A Mobile Application for Optimally Matching Real Estate Clients

Yu Asai and Steven Luu
Christopher E. Siu, advisor
Cal Poly, San Luis Obispo
March 2021
# CONTENTS

1 INTRODUCTION
2 BACKGROUND
  2.1 The Client-Property Assignment Problem . . . . . . 2
  2.2 The Minimum Weight Matching Problem . . . . . . 2
3 RELATED WORKS
  3.1 Zillow . . . . . . . . . . . . . . . . . . . . . . . . . 4
  3.2 Trulia . . . . . . . . . . . . . . . . . . . . . . . . . 4
  3.3 Compass . . . . . . . . . . . . . . . . . . . . . . . . 5
4 IMPLEMENTATION
  4.1 Application Architecture . . . . . . . . . . . . . . . 6
  4.2 Scoring Functions . . . . . . . . . . . . . . . . . . . 7
  4.3 The Kuhn–Munkres Algorithm . . . . . . . . . . . . 7
  4.4 Implementation Issues . . . . . . . . . . . . . . . . 8
    4.4.1 Authentication . . . . . . . . . . . . . . . . 8
    4.4.2 SwiftGraph . . . . . . . . . . . . . . . . . . 9
5 VALIDATION
  5.1 System Tests . . . . . . . . . . . . . . . . . . . . . 10
6 RESULTS
  6.1 Authentication . . . . . . . . . . . . . . . . . . . . 12
  6.2 Home Screen . . . . . . . . . . . . . . . . . . . . . 12
  6.3 Creating Matchings . . . . . . . . . . . . . . . . . . 13
  6.4 Viewing Matchings . . . . . . . . . . . . . . . . . . 13
7 CONCLUSION
  7.1 Future Work . . . . . . . . . . . . . . . . . . . . . 16
    7.1.1 Lead Tracking . . . . . . . . . . . . . . . . 16
    7.1.2 Server-Side Computation . . . . . . . . . . . . 16
    7.1.3 Real-Time Property Data . . . . . . . . . . . . 16
    7.1.4 Complex Preferences . . . . . . . . . . . . . . 16
    7.1.5 Additional Preferences . . . . . . . . . . . . . 17
  7.2 Summary of Contributions . . . . . . . . . . . . . . . 17
ABSTRACT

Real estate agents are often tasked with finding their clients’ ideal properties. This can be difficult because multiple clients may have varying preferences, such as number of bedrooms, square footage, or price. Furthermore, different clients may weight their individual preferences differently. Existing applications do not consider multiple clients’ satisfaction, nor do they allow clients to weigh their preferences, potentially leading to less-than-ideal matchings between clients and properties.

In this project, we design and implement an iOS application whereby real estate agents can match multiple clients with individually weighted preferences to properties scraped from web listings. We model this client-property matching problem as a weighted graph, and apply the Kuhn–Munkres algorithm to find matchings that lead to the greatest overall client satisfaction.
ACKNOWLEDGMENTS

Steven:

- To Lara, Garrett, and Elliot: you helped me realise that I belong in Computer Science, all while in the middle of a field under the Electric Sky. Thank you for all of your guidance, and showing me what’s possible when you listen to your heart, and chase your dreams.

- To Tiger and Robyn: you have brought me immense joy in life, just by being you. Let’s cook breakfast burritos sometime.

- To Ai and Yu: You made sure I was eating, made sure I wasn’t working too long, and made sure I was okay. Here’s to the culmination of 4 years of work, filled with many sleepless nights and unforgettable memories.

- To my sister, Lara: you have helped me in ways I cannot express through these words illuminated by the Red, Green, and Blue. You made sure I never felt alone in college, and introduced me to some of my best friends in life. Thank you for helping me realise what I wanted to study, helping me grow as a person, and for being my sibling.

Yu:

- To Steven: From meeting you as Lara’s brother, to my study buddy and cooking buddy to being my best friend. Best friend who selflessly help me succeed even if it means to put his time aside. I can’t wait for more memories we will make together.

- To my family: Thank you for being my biggest supporters. I love how we function well as a family, helping each other always. Excited to make more camping, outdoor adventures with you all.

- To all my friends (you know who you are): You have brought joy in my life. For some, I spent a lot of time studying to get our engineering degree, making studying bearable; enjoying our time talking about our career goals, talking about our dreams of becoming successful entrepreneurs. We have created unforgettable bond that we will continue to build on forever through the fun, sometimes sadness we endured. To many more.

- To all the professors I have encountered: Thank you for letting me into your office even if it wasn’t time for office hours, or even if you were not my professor. Letting me have your time because you genuinely like to help your students succeed. I really admire your patience and genuine will to help others. Special thank you to Chris who has been my advisor since my day1 of coding. You
have helped me succeed, seen me as a cpe101 student who did not understand for loops to cpe357 student who could not handle a new hard language C to loving C and graphs, of course. Always willing to help, explain in many different ways until I understand fully because he knows every student understands differently. P.S. Thank you for all the Hi-Chews and White Rabbit.

Thanks to:

- Our advisor and friend, Chris: Your guidance in these past 4 years has been invaluable - not only for this project, but in life as well. Thank you for everything, and yes, graphs are pretty cool.
I N T R O D U C T I O N

The phrase real estate refers to property, land, buildings, or natural resources, both above- and below-ground. Today, there are four main categories of real estate: residential, commercial, industrial, and land. In this project, we focus on residential real estate, which includes both newly constructed homes and resale homes.

There is a lot of work that goes into the day of a real estate agent. Getting client leads and contacting potential clients is just the precursor to the real work, which is working with clients to either sell their home or find them a new home. When clients are searching for their dream home, they may have certain criteria in mind, such as finding a home that has a certain number of bedrooms or one that falls within a certain budget. Any time that a realtor spends matching clients with properties is time that they cannot spend developing relationships with their clients.

In this project, we implement an iOS application to automate the process of matching clients and their preferences to properties scraped from web listings. In Chapter 2, we describe this problem and how it can be modeled as a graph. In Chapter 3, we examine other popular real estate applications and their implementations of similar automated recommendation systems. Then, in Chapter 4, we give an overview of our application, which implements the Kuhn-Munkres algorithm in Swift to solve the minimum-weight perfect matching problem. In Chapter 5 and Chapter 6, we demonstrate the correctness of our implementation by evaluating different scenarios. Finally, in Chapter 7, we summarize our project and explore potential future work.
2 BACKGROUND

2.1 THE CLIENT-PROPERTY ASSIGNMENT PROBLEM

A fundamental and novel feature of our project is to provide a way for multiple clients to specify weighted criteria characterizing their ideal homes. Suppose, for example, that Sarah is looking to purchase a single family home with 3 bedrooms for no more than $700,000. Further suppose that Sarah has told her real estate agent that price matters more to her than the number of bedrooms, and that her children could share a room if needed. The realtor must then make note of these preferences and match Sarah with a home for sale. This need not be a home that perfectly matches Sarah’s criteria, however, it must simultaneously maximize the satisfaction of all other clients that the realtor is currently managing.

This is the combinatorial optimization problem our application must solve: we are given a set of clients, $A$, and a set of properties, $B$, where $|A| = n$ and $|B| = m$. Using the clients’ preferences and the properties’ attributes, we must first develop a *scoring function*, $\delta : A \times B \rightarrow \mathbb{R}$, that provides a real-valued score for each possible client-property pair, where a lower score indicates a better match. We may think of $\delta(a, b)$ as the “penalty” of assigning client $a$ to property $b$.

We must then find an injection $f : A \rightarrow B$ such that $\sum_{(a, b) \in f} \delta(a, b)$ is minimized. That is, an assignment of exactly one unique property to each and every client such that the total penalty is minimized.

It is important to note that this assignment problem is *not* reducible from the satisfiability problem, or “SAT”. In SAT, each assignment potentially affects the suitability of all other remaining assignments: if a propositional variable $p$ is assigned the truth value $T$, this could change whether or not we prefer to assign the same value $T$ to another variable $q$. In contrast, in the client-property assignment problem, if a client $a_1$ purchases a property $b_1$, this does not affect how much another buyer $a_2$ prefers each property; it only means that $a_2$ cannot also purchase $b_1$. Importantly, it turns out that this difference allows us to solve the client-property assignment problem in polynomial time.

2.2 THE MINIMUM WEIGHT MATCHING PROBLEM

A *graph* $G = (V, E)$ consists of a set of vertices, $V$, and a set of edges, $E$. Each edge $e = (u, v)$ connects two vertices, $u$ and $v$, where $e \in E$, $u \in V$, and $v \in V$. Optionally, such an edge may be associated with a
numeral weight \( w_e \), and we may specify that it is directed only from \( u \) to \( v \), but not vice versa.

We can model the client-property assignment problem as a weighted graph. The sets \( A \) and \( B \) represent two subsets of vertices, \( A \subseteq V \) and \( B \subseteq V \). For all \( a \in A \) and for all \( b \in B \), there exists an edge \( e = (a, b) \) of weight \( w_e = \delta(a, b) \). Since the vertices are divided into two disjoint subsets, and the edges are exactly all of those between vertices in different subsets, such a graph is called a complete bipartite graph, \( K_{n,m} \).

Our goal, then, is to select edges, as each edge represents one possible client-property pair. No two selected edges may share a vertex, as a property certainly cannot be bought by multiple clients, and we assume that no client desires multiple properties. Finally, we wish to match as many clients as possible, while minimizing the total penalty incurred: the maximum matching of minimum weight.

One possible graph, modeling 3 clients and 4 properties, is shown in Figure 1 with the optimal edges drawn as solid lines. Note that clients are not necessarily matched with their locally optimal properties, nor properties with clients. For example, property \( p_2 \) incurs a penalty of 1 when matched with client \( c_1 \). However, a globally optimal solution instead matches \( p_2 \) with \( c_0 \), for a penalty of 3, which allows \( c_1 \) to be matched with \( p_1 \) instead.
There are many existing applications within the real estate industry which attempt to find optimal properties for clients. However, they all share the same fundamental limitation: they are primarily targeted towards home buyers, not towards real estate agents, and they therefore do not attempt to optimize over multiple clients. Perhaps due to their client-focused nature, they also attempt to predict what clients want, giving suggestions for other listings on their websites. However, they often do not allow clients to specify weighted criteria, such as price, square footage, or number of bedrooms. They may have simplistic filtering methods, but they eliminate properties that do not match clients’ preferences exactly, without consideration of how much a preference actually matters or how close a mismatch might be.

3.1 ZILLOW

Zillow [4] has a recommendation system utilizing a deep neural network in conjunction with their web application. They create a collection of vectors from each property’s attributes, construct a deep neural network vector space for each property, and perform the Siamese embedded method. By collecting data regarding which suggested properties were clicked on, they train their network to predict which properties will interest such clients in the future. This is designed to help a single client find their ideal single property, rather than the multiple clients with potentially overlapping preferences that are handled by our project.

3.2 TRULIA

Trulia [8] has a recommendation system for notifying clients of suggested properties via email. Like Zillow, they collect data regarding how their users interacted with their suggestions: for example, if a client is interested in a property, they should stay on the page longer, rather than reverting back to the search results page quickly. These clients and properties are used to create nodes in a bipartite network, on which is trained a gradient boosted logistic regression model. They find that what clients do not interact with is a better predictor of their underlying preferences than what they do, suggesting that the slight mismatches which we allow in this project are acceptable to most clients.
3.3 COMPASS

Compass [6] has a recommendation system to solve “learning to rank” problems by forming triplets of “anchor listings”, “positive listings”, and “negative listings”, where positive listings are viewed by users immediately after anchor listings, and negative listings are chosen randomly. Their model considers the differences between anchor-positive pairs and anchor-negative pairs. Like Zillow and Trulia, they ultimately attempt to provide personalized rankings of properties, additionally taking into account the freshness, and side-wide popularity of each listing.
IMPLEMENTATION

4.1 APPLICATION ARCHITECTURE

We created an iOS application using Swift, utilizing Google Firebase for authentication, database, and analytic services. When a user first opens our application, they are able to authenticate themselves using their Google account. Note that, in the context of our application, a “user” is a real estate agent, and different agents should have access to different sets of clients that they manage.

Within our application, each client is represented by a record containing two primary pieces of information: the client’s identity and their property preferences. For the purposes of this project, we consider the preferred number of bedrooms, number of bathrooms, total square footage, and available budget. An example set of clients is shown in Table 1. Similar records are created to contain available properties and their attributes, which were scraped from a variety of popular websites using a server-side Python script. An example set of properties is shown in Table 2. These records are presented by interactable UITableViews for realtors’ consideration — a realtor can view, create, modify, or delete both clients and properties.

<table>
<thead>
<tr>
<th>Client</th>
<th>Beds</th>
<th>Baths</th>
<th>Square Footage</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_0 )</td>
<td>pref.</td>
<td>3</td>
<td>2</td>
<td>1200</td>
</tr>
<tr>
<td>weight</td>
<td>1.00</td>
<td>0.21</td>
<td>0.30</td>
<td>0.81</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>pref.</td>
<td>4</td>
<td>4</td>
<td>1750</td>
</tr>
<tr>
<td>weight</td>
<td>0.68</td>
<td>0.28</td>
<td>0.40</td>
<td>0.40</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>pref.</td>
<td>3</td>
<td>1</td>
<td>1200</td>
</tr>
<tr>
<td>weight</td>
<td>0.50</td>
<td>0.50</td>
<td>0.68</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Table 1: An example set of clients and their preferences

<table>
<thead>
<tr>
<th>Property</th>
<th>Beds</th>
<th>Baths</th>
<th>Square Footage</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_0 )</td>
<td>2</td>
<td>1</td>
<td>832</td>
<td>$699,999</td>
</tr>
<tr>
<td>( p_1 )</td>
<td>5</td>
<td>3</td>
<td>2358</td>
<td>$1,375,000</td>
</tr>
<tr>
<td>( p_2 )</td>
<td>3</td>
<td>2</td>
<td>1684</td>
<td>$1,050,000</td>
</tr>
</tbody>
</table>

Table 2: An example set of properties and the subset of their attributes which are considered during matching.
4.2 SCORING FUNCTIONS

We recall that, to assess the quality of each match, we must develop a scoring function \( \delta : A \times B \rightarrow \mathbb{R} \), where a smaller value of \( \delta(a, b) \) indicates a better match between \( a \)'s preferences and \( b \)'s attributes. Since clients indicate weights for each preference, the extent to which a single preference matches a single attribute can be described as:

\[
\text{weightedDifference} = \left( \frac{|\text{preference} - \text{attribute}|}{\text{preference}} \right) \cdot \text{weight}
\]

The total score is therefore, over all preferences and attributes:

\[
\text{weightedScore} = \sum \frac{\text{weightedDifference}}{\text{numberOfPreferences}}
\]

The only special case is budget: if a property’s asking price exceeds a client’s budget, then the total score is set to \( \infty \), as the client cannot afford the property. Else, the weighted difference corresponding to budget is set to 0, so that it is effectively ignored in the above computation.

Our application’s UI allows only weights in the range \([0, 1]\). Hence, the score will always be a non-negative real number, where a smaller score indicates a better match, and a score of 0 indicates a perfect match. As described in Chapter 2, these calculated scores are then used as edge weights in a weighted graph for the minimum weight maximum matching problem, as shown in Figure 2.

\[\text{Figure 2: The graph constructed from the values in Table 2 and Table 1. Note that all edges incident to } s \text{ or } t \text{ have weight 0, which are not shown for clarity.}\]

4.3 THE KUHN–MUNKRES ALGORITHM

The so-called “Hungarian Algorithm” was published in 1955 by Harold Kuhn. Its name comes from the fact that its development was largely based on the works of Hungarian mathematicians Dénes König
and Jenő Egerváry. In 1957, James Munkres observed that it was strongly polynomial, and since then the algorithm has been known as the Kuhn–Munkres algorithm \[5\].

We selected this algorithm because, as previously noted, it is able to solve our assignment problem in polynomial time. This means it is feasible for our mobile application to generate matches on the fly, rather than resorting to querying a server-side function to perform a more computationally intensive combinatorial optimization.

When the problem is modeled as a weighted graph, the Kuhn–Munkres algorithm additionally tracks vertices’ weights and edges’ directions \[1\]. It adds “dummy” vertices as necessary to ensure that \(|A| = |B|\), so that the solution becomes a perfect matching. It also adds source and sink vertices, \(s\) and \(t\), where \(s\) is the predecessor of all vertices in \(B\), and \(t\) is the successor of all vertices in \(A\). These augmentations are all shown in Figure\[2\].

The Kuhn-Munkres algorithm repeatedly finds the shortest path from \(s\) to \(t\), incrementally selecting edges until a perfect matching — which will consist of exactly \(n = m\) edges directed from \(A\) to \(B\) — is found. We implemented the algorithm using the open-source project SwiftGraph, which implements weighted, directed graphs as well as Dijkstra’s algorithm for finding shortest paths \[2\]. The result of running the Kuhn-Munkres algorithm on the graph from Figure\[2\] is shown in Figure\[3\].

![Figure 3: The perfect matching resulting from Figure\[2\]. Each and every client has been matched with exactly one unique property, and the sum of the selected edges’ weights has been minimized.](image)

### 4.4 IMPLEMENTATION ISSUES

#### 4.4.1 Authentication

When implementing authentication, we initially tried creating the TabBarViewController programmatically, which caused occasional display issues. The very first screen on the tab bar would load data...
incorrectly, causing the table controller to display incorrectly, and navigation controllers were not embedded properly within the view stack. The solution was to create a custom segue to the tab bar controller in the storyboard file, so that actions could be specified via storyboards instead of being instantiated programatically.

4.4.2 SwiftGraph

Before finding SwiftGraph, we tried to implement graphs and Dijkstra’s algorithm in Swift from scratch. This was costly in terms of time spent implementing and testing. After including SwiftGraph in our implementation, we also had issues integrating the library into our project. In our code, client and property vertices are represented by structures, whereas SwiftGraph only supports strings as vertices. To solve this issue, we used unique identifiers (p₀, p₁, ...) in order to name the vertices, and then used the vertex names as keys for a separate Swift dictionary of corresponding structures.
### 5.1 SYSTEM TESTS

<table>
<thead>
<tr>
<th>Description</th>
<th>Pre-Conditions</th>
<th>Input</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tests matching one client with an optimal property</td>
<td>The user is attempting to create a match, and there is at least one client and one property in the database</td>
<td>The user selects exactly one property and exactly one client, where the property matches all of the clients preferences</td>
<td>The client will be matched to the property, where each match is a perfect matching</td>
</tr>
<tr>
<td>Tests matching one client with a non-optimal property</td>
<td>The user is attempting to create a match, and there is at least one client and one property in the database</td>
<td>The user selects exactly one property and exactly one client, where the property does not match any of the clients preferences</td>
<td>The client will be matched to the property, even though it is a poor match, because there is no other option</td>
</tr>
<tr>
<td>Multiple clients multiple properties</td>
<td>The user is attempting to create a match, and both client and property have more than 1 record in the database</td>
<td>The user selects an equal number of clients and properties, where properties can be both optimal and non-optimal</td>
<td>The clients will each be matched to exactly one property</td>
</tr>
<tr>
<td>More clients than properties</td>
<td>The user is attempting to create a match, and there is more clients than properties</td>
<td>The user selects more clients than properties</td>
<td>Not all clients will have a match, but every property will be matched with a client</td>
</tr>
<tr>
<td>More properties than clients</td>
<td>The user is attempting to create a match, and there is more properties than clients in the database</td>
<td>The user selects more properties than clients</td>
<td>All clients will be matched with one property, where each match is a perfect matching</td>
</tr>
</tbody>
</table>

Table 3: System test matrix
To validate our implementation, we designed a system test suite covering a variety of different scenarios that our application might encounter, as enumerated in Table 3. For each test, we created corresponding sets of clients and properties, just as we did in Table 1 and Table 2.

As our application solves a problem that is not considered by any existing real estate software, as described in Chapter 3, we made no attempt to compare our application’s results to those of any other.
6 RESULTS

6.1 AUTHENTICATION

When a real estate agent first opens our app, they are prompted to log in using their Google account, as shown in Figure 4. Authentication is handled by Firebase and determines the client and property data that a user can access.

![Figure 4: Login and Google Authentication](image)

6.2 HOME SCREEN

Authenticated users are shown a home screen, as shown in Figure 5. From here, they can choose to create a matching. Alternatively, they can view or edit client and property data.
6.3 creating matchings

Upon clicking “create new matches”, users are walked through two views, as shown in Figure 6 and Figure 7. These views allow users to selection which properties and clients are available to be matched. For example, if a real estate agent wishes to reserve a particular property for a particular client, they can exclude that property and client when creating matchings.

6.4 viewing matchings

After available properties and clients have selected, the user is shown a table enumerating the ideal matchings between each client and property, as shown in Figure 8. If there are more clients than properties, some clients will be listed as not be matched; if there are fewer clients than properties, some properties will not be displayed in this table.
6.4 Viewing Matchings

Figure 6: Selecting Available Properties for Matching

Figure 7: Selecting Available Clients for Matching
### Figure 8: Ideal Matchings with the Data Selected from Figure 6 and Figure 7

<table>
<thead>
<tr>
<th>Clients</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yu</td>
<td>Asal</td>
</tr>
<tr>
<td>Lara</td>
<td>Luu</td>
</tr>
<tr>
<td>Chris</td>
<td>Siu</td>
</tr>
<tr>
<td>Steven</td>
<td>Luu</td>
</tr>
<tr>
<td>Al</td>
<td>Asal</td>
</tr>
<tr>
<td></td>
<td>3260 Whitwood Ct</td>
</tr>
<tr>
<td></td>
<td>926 Hy Glen Dr</td>
</tr>
<tr>
<td></td>
<td>170 S 34 Th St</td>
</tr>
</tbody>
</table>
CONCLUSION

7.1 FUTURE WORK

7.1.1 Lead Tracking

One potential expansion of our application is a CRM-style lead-tracking system. This could include keeping track of clients, logging any communications between clients and users of our application, and evaluating how much the client likes the property that was matched to them.

7.1.2 Server-Side Computation

Because the Kuhn-Munkres algorithm runs in polynomial time, we judged that it was feasible to implement within our mobile application. One potential improvement would to make the algorithm asynchronous: instead of displaying the match results immediately, we would send the selected clients and properties data to Firebase, where the computational power of Firebase could be used. Then, we could retrieve and store the results. This could potentially allow our application to run more smoothly on less powerful devices.

7.1.3 Real-Time Property Data

For the purposes of our project, we developed a Python script to scrape property listings from popular websites. We ran this script just once to populate an initial dataset, with which we developed the rest of our application. To be truly useful, we would need to be able to periodically update our dataset. This could be as simple as setting up a server-side cron job to run the scraping script. However, outside of the context of an academic project, this data should properly be acquired using paid accounts and official APIs.

7.1.4 Complex Preferences

Currently, our application assumes that clients will have exactly all of the required preferences. There is no option for a client to indicate, “I don’t care.” For example, we assume that no client will say, “I recently inherited a lot of money, and I do not care about price.” We similarly assume that no preference is a deal-breaker, and we assume that “too
few" is equally bad as "too many". For example, we do not account for the possibility that a growing family may not mind having one too many bedrooms, but would not settle for one too few bedrooms.

7.1.5 Additional Preferences

To better future-proof our application, users should be able to add custom preferences and attributes, rather than being limited to the handful that we have explicitly mentioned. In addition, users should also be able to modify clients’ preferences within the app.

One notable preference that we have excluded is location. We did so because, although it is easy to calculate the Euclidean distance between two coordinates, accurately assessing location requires an external service, such as Google Maps, to quantify the distance between two points.

7.2 SUMMARY OF CONTRIBUTIONS

To ease the workloads of realtors and improve the matchings of clients to properties, we implemented the Kuhn–Munkres algorithm within an iOS application using Swift. Existing similar solutions employ sophisticated methods to predict preferences and suggest properties, but consider only singular client. We showed, by representing clients and properties as vertices in a weighted, directed graph, that we can consider the preferences of multiple clients, producing an ideal matching that maximizes overall satisfaction.


