Cook userID, itemID → 0/1

- popularity → Jaccard

→ \( x = [1, \text{pop}(i), \text{pop}(u), \text{Jaccard}, \text{cosine}, ... ] \)

\( y = \sigma(x \cdot \theta) \) ← ensembling

→ \((u, i) \rightarrow (u, i^-)\)

→ In test, every user appears 2n times and 'half are '+'ve
Assignment 1 tips...

→ cold-start
  user → item popular?
  item → metadata (e.g. ingredients)

Rating: \( \alpha + \beta_1 + \beta_2 + \gamma_i - \sigma \)

→ Surprise, TensorFlow
→ Early stopping → use val. every few iterations
→ how to choose K, start with K=1
→ initialization of \( \gamma \)
→ heuristics are fine too
Assignment 1 tips...

\[
\text{Logistic Regression: } \theta \text{ s.t. } \sum_{i=1}^{n} \left( y_i \cdot \theta^T x_i - \log(1 + e^{\theta^T x_i}) \right)
\]

\[
\text{Ridge: } m \text{ s.t. } \sum_{i=1}^{n} (y_i - x_i \theta)^2 + \lambda \theta^T \theta
\]

\[
t = \Theta \cdot x
\]

- n-grams, stopwords, stemming, etc.
- dictionary size, TF-IDF
- regularization
- other features: numbers, ingredients, timestamp?
Web Mining and Recommender Systems

Content and structure in recommendation
Learning Goals

- Study how to incorporate *features* into recommender systems
- Look at a few types of recommendation that don't conform to the basic settings we've studied so far
- Explore systems for online advertising
So far, our basic design principle when designing recommenders was to uncover *interactions between users and items* $f(u,i) \rightarrow y$

We essentially argued that *interactions* can replace *features* in recommendation scenarios.

In what scenarios might this logic not hold?
In what scenarios might this logic not hold?

- **Cold-start:** We may have only a limited amount of interaction data; our previous argument applied only in the limit if we have lots of interactions per user and per item.
- **Interpretability:** Features may help to understand what the model is doing, even if they don't improve performance.
- **Temporal evolution:** None of the previous arguments hold if preferences/properties are non-stationary!
- **Other settings:** Some settings just don't look like simple user/item interaction data like we've seen so far.
Basic plan:

• Introduce *Factorization Machines*, a general-purpose tool for modeling interactions among features

• Study a few other recommendation settings that don't conform to current assumptions (online dating, bartering, social recommendation)

• Talk about algorithms for ad recommendation
Web Mining and Recommender Systems

Factorization Machines
Factorization Machines are a general-purpose technique to extend (latent-factor) recommender systems to include arbitrary features.

Rather than modeling interactions between users and items, we can model interactions between any pair of features (user and item interactions are then just one of many).
Factorization Machines

**Idea:** Describe our interaction data in terms of a feature matrix, including the user ID, item ID, and other features associated with an interaction.

\[ x: [0000010000 \ 00000015.0 \ 016000] \rightarrow y: [4.5] \]

User, Item, Price, Day
Factorization Machines

**Idea:** Describe our interaction data in terms of a feature matrix, including the user ID, item ID, and other features associated with an interaction.

\[
\begin{bmatrix}
1000000 & \ldots & 000100000 & \ldots & 0001000 & \ldots & 15.95 \\
0001000 & \ldots & 0000000010 & \ldots & 0001000 & \ldots & 12.25 \\
0100000 & \ldots & 000100000 & \ldots & 0000010 & \ldots & 15.00 \\
0000100 & \ldots & 010000000 & \ldots & 0010000 & \ldots & 17.50 \\
1000000 & \ldots & 000000010 & \ldots & 1000000 & \ldots & 19.95 \\
0000100 & \ldots & 000010000 & \ldots & 0000010 & \ldots & 10.15 \\
\end{bmatrix}
\]

\[\begin{bmatrix}
\text{user} \\
\text{item} \\
\text{weekday} \\
\text{price} \\
\end{bmatrix}
\]
Recall: We argued before why we can't just put all of these features into a simple linear model – we need to capture interactions.
Recall: We argued before why we can't just put all of these features into a simple linear model – we need to capture *interactions*
Factorization Machine Equation

**Idea:** Every feature is associated with a low-dimensional vector:

\[ f(x) = w_0 + \sum \omega_i x_i + \sum \sum_{i < j} x_i x_j \]
**Idea:** Every feature is associated with a low-dimensional vector:

\[
f(x) = w_0 + \sum_{i=1}^{F} w_i x_i + \sum_{i=1}^{F} \sum_{j=i+1}^{F} \langle \gamma_i, \gamma_j \rangle x_i x_j
\]

- **offset and bias terms**: \( \alpha + \sum \beta_i + \sum \varepsilon \epsilon_i \cdot \varepsilon_j \cdot \gamma_i \cdot \gamma_j \)
Factorization Machine Equation - example

\[ x \in \{00\ldots1\ldots0, 0\ldots1\ldots00\ldots\} \]

\[ \alpha + \sum_{j} \beta_{j} \]

\[ \sum_{j} \sum_{k \neq j} x_{kj} \delta_{j} \cdot \delta_{k} \]

\[ \alpha + B_{u} + \beta_{i} + \delta_{u} \cdot \delta_{i} \]
Factorization Machine Equation - example

\[ x = \begin{bmatrix}
  \text{user} \\
  \text{iter} \\
  \text{price}
\end{bmatrix}
\]

\[ x_{\text{user}}, \quad x_{\text{iter}}, \quad x_{\text{price}} \]

\[ \alpha + \beta_u + \beta_i + p(i) \cdot \beta_p + x_{\text{user}} \cdot \sigma_i + p(i) \cdot x_{\text{user}} \cdot x_{\text{price}} \]
### Factorization Machine Equation – weekday feature

\[
\begin{bmatrix}
1000000 \ldots 000000010 \ldots 1000000 \ldots 19.95 \\
0000100 \ldots 000010000 \ldots 0000010 \ldots 10.15 \\
\end{bmatrix}
\begin{aligned}
\text{user} & & \text{item} & & \text{weekday} & & \text{price} \\
\end{aligned}
\]

\[
\alpha + \beta_u \gamma + \beta_i \delta_i + \gamma_{ui} \times x_i + \delta_u \times \delta_i \\
+ \delta_u \times \delta_i + \gamma_i \times \delta_u
\]

\[
= r(x)
\]
Recall: FISM (from Recommender Systems slides)

Replace

\[ f(u, i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i \]

with

\[ f(u, i) = \alpha + \beta_u + \beta_i + \frac{1}{|I_u \setminus \{i\}|} \sum_{j \in I_u \setminus \{i\}} \gamma'_j \cdot \gamma_i \]

Can be implemented with a Factorization Machine!
Factorization Machines: FISM

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
\frac{1}{N} & 0 & 0 & 0 & \frac{1}{N} \\
\end{bmatrix}
\]

item \( i \) being recommended \( \rightarrow \) items \& history

\( x_i \), \( i, j \)
Factorization Machines generalize many of the techniques we've seen to build "latent factor"-style models.

And many of the techniques we'll see later: as we explore sequential/temporal recommenders, many can also be implemented via factorization machines!
Web Mining and Recommender Systems

Factorization Machines in fastFM
Factorization Machines – fastFM

```python
In [4]: data = list(parseData(dataDir + "goodreads_reviews_comics_graphic.json.gz"))

In [5]: random.shuffle(data)

In [6]: data[0]
```

```python
Out[6]: {'book_id': '15799191',
       'date_added': 'Sun Jun 30 06:12:24 -0700 2013',
       'date_updated': 'Thu Jul 04 08:02:07 -0700 2013',
       'n_comments': 0,
       'n_votes': 0,
       'rating': 4,
       'read_at': 'Thu Jul 04 08:02:07 -0700 2013',
       'review_id': '86b43a448463928a5d7887b364e2fcd',
       'started_at': 'Sun Jun 30 00:00:00 -0700 2013',
       'user_id': '23852e2647c217deb24964aadb26be64',
       'year': 2013}
```

```python
In [7]: userIDs, itemIDs = {},{}

   for d in data:
       u, i = d['user_id'], d['book_id']
       if not u in userIDs: userIDs[u] = len(userIDs)
       if not i in itemIDs: itemIDs[i] = len(itemIDs)

   nUsers, nItems = len(userIDs), len(itemIDs)
```

```python
In [8]: nUsers, nItems
```

```python
Out[8]: (59347, 89311)
```
### Factorization Machines – fastFM

<table>
<thead>
<tr>
<th>In [9]:</th>
<th>( X = \text{scipy.sparse.lil_matrix}((\text{len(data)}, \text{nUsers} + \text{nItems})) )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>50000 x 150000</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>In [10]:</th>
<th>( \text{for i in range(len(data))} : )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{user} = \text{userIDs[data[i]]['user_id']} )</td>
<td></td>
</tr>
<tr>
<td>( \text{item} = \text{itemIDs[data[i]]['book_id']} )</td>
<td></td>
</tr>
<tr>
<td>( X[i, \text{user}] = 1 \ # \text{One-hot encoding of user} )</td>
<td></td>
</tr>
<tr>
<td>( X[i, \text{nUsers} + \text{item}] = 1 \ # \text{One-hot encoding of item} )</td>
<td></td>
</tr>
</tbody>
</table>

Target (rating) to predict for each row

| In [11]: | \( y = \text{numpy.array}([d['rating'] \text{ for d in data}]) \) |

Initialize the factorization machine

| In [12]: | \( \text{fm} = \text{als.FMRegression(n_iter=1000, init_stdev=0.1, rank=5, l2_reg_w=0.1, l2_reg_v=0.5)} \) |

Split data into train and test portions

<table>
<thead>
<tr>
<th>In [13]:</th>
<th>( \text{X_train, y_train = X[:400000], y[:400000]} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{X_test, y_test = X[400000:], y[400000:]} )</td>
<td></td>
</tr>
</tbody>
</table>
Train the model

In [14]: fm.fit(X_train, y_train)

Out[14]: FMRegression(init_stdev=0.1, l2_reg=0, l2_reg_V=0.5, l2_reg_w=0.1, n_iter=1000, random_state=123, rank=5)

Extract predictions on the test set

In [15]: y_pred = fm.predict(X_test)

In [16]: y_pred[:10]

Out[16]: array([2.23866277, 4.37847526, 3.84866594, 5.03002627, 3.94993096, 4.36740166, 4.22497778, 4.30029268, 3.28282377, 4.05036905])
Web Mining and Recommender Systems

Multisided Recommendation
"Recommendation" so far has consisted of identifying which items a user is likely to "like best" or be most likely to interact with.

When is this not suitable?
Multisided recommendation

When is this not suitable?

- when items are users (e.g. online dating)
- constraints (e.g. ad recommendation)
- social (e.g. group recommendation)
Recommendations should be made to satisfy the preferences of multiple people in a group (e.g. recommending a movie to a group of people)

How can we determine whether an item is compatible with a group of users?
(a) the item should be, **on average**, compatible with the users:

\[ f(G, i) \]

(given a user-to-item compatibility function \( f(u,i) \))

**Average Compatibility:**

\[ \bar{f}(G; i) = \frac{1}{N} \sum_{u \in G} f(u, i) \]

(a) the item should be, **on average**, compatible with the users:

(given a user-to-item compatibility function $f(u, i)$)

**Average Compatibility:**

$$rel(G, i) = \frac{1}{|G|} \sum_{u \in G} f(u, i)$$
(b) no user should hate the item (e.g. consider recommendation in cases where users have constraints)

Least misery: 

\[ f(G;i) = \min_{u \in G} f(u;i) \]

(b) no user should hate the item (e.g. consider recommendation in cases where users have constraints)

Least misery: \[ \text{rel}(G, i) = \min_{u \in G} f(u, i) \]
(c) users should have **consensus** about the quality of an item (regardless of the actual scores)

**Average pairwise disagreement:**

\[
\text{dis}(G, i) = \frac{2}{|G|(|G| - 1)} \sum_{(u,v) \in G, u \neq v} |f(u, i) - f(v, i)|
\]

**Disagreement variance:**

\[
\text{dis}(G, i) = \frac{1}{|G|} \sum_{u \in G} \left( f(u, i) - \frac{1}{|G|} \sum_{v \in G} f(v, i) \right)^2.
\]

average compatibility in \( G \)
Amer-Yahia suggest that group recommendation should consist of a combination of the two elements (high **relevance** and low **disagreement**):

$$ F(G, i) = w_1 \times \text{rel}(G, i) + w_2 \times (1 - \text{dis}(G, i)) $$

(see more in paper re. how to actually select sets of items, and user studies re. what kinds of relevance/agreement functions people like)
Unlike other recommendation settings, the "items" in online dating settings are other users. A typical goal might consist of e.g. estimating reciprocal compatibility:

\[
\text{reciprocal compatibility}(u, v) = \frac{2}{f(u, v)^{-1} + f(v, u)^{-1}}
\]

Bartering (i.e., item exchange) is similar: recommendations are pairs of users, who must have compatible items to trade.
Web Mining and Recommender Systems

Socially-Aware Recommendation
Socially-aware recommendation

In a recommendation setting, we probably don't care about predicting the social network, but it can still be useful:

- Our friends are likely (?) to have similar preferences to us
- Knowing what our friends do could help to predict what we'll do next
Roughly speaking, we now have two sources of data to "explain":

\[
R = \begin{pmatrix}
5 & 3 & \cdots & 1 \\
4 & 2 & 1 & 3 \\
3 & \cdot & 3 & 4 \\
1 & 5 & \cdot & \cdot \\
\vdots & \vdots & \vdots & \vdots \\
1 & 2 & \cdots & \cdot \\
\end{pmatrix}
\]

\[
S = \begin{pmatrix}
1 & 0 & \cdots & 1 \\
0 & 0 & \cdots & 1 \\
\vdots & \vdots & \ddots & \vdots \\
1 & 0 & \cdots & 1 \\
\end{pmatrix}
\]
Two general strategies to incorporate social links:

1. Social signals can act as a *regularizer*. That is, a user's latent representation should be somehow similar to that of their friends.

2. Social signals help us to harvest more data: friends' actions live somewhere in between positive and negative feedback.
1. Social signals can act as a *regularizer*.

- The user parameter $\gamma_u$ should simultaneously be able to predict interactions and social links.
- In practice this means that if we have few *interactions* associated with a user, we can still fit their model (estimate $\gamma_u$) based on their friends.
Use a *joint objective:*
Socially-aware recommendation

Use a joint objective:

\[
\sum_{(u,i) \in R} (r_{u,i} - \sigma(\gamma_u \cdot \gamma_i))^2 + \lambda^{(\text{trust})} \sum_{(u,v) \in A} (a_{u,v} - \sigma(\gamma_u \cdot \gamma'_v))^2 + \lambda \|\gamma\|^2_2
\]

- rating prediction error
- trust prediction error
Socially-aware recommendation

$$r_{u,i} = \sigma(\gamma_u \cdot \gamma_i)$$

$$a_{u,v} = \sigma(\gamma_u \cdot \gamma'_v)$$
2. Social signals act as additional data from which to generate samples

**Recall:** When we built Bayesian Personalized Ranking we were interested in the relative strength of different signals, e.g. an interaction should rank higher than a non-interaction
What do friends' interactions mean?

(a) If a friend interacts with an item (that we haven't yet), we are more likely to interact with that item in the future

\[ x_{u,i} \geq x_{u,k} ; \quad x_{u,k} \geq x_{u,j} \]

positive \quad social \quad social \quad negative
Socially-aware recommendation

What do friends' interactions mean?

(b) If a friend interacts with an item (that we haven't yet), we are less likely to interact with it in the future (why)?

\[
\underbrace{x_{u,i}}_{\text{positive}} \geq \underbrace{x_{u,k}}_{\text{social}} \quad ; \quad \underbrace{x_{u,i}}_{\text{positive}} \geq \underbrace{x_{u,j}}_{\text{negative}}
\]
Web Mining and Recommender Systems

Recommendation in a few other settings
What's different about music?

- Hard to make use of signals: "listen to end" and "skip" are difficult to interpret as "positive" and "negative" (Pampalk et al. 2005)
- A large fraction of consumption is *repeat consumption* (Anderson et al. 2014)
- Difficult to extract useful features from songs (e.g. Wang and Wang, 2014)
What role does geography play?

- Subsequent activities are likely to be nearby (see sequential recommendation, coming up!)
- Location features could be hierarchical (see e.g. Zheng et al. 2009)
- Other familiar problems, e.g. location-based networks can have a social component (etc.)
Trivially, price can be incorporated into a factorization machine; but dynamics could be quite complicated

• Can we incorporate price *constraints* e.g. for recommending travel itineraries (Ge et al., 2011)
• Predict eventual purchase price from browsing trajectories (Hu et al., 2018)
• How do changes in price affect purchasing decisions (e.g. purchase quantity or brand) (Wan et al., 2017)
Lots still to cover (also in textbook)

• Incorporating **text** into recommender systems is not so straightforward; note that (e.g.) review data isn't available at test time! Lots of other text-based tasks too

• **Visual** features: high-dimensional, relationship to user preferences is subtle (coming up later)

• **Temporal dynamics**: turn out to be critical in real-world recommender systems (coming up later)
Web Mining and Recommender Systems

Algorithms for advertising
Predicting which ads people click on might be a **classification** problem.

Will I **click on** this ad?
Recommendation

Or... predicting which ads people click on might be a recommendation problem.
So, we already have good algorithms for predicting whether a person would click on an ad, and generally for recommending items that people will enjoy.

So what’s different about ad recommendation?
1. We can’t recommend everybody the same thing (even if they all want it!)

- Advertisers have a limited budget – they wouldn’t be able to afford having their content recommended to everyone
- Advertisers place bids – we must take their bid into account (as well as the user’s preferences – or not)

- In other words, we need to consider both what the user and the advertiser want (this is in contrast to recommender systems, where the content didn’t get a say about whether it was recommended!)
2. We need to be **timely**

- We want to make a personalized recommendations immediately (e.g. the moment a user clicks on an ad) – this means that we can’t train complicated algorithms (like what we saw with recommender systems) in order to make recommendations later.
- We also want to update users’ models **immediately** in response to their actions.

- (Also true for some recommender systems)
3. We need to take context into account

• Is the page a user is currently visiting particularly relevant to a particular type of content?
• Even if we have a good model of the user, recommending them the same type of thing over and over again is unlikely to succeed – nor does it teach us anything new about the user

• In other words, there’s an explore-exploit tradeoff – we want to recommend things a user will enjoy (exploit), but also to discover new interests that the user may have (explore)
So, ultimately we need

1) Algorithms to match users and ads, given **budget constraints**
So, ultimately we need

2) Algorithms that work in real-time and don’t depend on monolithic optimization problems

users arrive one at a time (but we still only get one ad per advertiser) – how to generate a good solution?

(users) -> .92 -> (advertisers)

(each advertiser gets one user)
So, ultimately we need
3) Algorithms that adapt to users and capture the notion of an exploit/explore tradeoff
Web Mining and Recommender Systems

Advertising: Matching problems
Let’s start with...

1. We can’t recommend everybody the same thing (even if they all want it!)

• Advertisers have a limited budget – they wouldn’t be able to afford having their content recommended to everyone
• Advertisers **place bids** – we must take their bid into account (as well as the user’s preferences – or not)

• In other words, we need to consider both what the **user and the advertiser** want (this is in contrast to recommender systems, where the content didn’t get a say about whether it was recommended!)
Let’s start with a simple version of the problem we ultimately want to solve:

1) Every advertiser wants to show **one ad**
2) Every user gets to see **one ad**
3) We have some pre-existing model that assigns a score to user-item pairs
Bipartite matching

Suppose we’re given some scoring function:

\[ f(u, a) = \text{score for showing user } u \text{ ad } a \]

Could be:

- How much the owner of \( a \) is willing to pay to show their ad to \( u \)
- How much we expect the user \( u \) to spend if they click the ad \( a \)
- Probability that user \( u \) will click the ad \( a \)

Output of a regressor / logistic regressor!
Then, we’d like to show each user one ad, and we’d like each ad to be shown exactly once **so as to maximize this score** (bids, expected profit, probability of clicking etc.)

\[
\sum_u f(u, ad(u))
\]

s.t.

\[
ad(u) = ad(v) \rightarrow u = v
\]

each advertiser gets to show one ad
Then, we’d like to show each user one ad, and we’d like each ad to be shown exactly once **so as to maximize this score** (bids, expected profit, probability of clicking etc.)

\[
\sum_{u,a} A_{u,a} f(u, a)
\]

s.t.

\[
\forall a \sum_u A_{u,a} = 1
\]

each advertiser gets to show one ad
Bipartite matching

We can set this up as a **bipartite matching** problem

- Construct a complete bipartite graph between users and ads, where each edge is weighted according to \( f(u,a) \)
- Choose edges such that each node is connected to exactly one edge

(Each advertiser gets one user)
Bipartite matching

This is similar to the problem solved by (e.g.) online dating sites to match men to women. For this reason it is called a **marriage problem**

(each user of an online dating platform gets shown exactly one result)
Bipartite matching

This is similar to the problem solved by (e.g.) online dating sites to match men to women. For this reason it is called a **marriage problem**

- A group of men should marry an (equally sized) group of women such that happiness is maximized, where “happiness” is measured by $f(m,w)$

  compatibility between male $m$ and female $w$

- Marriages are monogamous, heterosexual, and everyone gets married

(see also the original formulation, in which men have a preference function over women, and women have a *different* preference function over men)
We’ll see one solution to this problem, known as **stable marriage**

- Maximizing happiness turns out to be quite hard
  - **But**, a solution is “unstable” if:
    - A man $m$ is matched to a woman $w'$ but would prefer $w$ (i.e., $f(m,w') < f(m,w)$)
    - The feeling is mutual – $w$ prefers $m$ to her partner (i.e., $f(w,m') < f(m,w)$)
    - In other words, $m$ and $w$ would both want to “cheat” with each other
Bipartite matching

We’ll see one solution to this problem, known as **stable marriage**

- A solution is said to be **stable** if this is **never satisfied** for any pair \((m,w)\)

  - Some people may covet another partner,
    - **but**
      - The feeling is never reciprocated by the other person
      - So no pair of people would **mutually** want to cheat
Bipartite matching

The algorithm works as follows:
(due to Lloyd Shapley & Alvin Roth)

• Men propose to women (this algorithm is from 1962!)
• While there is a man $m$ who is not engaged
  • He selects his most compatible partner, $\max_w f(m, w)$
    (to whom he has not already proposed)
  • If she is not engaged, they become engaged
  • If she is engaged (to $m'$), but prefers $m$, she breaks things off with $m'$ and becomes engaged to $m$ instead
The algorithm works as follows:

(due to Lloyd Shapley & Alvin Roth)

All men and all women are initially ‘free’ (i.e., not engaged)

while there is a free man m, and a woman he has not proposed to

\[ w = \max_w f(m,w) \]

if (w is free):

(m,w) become engaged (and are no longer free)

else (w is engaged to m'):

if w prefers m to m' (i.e., \( f(m,w) > f(m',w) \)):

(m,w) become engaged

m’ becomes free
The algorithm works as follows:
(due to Lloyd Shapley & Alvin Roth)

• The algorithm terminates
The algorithm works as follows:
(due to Lloyd Shapley & Alvin Roth)

- The algorithm terminates

(either the number of free people decreases at each step, or, if it stays the same, the happiness increases)
The algorithm works as follows:
(due to Lloyd Shapley & Alvin Roth)

- The solution is stable
The algorithm works as follows:
(due to Lloyd Shapley & Alvin Roth)

- The solution is stable

(suppose m and w prefer each other to their current partners, w' and m'
But m would have proposed to w before he proposed to w'
- if w rejected his proposal, she must have been with someone she liked better
- if w accepted his proposal (but dumped him later), it must also have been for someone she likes better)
Bipartite matching

The algorithm works as follows:  
(due to Lloyd Shapley & Alvin Roth)

• The solution is $O(n^2)$
Bipartite matching

The algorithm works as follows:
(due to Lloyd Shapley & Alvin Roth)

• The solution is $O(n^2)$

(every proposal is made at most once, and there are $O(n^2)$ proposals
The input is $O(n^2)$ (i.e., the compatibility function) so it certainly couldn’t be **better** than $O(n^2)$)
Can all of this be improved upon?

1) It’s not optimal
Can all of this be improved upon?

1) It’s not optimal

• Although there’s no pair of individuals who would be happier by cheating, there could be groups of men and women who would be ultimately happier if the graph were rewired

• To get a truly optimal solution, there’s a more complicated algorithm, known as the “Hungarian Algorithm”
  • But it’s O(n^3)
  • And really complicated and unintuitive (but there’s a ref later)
Bipartite matching – extensions/improvements

Can all of this be improved upon?

2) Marriages are **monogamous**, heterosexual, and everyone gets married

- Each advertiser may have a fixed budget of (1 or more) ads
- We may have room to show more than one ad to each customer
- See “Stable marriage with multiple partners: efficient search for an optimal solution” (refs)
Can all of this be improved upon?

2) Marriages are monogamous, \textit{heterosexual}, and everyone gets married

- This version of the problem is known as \textit{graph cover} (select edges such that each node is connected to exactly one edge)
- The algorithm we saw is really just \textit{graph cover} for a bipartite graph
- Can be solved via the “stable roommates” algorithm (see refs) and extended in the same ways
Can all of this be improved upon?

2) Marriages are monogamous, heterosexual, and everyone gets married

- This version of the problem can address a very different variety of applications compared to the bipartite version
  - Roommate matching
  - Finding chat partners
  - (or any sort of person-to-person matching)
Can all of this be improved upon?

2) Marriages are monogamous, heterosexual, and everyone gets married

- Easy enough just to create “dummy nodes” that represent no match
- No ad is shown to the corresponding user
Bipartite matching – applications

Why are matching problems so important?

• Advertising
• Recommendation
• Roommate assignments
• Assigning students to classes
• General resource allocation problems
• Transportation problems (see “Methods of Finding the Minimal Kilometrage in Cargo-transportation in space”)
  • Hospitals/residents
Bipartite matching – applications

Why are matching problems so important?

• Point pattern matching
What about more complicated rules?

- (e.g. for hospital residencies) Suppose we want to keep couples together
- Then we would need a more complicated function that encodes these pairwise relationships:

$$\sum_{u,v} f(u, v, \text{hospital}(u), \text{hospital}(v))$$

pair of residents  hospitals to which they’re assigned
Surfacing ads to users is a little like building a **recommender system** for ads

- We need to model the compatibility between each user and each ad (probability of clicking, expected return, etc.)
- **But**, we can’t recommend the same ad to every user, so we have to handle “budgets” (both how many ads can be shown to each user and how many impressions the advertiser can afford)
- **So**, we can cast the problem as one of “covering” a bipartite graph
- Such **bipartite matching** formulations can be adapted to a wide variety of tasks
Further reading:

• The original stable marriage paper
  https://www.jstor.org/stable/2312726

• The Hungarian algorithm
  “The Hungarian Method for the assignment problem” (Kuhn, 1955):
  https://tom.host.cs.st-andrews.ac.uk/CS3052-CC/Practicals/Kuhn.pdf

• Multiple partners
  “Stable marriage with multiple partners: efficient search for an optimal solution” (Bansal et al., 2003)

• Graph cover & stable roommates
  “An efficient algorithm for the ‘stable roommates’ problem” (Irving, 1985)
  https://dx.doi.org/10.1016%2F0196-6774%2885%2990033-1
Web Mining and Recommender Systems

AdWords
1. We can’t recommend everybody the same thing (even if they all want it!)

- So far, we have an algorithm that takes “budgets” into account, so that users are shown a limited number of ads, and ads are shown to a limited number of users
- **But,** all of this only applies if we see all the users and all the ads in advance
  - This is what’s called an **offline algorithm**
2. We need to be timely

- But in many settings, users/queries come in one at a time, and need to be shown some (highly compatible) ads
- But we still want to satisfy the same quality and budget constraints

- So, we need online algorithms for ad recommendation
What is adwords?

**Adwords** allows advertisers to bid on keywords

- This is similar to our matching setting in that advertisers have limited **budgets**, and we have limited space to show ads
What is adwords?

**Adwords** allows advertisers to bid on keywords

- This is similar to our matching setting in that advertisers have limited **budgets**, and we have limited space to show ads
  - **But**, it has a number of key differences:

  1. Advertisers don’t pay for impressions, but rather they pay when their ads get clicked on
  2. We don’t get to see all of the queries (keywords) in advance – they come one-at-a-time
What is adwords?

**Adwords** allows advertisers to bid on keywords

- We still want to match advertisers to keywords to satisfy budget constraints
- But can’t treat it as a monolithic optimization problem like we did before
- Rather, we need an **online** algorithm
What is adwords?

Suppose we’re given

- Bids that each advertiser is willing to make for each query
  \[ f(q, a) \]
  (this is how much they’ll pay **if the ad is clicked on**)
  - Each is associated with a click-through rate
  \[ ctr(q, a) \]
  - Budget for each advertiser \( b(a) \) (say for a 1-week period)
  - A limit on how many ads can be returned for each query
What is adwords?

And, every time we see a query

- Return at most the number of ads that can fit on a page
- And which won’t overrun the budget of the advertiser (if the ad is clicked on)

Ultimately, what we want is an algorithm that maximizes revenue – the number of ads that are clicked on, multiplied by the bids on those ads
What we’d like is:

the revenue should be as close as possible to what we *would* have obtained if we’d seen the whole problem up front
(i.e., if we didn’t have to solve it online)

We’ll define the **competitive ratio** as:

\[
\frac{\text{revenue of our algorithm}}{\text{revenue of an optimal algorithm}}
\]

Let’s start with a simple version of the problem...

1. One ad per query
2. Every advertiser has the same budget
3. Every ad has the same click through rate
4. All bids are either 0 or 1
(either the advertiser wants the query, or they don’t)
Then the greedy solution is...

• Every time a new query comes in, select any advertiser who has bid on that query (who has budget remaining)
  • What is the competitive ratio of this algorithm?
Greedy solution
The balance algorithm

A better algorithm...

• Every time a new query comes in, amongst advertisers who have bid on this query, select the one with the largest remaining budget

• How would this do on the same sequence?
A better algorithm...

- Every time a new query comes in, amongst advertisers who have bid on this query, select the one with the largest remaining budget

- In fact, the competitive ratio of this algorithm (still with equal budgets and fixed bids) is \((1 - \frac{1}{e}) \approx 0.63\)

The balance algorithm

What if bids aren’t equal?

<table>
<thead>
<tr>
<th>Bidder</th>
<th>Bid (on q)</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>110</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>
The balance algorithm

What if bids aren’t equal?

<table>
<thead>
<tr>
<th>Bidder</th>
<th>Bid (on q)</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We need to make two modifications

• We need to consider the bid amount when selecting the advertiser, and bias our selection toward higher bids
• We also want to use some of each advertiser’s budget (so that we don’t just ignore advertisers whose budget is small)
The balance algorithm v2

Advertiser: $A_i$

fraction of budget remaining: $f_i$

bid on query $q$: $x_i(q)$

Assign queries to whichever advertiser maximizes:

$$\Psi_i(q) = x_i(q) \cdot (1 - e^{-f_i})$$

(could multiply by click-through rate if click-through rates are not equal)
Properties

• This algorithm has a competitive ratio of \((1 - \frac{1}{e})\).

• In fact, there is no online algorithm for the adwords problem with a competitive ratio better than \((1 - \frac{1}{e})\).

(proof is too deep for me...
So far we have seen...

• An **online** algorithm to match advertisers to users (really to queries) that handles both **bids** and **budgets**
  • We wanted our **online** algorithm to be as good as the **offline** algorithm would be – we measured this using the **competitive ratio**
  • Using a specific scheme that favored high bids while trying to balance the budgets of all advertisers, we achieved a ratio of \((1 - \frac{1}{e})\).
    • And no better online algorithm exists!
We haven’t seen...

- AdWords actually uses a second-price auction (the winning advertiser pays the amount that the second highest bidder bid)
- Advertisers don’t bid on specific queries, but inexact matches (‘broad matching’) – i.e., queries that include subsets, supersets, or synonyms of the keywords being bid on
Further reading:

• Mining of Massive Datasets – “The Adwords Problem”
• AdWords and Generalized On-line Matching (A. Mehta)