses that offer visitations as well as schedules of sport events, advisor meetings and other relevant events (family lunches, president's talk, etc.). This knowledge is easily obtainable from the College Calendar, the sports schedule at GoLeopards.com, as well as relevant information from the Office of Residence Life, the Office of Co-curricular Programs and the Office of Scheduling. Once the knowledge is obtained, it would be parsed and normalized then added to the fully connected graph created in the previous step. *Figure 10.* shows the information after it has been added to a node of the graph.

4. Augmenting gathered knowledge with tour-specific information

To finalize the pre-processing step and obtain comprehensive information about the artificial intelligence agent's environment, we augment our gathered information with more tour-specific knowledge such as whether each building may be affected by weather conditions or not (for example, we would not want to schedule a visit of 15 minutes to the Quad if it is raining), whether buildings can be accessed without a key-fob or not, as well as the total capacity of each building which would help us determine how many people to send to a certain building at each time. At the end of the pre-processing phase, we end up with the graph illustrated in *Figure 10.*

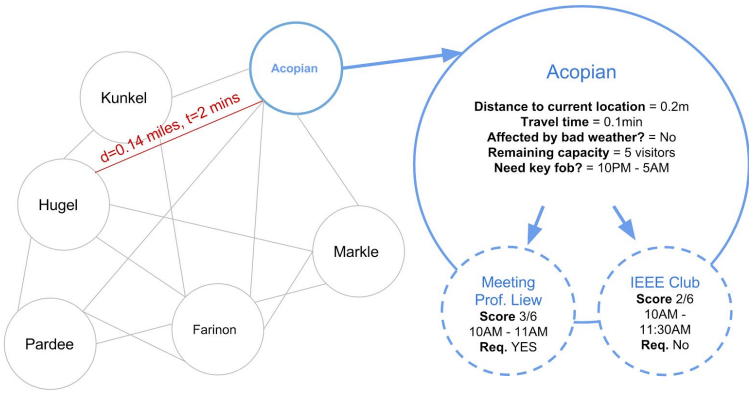

Figure 10. Example output from the pre-processing steps, serves as input to the main algorithm.

IV. Phase II : User Preference Matching (Recommender Sub-Problem)

The first sub-problem that our algorithm tries to solve is matching buildings and events within buildings to the user's specified preference. To this end, we employ a Neural Network based algorithm inspired by the Word2Vec algorithm (more specifically a Relevance-based Word Embedding algorithm [8]).

At this point, we have already created a static list of tags/interests (e.g. "technology", "science", "athletics", "dining", etc.) then through the pre-processing phase (section III.1.b) we matched Lafayette buildings to one or more tags/interests. At the same time, through user input to the interface, we gather a list of selected tags/interests that the user chose from the full list of tags/interests, which indicating the user's interest.

Looking at this phase from a high-level point of view, assuming the neural network has already been trained (we will go through the training step in depth later), the algorithm would take the user's chosen tags/interest list as input (as 1s for the selected tags and 0s for the rest of tags) then compute the probability of the user's interest in every building in the system as a number between 0 and 1 (*Figure 11.*).

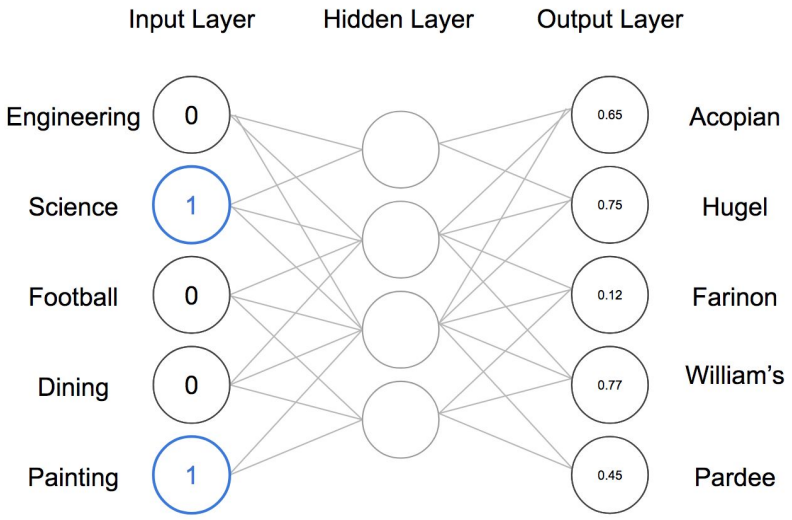

Figure 11. High-level view of the inputs/outputs of the recommender algorithm sub-problem.

To achieve the desired output, we have considered multiple algorithms, most notably the bayesian network (see the Previous Attempts section). However, we have determined that a neural network based on Word2Vec's architecture is the most fit algorithm for this problem given its numerous advantages:

- Easy to set up, would not require any assumptions about the building's relationships with each other like the Bayesian network
- Can be trained easily with the data that we have gathered in the pre-processing phase
- Once trained, running the algorithm is computationally efficient and can be easily ported to work on a mobile device (tablet or phone). Most of the computationally intensive tasks lie in the training step
- Easy to alter (by adding new buildings / tags) if enough data to fine-tune the neural network is available through the Net2Net process [9]

Overview of the traditional Word2Vec algorithm: The CBOW and Skip-gram models

Word2Vec models [7][14] are used to produce "word embeddings" – vector representation of natural language words. These models usually leverage a large corpus of input text to learn a vector space of hundreds of dimensions, where words from the corpus correspond to vectors in the space. The goal is to make related words with close meanings or context have their vectors in close proximity [7][14]. There are two model architectures for Word2vec: Continuous bag-of-words (CBOW) and continuous skip-gram. CBOW use a window of surrounding words as the "context" to predict the word itself. For example, a window of size 3 for the word "predict" in the last sentence would include 3 words before ("the", "context", and "to") and 3 words after ("the", "word", itself). With those six words as input, CBOW's job is to predict the word "predict." Continuous skip-gram, on the other hand, does the reverse; it takes the current word "predict" and tries to guess for "to", "the", "context", etc - the surrounding words.

Our approach: Cross-domain Relevance Based Word Embedding

Our approach consist of modifying the Word2Vec algorithm to obtain word embeddings based on the relevance of the source words to target words rather than the traditional approach (detailed above) which optimizes for proximity.

This algorithm is based on a rather simple architecture consisting of a feed-forward neural network with a single linear hidden layer [8]. The goal is to learn the weights associated with each term by training the neural network. We chose a simple single hidden layer structure because of its proven effectiveness [8] and as to avoid the problem of overfitting [1]. The activation function of the neural network at each node is given by the "Relevance Posterior Estimation Model" detailed in the same paper [8].

Modeling building/event recommendation problem as a cross-domain Word2Vec problem

	Word2vec Skip-gram	**Building Recommendation**
Input	A word as a one-hot vector	An interest tag (e.g. "engineering")
Output	Probability for each word in the vocabulary to be a nearby words	Probability for each building or event to be the one of interest
Training data	Pairs of (current word, nearby word)	Pairs of (interest tag, related building/event) *obtained from the pre-processing step*
Domains	Input domain = output domain = the entire vocabulary	Input domain = set of interest tags Output domain = set of buildings & events

Initial Training of the Neural Network

Initial training of the neural network is made easier given the abundance of source data that we possess, parsed through the pre-processing stage. Given a single building and the n tags associated with it (obtained in phases I.2.a and I.2.b), we can build a training set consisting of n

2-tuples that can be used to train the neural network to recognize that interest in that set of tags implies interest in that building.

E.g.: Given the relation **Acopian** = [“Engineering”, “Science”, “Technology”], we obtain 3 2-tuples of input/output (“Engineering”, “Acopian”), (“Science”, “Acopian”) and (“Technology”, “Acopian”). Training the network with these three 2-tuples would increase the probability of interest in acopian given any of the tags “Engineering” or “Science” or “Technology”. *Figure 12* given an overview of this training step.

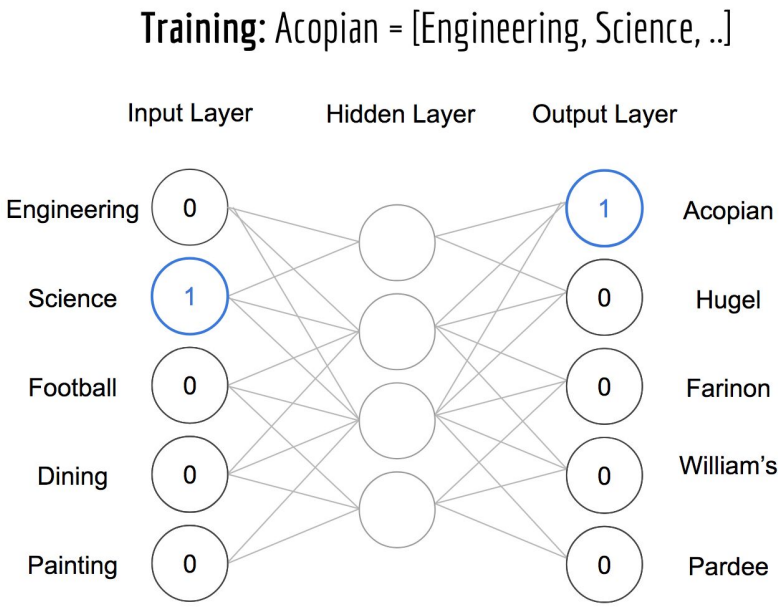

Figure 12. Training through the 2-tuple (“Science”, “Acopian”)

Fine-tuning the Neural Network based on user feedback (the learning step)

Following the initial training, the neural network should be recommending buildings based on the information collected from the various sources detailed in the pre-processing step (the college Directory, Campus Maps, the course offering schedule, etc.). However, in practice, students have diverse interests and different preferences. A user of our algorithm who specified a

list of interests might be interested in buildings other than the ones that were inferred based on Lafayette College's description of the buildings. On the other hand, a user who specified interest in a certain major might not enjoy visiting the building that we associated with that major during the pre-processing step. For this reason, we would like our algorithm to take user feedback into account and adaptively learn preferences as it is used more and more often.

One of the characteristics of neural networks is the ability to fine-tune the learned weights through more training without loss of knowledge. Our proposed user interface (*Figure 3.*) would have three main user feedback channels :

- User's choice of the second or third proposed itinerary which serves as an implicit deprecation of the best proposed itinerary
- Explicit feedback by "liking" certain events / building visits through the user interface
- Implicit feedback based on context awareness of the user's current location [2] as determined by the device's GPS module. This would give us information about the amount of time they spent at a building and thus can be considered an indicator of whether they liked it or not.

Based on data coming through these three feedback channels we can fine-tune the neural network by associating the the user's selected tags/interests with the chosen itinerary's buildings/events or with "liked" buildings/events. This would give the neural network a more subjective understanding of user preference than the initially trained network that was based on purely objective parsed data.

V. Phase III : Itinerary Generation (Planner Sub-Problem)

Augmenting the fully connected graph of buildings established at the pre-processing phase with building preference scores deduced from Phase II, we obtain an augmented graph G(V, E) as **input** containing the following information at each node (building) :

- User preference score for the current user's selected tags (obtained through a run of the Phase II algorithm)
- List of events hosted in the building (100-level classes, clubs, meetings, etc), their start/end times
- List of possible reasons to visit the building
- Time that building is open and if they require key fob
- Whether the building is affected by bad weather or not
- Remaining capacity of the building (in visitors)

In addition to this information, the graph also contains knowledge about distance between each two buildings (and the computed travel time between every two building by multiplying the distance by the average human walking speed).

Our algorithm's **state** at each step encompasses the variables :

- Visited Nodes (e.g. = [Marquis, Kirby, Kamine])
- Current location = (e.g. Acopian)
- Weather = Sunny/Rainy
- Current time (simulated, e.g. = 10AM)
- Tour end time (e.g = 2PM)
- Number of Visitors (number of visitors taking this tour e.g. = 1 for single user, or >=1 for a group tour)

Automatic generation of personalized itinerary is not an easy problem. The challenge is enhanced when facing multiple constraints such as certain events must happen at fixed time, locations have limited capacity, some locations are not suitable for certain type of weather or disabled users, … Thus, we believe that one highly potential approach for solving this problem is to treat it as a Constraints Satisfaction Problem (CSP). The basic idea of CSP is to view the problem of interest as a set of predefined constraints that need to be satisfied. A solution is a collection of possible states or conditions that together fulfil all constraints. [1] CSPs model the environments using factored representation, where each state has a set of variables with specific

values. For example, for our problem, a state is a building or location. The set of variables for each building would include those such as interest scores, topic tags, total capacity, remaining capacity, etc. The technique then utilizes the structure, exploits general-purpose heuristics (insteads of problem-specific ones), and can be applied to efficiently solve a wide range of complex problems. With CSP, one can quickly remove a large portion of the search space by filtering out entities that do not meet the constraints. For instance, in terms of our tour planner, events that not meet the user's time range can be eliminated immediately. The same principle is applied for locations that are not suitable for disabled users or rainy weather.

Among problems that are commonly represented as CSPs, scheduling problem appears to be the most applicable to our case because the problem is static: All activities, variables, and constraints are known. Generally, scheduling problems try to optimally allocate available resources, such as a given set of activities, with regards to time. In our case, these activities correlate with visiting locations and attending events. Examples of scheduling problems include fitting classes into the school year schedule, organizing to-do tasks in a process of production (job shop problem), etc.

Our problem as an analogy to the job shop problem:

Job Shop Problem	Personalized Tour Generation
Making a product requires several tasks	Taking a tour requires visiting several buildings and/ or events
Some tasks need to be done before others	Some locations/ events should be visited before other
Tasks need to be done between release and due dates	Some locations/event should be visited before other

Previous works show that algorithms such as Artificial Bee Colony (ABC) are capable of efficiently solving scheduling problems.[10][11][12][13] Thus, we utilize the Artificial Bee Colony algorithm to generate an itinerary that maximizes interest scores while minimizing

effects of constraints such as time, capacity of buildings or events, weather, and travel time. ABC is a swarm intelligence algorithm which mimics bee food collection behavior.[6] ABC excels in dynamic environments and assessing a large quantity of potential ‘food sources’ for an ideal choice. The basis of the ABC algorithm is the application of multiple agents in the form of ‘bees’. Bees can be one of three possible types at a single time: employee, onlooker, and scouts. First, scout bees randomly traverse the graph for a potential food source. Once they find a food source, the scouts transition to employee bees, associated with the food source, and return to the hive. The employee bees dance to onlooker bees, who assess the fitness/ quality, of the food source. Food sources are greedily selected based upon the fitness/ quality, which is determined in the baseline algorithm by distance and quantity of food.[3][5]After the best food source is selected, scouts return to the graph. Scouts whose solutions cannot be improved after a certain number of iterations abandon their current location on the graph and reenter at a random location. This allows for the refinement of solutions, as poor solutions are abandoned and negative feedback behavior arises in the swarm to balance positive feedback. Positive feedback in the swarm takes the form of onlooker bees visiting the source. This loop is repeated until a condition is met, in our case the generation of a complete itinerary for the user.

Food sources, terms of our application, are each possible location/ event that the user could visit during their tour time. Fitness of food sources is determined through the interest score of the location/ event, as well as constraints such as the distance to the location/ event, capacity of the location/ event, duration of the event, and when the event takes place. Starting with a empty itinerary, the only known information is the user’s start and end locations. From here ABC is applied, scouts search the graph for events that take place within the timeframe of the tour. Each scout returns with their possible solution and a greedy selection is made. Because the itinerary is empty at this point, higher priority events are implicitly selected, as their interest scores are higher and constraints will play a lesser role. Based upon the event selected, overlapping tags are removed; for example if the event selected is to sit in on a class, all other classes are ignored by the scouts for the remainder of the ABC algorithm. Once the event has been selected, the scout bees return to the graph to find further solutions. After the scouts return,

constraints are weighted more heavily in the greedy selection. As the time between locked in events decreases, distance between those two events plays a more important role in the selection, negatively impacting the fitness score of events that are further away. This prevents or combats potential zigzagging across campus, as events that are closer in physical proximity have higher fitnesses than those further away, and are thus more likely to be selected in the greedy selection. If the time between two events is too small, it is allotted to travel between the two events. This process repeats until the time block is fully filled.

ABC was selected because of it's ability to quickly assess a variety of solutions with a large number of a constraints to inform the selection, as well as the swarm's ability to refine over iterations to quickly find optimal solutions. Fitness scores of solutions based upon constraints allows for us to easily expand upon selection constraints. For example, the wrinkle of capacity as a constraint to the tour was quickly addressed by adding capacities to all events and comparing the capacity of the event to the size of the tour group inside the fitness calculation of the ABC algorithm. Events whose capacities are smaller than the group size are ignored by the scouts, and scouts are reinforced to ignore these events by both positive and negative feedback. The constraint system is thus easily expandable, allowing for further restrictions on event selection while still allowing for quick and efficient itinerary selection.[6]

Modeling itinerary generation problem as an ABC problem

Artificial Bee Colony	Itinerary Generation
Food sources	Possible buildings & events
Quality of the food source	Potential of buildings/ event to fit into the tour, assessed by interest scores, capacity, hasVisited, ...
Scout bees discovering food sources	System examining the campus graph with tour info returned by Phrase 2
Onlooker bees assessing returned ones	System choosing buildings and/or events
Positive sources	Chosen buildings or events for the tour
Onlooker bees visiting sources	User carrying out the tour, visiting buildings and events

VI. Algorithm Evaluation

Criteria	Our solution
Use cases	
Can the algorithm handle different use cases?	The algorithm handles most use cases detailed in section VI. See the Future Work section for potential optimization for rare use-cases.
Are different use cases handled separately or is the algorithm generic enough to handle multiple ones in the same fashion?	- Individuals-vs-Groups and Weather are handled separately (using average interest for groups, filtering out locations for rainy days) - Tours of different length or at different times of the day are generically handled, given daily access hours to buildings. ...
Does it account for extreme cases such as busy hours, rainy day, or late evening tours? How well does it deal with these cases?	- Busy hours: yes, buildings have predefined maximum capacity - Rainy days: yes, filtering unsuitable locations, have baselined tour - Late evening yes, filtering by building access, dynamically schedule available events
Time	
Can the algorithm generate tours of different lengths?	- Yes, the system will fill any amount of time with the maximum number of available events.
Can the algorithm generate tours for different times of the day?	-Yes, events on the graph are populated by time stamps but there are static events such as visiting a building (internally or externally) that are always on the graph.
Does the algorithm always satisfy user's input preferences for time? If not, why?	- The system will satisfy the user's input except for one case. If there is an event that will be interesting to the user but goes over the end time of the tour by a couple of minutes, the system will add it to the itinerary.
Physical path	
Does the algorithm produce reasonable paths minimize its physical length?	- Yes, ABC algorithm uses distance between events to constrain the greedy selection.
Are the output paths the shortest ones to travel through given locations?	- Not necessarily. The output path is the one that maximizes the interest score of the itinerary while reducing distance between events as much as possible.
Interests	

Do generated tours include locations and events relevant to the user's field(s) of interest?	- ABC algorithm generates tours to include locations and events that are relevant to the user interest inputs by favoring events with high matching interest scores.
Are these locations/ events prioritized? If yes, how? If no, why?	- Events are prioritized based upon their interest scores that are generated during phase II. Events that are prioritized have higher interest scores, and thus higher fitness scores. ABC will select these events first when generating the itinerary.
Does the algorithm weigh locations/ events that are not directly of the user's interest? Does it rule them out entirely (and why?) or include certain ones (and how does it do that?) ?	- Indirect interests: Events that are not directly related to the user input are still considered by the algorithm based upon phase II. Indirect events and locations' interest values are calculated based upon relationships in the neural net. The neural net generates and infers interest scores for all tags, not just those input by the user. Events that can be inferred to not be of any interest to the user are given low scores, and are then ignored during itinerary generation by ABC.
How does the algorithm account for interests in the schools versus interests in specific major(s)?	- The tag system utilized to generate interests includes topics that are not just academic. Our system can generate interest scores and considers events that are not limited to purely academic events. For example, student life, culture, and clubs are included in our tags.
Others	
How does the algorithm account for building capacity?	- Capacity: Building capacity and event capacity is included in building and event information. As itineraries are generated, ABC includes capacity as part of the fitness evaluation of an event. If the capacity of the event or building is less than or equal to the size of the tour, the event is considered as a possible solution. If the event is selected, the building or event's capacity is decreased by the size of the tour.
What are the strengths of the algorithm?	- Our user preference matching algorithm's strengths stem from the properties of neural nets. In fact the algorithm is easy to set up, does not require assumptions about the data, is computationally efficient on the test phase and can be easily tuned to account for more input. - ABC's strengths lie in its ability to assess a wide variety of potential solutions with many constraints on the selection criteria. ABC's constraint evaluation system can be easily expanded without decreasing efficiency. The swarm also increasingly improves through positive and negative feedback processes during the algorithm.
What limitations does the algorithm have?	- The user preference matching algorithm is computationally intensive for the training task, can easily be biased by false/outlier data and since it is based on a neural network, it is hard to understand the meaning of the weights and biases in case we would like to manually change them. - ABC's limitations lies in the solution selection processes. While the

	constraint selection processes allows for adaptability and scalability of the algorithm, as well as for maximization of interest scores, it makes decisions greedily. Without properly constraining distances, the algorithm can generate zigzagging routes. Fitness calculations also required when expanding the constraint system.
Is the algorithm scalable?	- The pool of predefined tags could be larger, depending on the number of majors, sports, or extracurricular activities the school has. This may require an sub-step of selecting/grouping tags so that the number displayed to the user is thorough but not overwhelmed. However, the rest of the algorithm still applies.

VII. Future work

Further work can be done to improve the proposed solution. Here we address a few key points that can be used to navigate such attempt to strengthen the algorithm.

- Examination of sources of biases
    - Users may “like” newer buildings that do not necessarily match preference
    - Groups of students that are more likely to “like” their buildings?
    - Users’ opinions may be affected by those of those while touring in groups
    - Are people more likely to pick the first suggested tour on the list without any consideration of other options? Could specific UI designs introduce biases?
- Privacy concerns
    - User input personal information need to be protected
    - Real-time location based data of the users need to be protected
    - Collected data of user interests that can be used to pinpoint the specific user need to be protected
- Expansion of the system
    - Since neural nets are easily portable and scalable, one potential area of improvement could be to apply the model to contexts of larger scale, such as big universities.

D. Conclusion/ Ethics

Artificial Intelligence has been regarded as a double edged sword considering it has the power to greatly aid yet severely hurt people and society. Before implementing and deploying a system, it is imperative to consider the advantages and the disadvantages that the system may bring about. Fortunately, this system has more advantages than disadvantages. Introducing a self-guided, personalized tour system to replace the current tour procedure will be valuable to both admissions and prospective students. For admissions, the investment in time and money to research, develop and assimilate this system will be considerable but the outcomes will be profitable. Admissions will be able to have a rather hands-off approach while still collecting a vast amount of data. Additionally, they will be personalizing the tour for each prospective student which might make the school more favorable of an option for the prospective student since their individual interests and desired events were met. The visitors will be more engaged in the tour since it will be their responsibility to direct and lead themselves. The final output will be available to the visitors long term such that they can look back at their itinerary when applying to schools or making a final decision. This will help leave a long lasting and memorable effect. One last major advantage is that admissions will save money by not having to pay tour guides and will save resources by not having to organize and disperse their touring shifts.

On the contrary, there are some disadvantages to implementing this system. Visitors will not be able to ask on the spot questions, there will be no “local knowledge” for the visitors to experience and it will relieve tour guides of their current jobs. This depends on how the admissions office decides to implement this system within their current procedure. Fortunately, none of these disadvantages are life threatening. Visitors with additional questions could attend an information session or meet with a counselor. Tour guides who need a campus job for financial reasons should be able to find another on-campus job. Additionally, it is possible for tour guides to still be available to visitors and to potentially lead them on their personalized tour.

In conclusion, we believe that our system efficiently implements a self-guided tour itinerary for visitors such that the tour will be personalized to the visitor within the visitor's time constraints. The visitor has a straightforward user interface where they can fill out a form to express their interests and desires to the system, the system takes it from there and successfully delivers the itinerary best fitted for that visitor. Lafayette College has the opportunity to implement this system in various ways thus making the system applicable to various tour procedures.

Team Members

- Agathe Benichou
- Wassim Gharbi
- Greg Shindel
- Nick Turney
- Thanh Vu

References

[1] Russell, Stuart J., and Peter Norvig. *Artificial intelligence: a modern approach*. Boston, Pearson, 2016.

[2] Abowd, G.D., Atkeson, C.G., Hong, J. et al. “Cyberguide: A mobile context-aware tour guide”. Wireless Networks (1997) 3: 421. https://doi.org/10.1023/A:1019194325861

[3] Karaboga, Dervis, et al. “A Comprehensive Survey: Artificial Bee Colony (ABC) Algorithm and Applications.” SpringerLink, Springer Netherlands, 11 Mar. 2012, link.springer.com/article/10.1007/s10462-012-9328-0.

[4]“How do I convert the distance between two lat/Long points into feet/Meters?” Stackexchange, Stack Exchange Inc., 26 Mar. 2011, math.stackexchange.com/questions/29157/how-do-i-convert-the-distance-between-two-lat-long-points-into-feet-meters.

[5] Abu-Mouti, Fahad S., and Mohamed E. El-Hawary. “Overview of Artificial Bee Colony (ABC) algorithm and its applications.” 2012 IEEE International Systems Conference SysCon 2012, 26 Apr. 2012. IEEE Xplore Digital Library, doi:10.1109/syscon.2012.6189539.

[6] Brajevic, Ivona, and Milan Tuba. “An Upgraded Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Problems.” SpringerLink, Springer US, 10 Jan. 2012, link.springer.com/article/10.1007%2Fs10845-011-0621-6?LI=true.

[7] Bussiek, Jon. “Demystifying Word2Vec.” *Deep Learning Weekly*, 5 Feb. 2017, www.deeplearningweekly.com/blog/demystifying-word2vec.

[8] Zamani, Hamed, and W. Bruce Croft. “Relevance-Based Word Embedding.” Arvix, University of Massachusetts, 9 May 2017, arxiv.org/abs/1705.03556. Accessed 23 Sept. 2017.

[9] Chen, Tianqi, et al. “Net2Net: Accelerating Learning via Knowledge Transfer.” Arvix, Google Inc., 18 Nov. 2015, arxiv.org/abs/1511.05641. Accessed 23 Sept. 2017.

[10] Wang, Ling, et al. “An effective artificial bee colony algorithm for the flexible job-Shop scheduling problem.” The International Journal of Advanced Manufacturing Technology, vol. 60, no. 1-4, 2 Sept. 2011. SpringerLink, doi:https://doi.org/10.1007/s00170-011-3610-1.

[11] Kimpan, Warangkhana, and Boonhatai Kruekaew. “Heuristic Task Scheduling with Artificial Bee Colony Algorithm for Virtual Machines.” 2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS), 29 Dec. 2016. IEEE Xplore Digital Library, doi:10.1109/scis-isis.2016.0067.

[12] Madureira, Ana, et al. “Towards Scheduling Optimization through Artificial Bee Colony Approach.” 2013 World Congress on Nature and Biologically Inspired Computing, 2013. IEEE Xplore Digital Library, doi:10.1109/nabic.2013.6617872.

[13] Liang, Yun-Chia, et al. “Artificial Bee Colony for workflow scheduling.” 2014 IEEE Congress on Evolutionary Computation (CEC), 11 July 2014. IEEE Xplore Digital Library, doi:10.1109/cec.2014.6900537.

[14] Mikolov, Tomas. “Efficient Estimation of Word Representations in Vector Space.” ArXiv.org, 7 Sept. 2013, arxiv.org/abs/1301.3781.