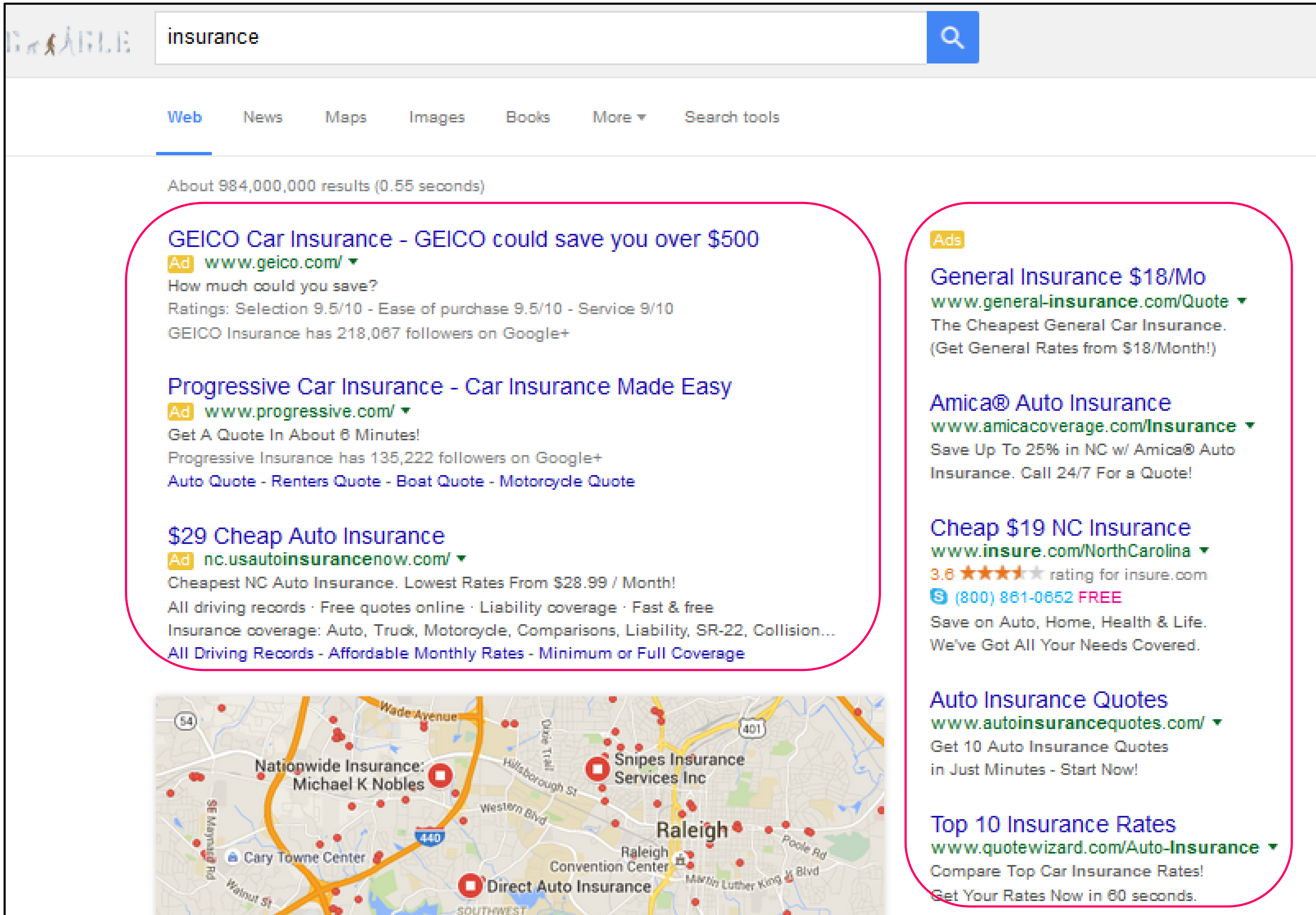
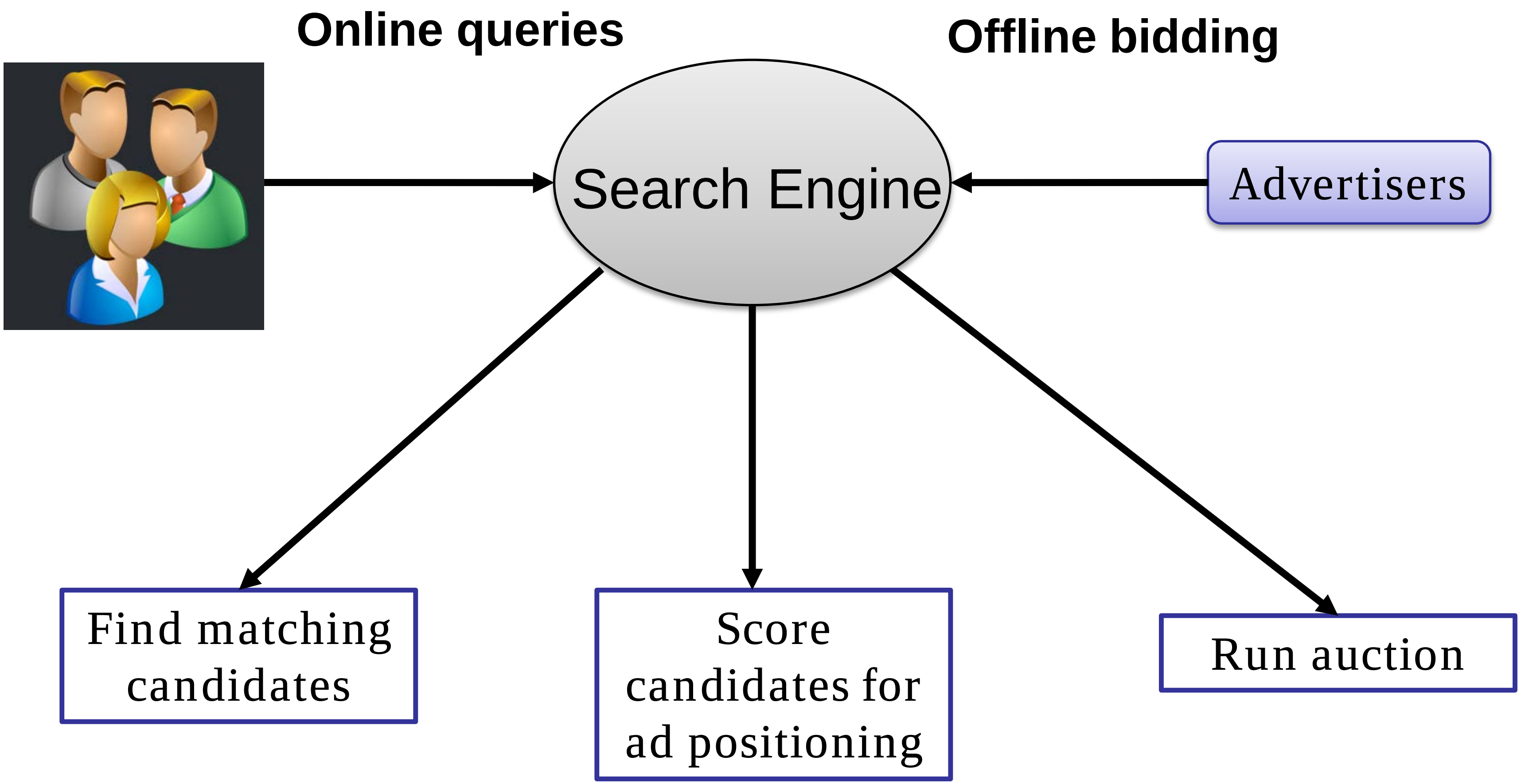
Online Bipartite Matching and the Adwords Problem

Kshitij Sharma and Steve Harenberg
Department of Computer Science
North Carolina State University

Online Advertisement or Adwords



Online Advertisement or Adwords



Graphs

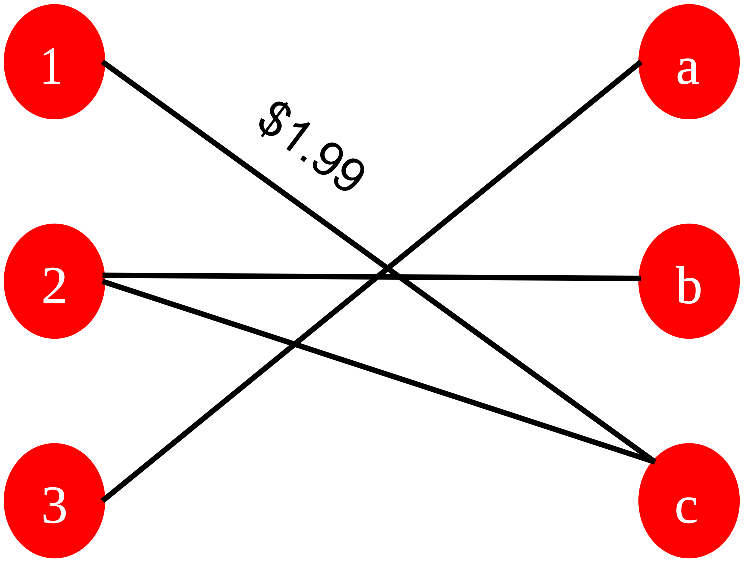
- A ***graph*** is a representation of a set of objects and the relationships between them.
- We denote a ***graph*** as $G = (V, E)$, where
  - V is a set of ***vertices*** (i.e., objects).
  - E is a set of ***edges*** (i.e., relationships between objects).

- Adwords problem formulated using ***graphs***.
  - ***Vertices*** represent advertisers and advertisement slots (which based on users search queries).
  - ***Edges*** represent an advertisers bid for that slot

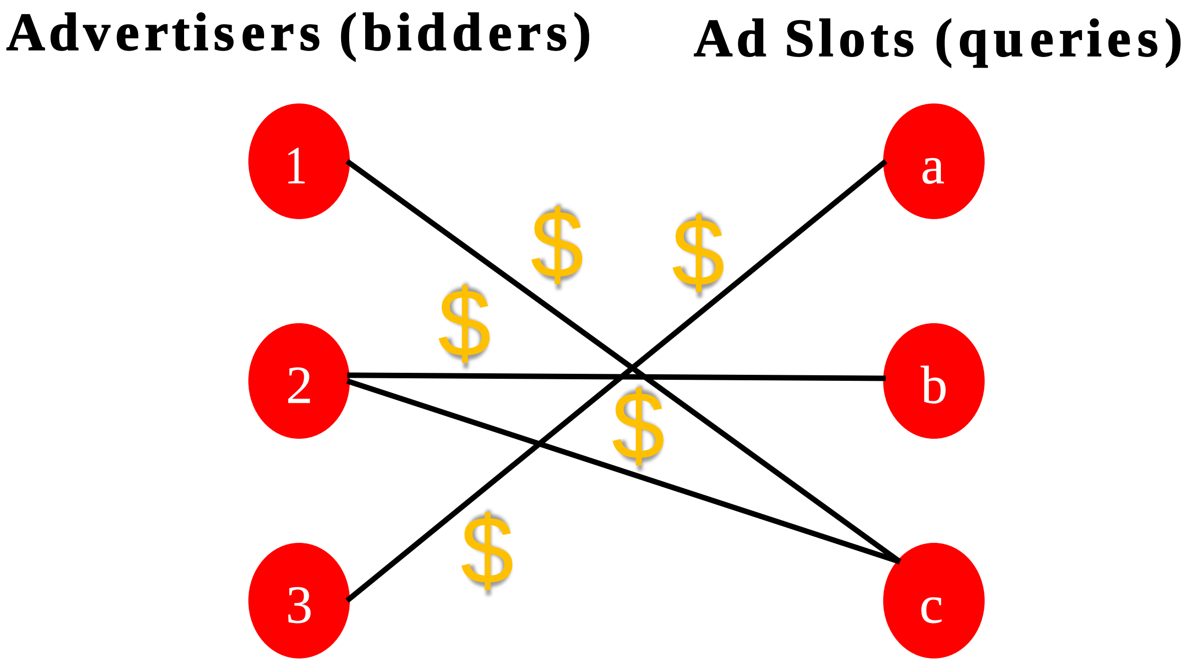
Graph Terminology

- Vertices 1 and c are the ***endpoints*** of edge $(1, c)$.
- Edges $(1, c)$, and $(2, c)$ are **incident** on vertex c.
- Vertices 1 and c are **adjacent**.
- The ***degree*** of vertex c (i.e., ***number of edges incident on vertex*** c) is 2.
- Edges may have weights (in this case, bid amounts).

Advertisers (bidders) **Ad Slots (queries)**



Adwords Problem



Find the best **MATCHING** of advertisers to ad slots that will maximize revenue.

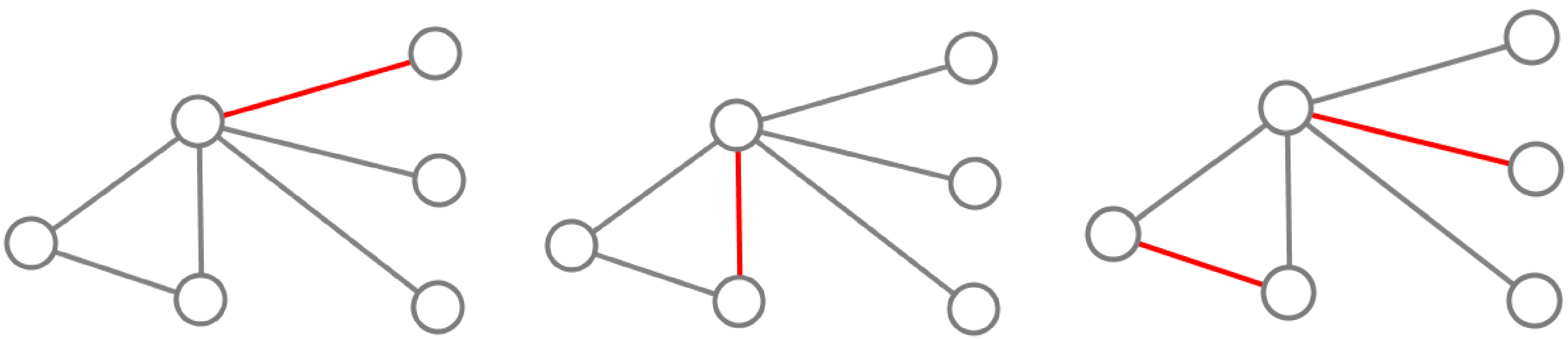
(constraint is that advertisers have a budget)

Terminology and Definitions

A **matching M** in a graph is a set of edges without common vertices.

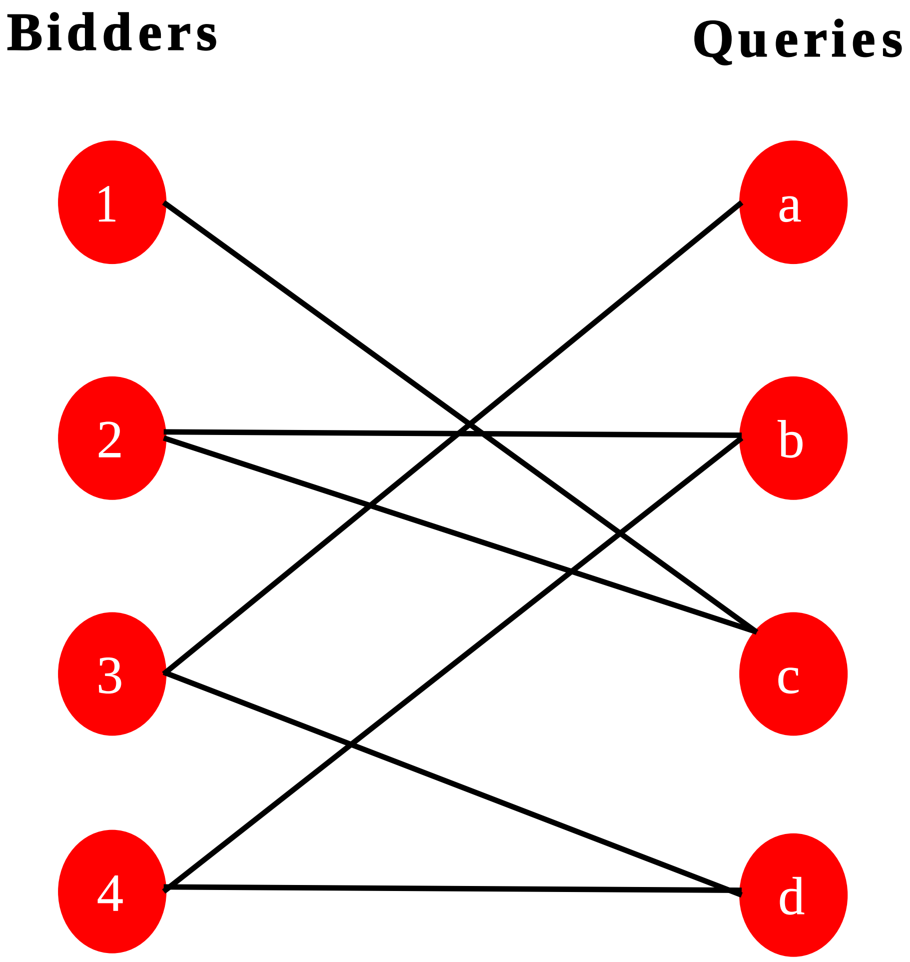
M is **maximal** if it is not a proper subset of any other matching in graph G

M is **maximum** if there is no larger matching than M in G



Not maximal **Maximal** **Maximum**

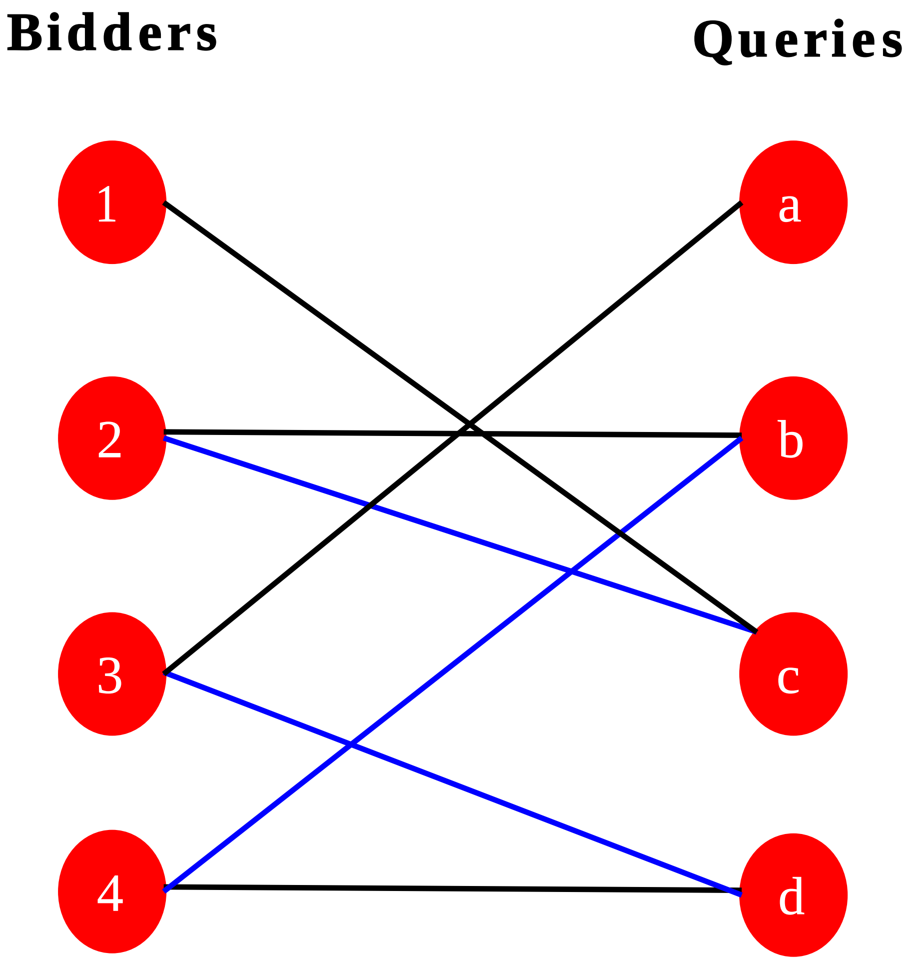
Bipartite Matching



Bipartite = only edges between bidders and queries

Goal: Maximize matching of bidders to search queries.

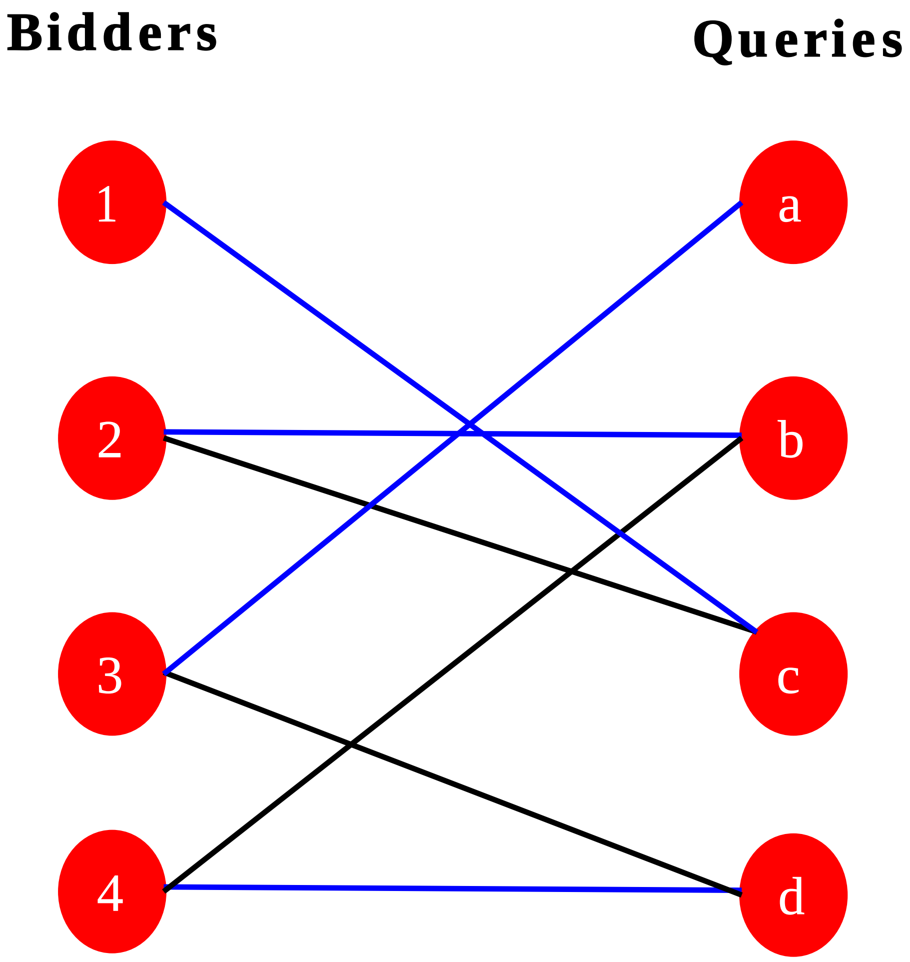
Bipartite Matching



$|M| = 3$

Cardinality of the matching is 3,
M = {(2,c), (3,d), (4,b)}

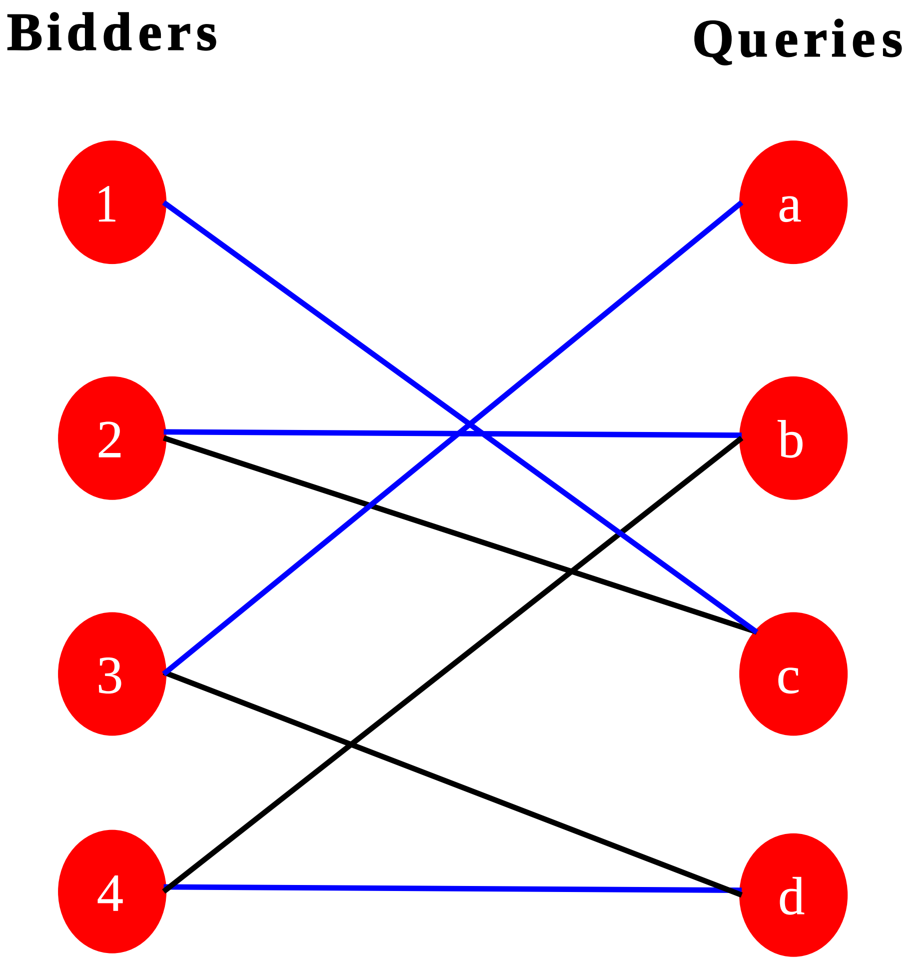
Bipartite Matching



$|M| = 4$

M is a **Perfect matching**

Bipartite Matching

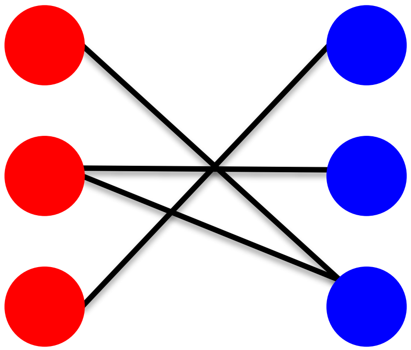


- **Hopcroft & Karp algorithm** finds maximum matching in polynomial time in a bipartite graph.
- $O(|E|\sqrt{|V|})$
- Algorithm is based on augmenting paths

What if the graph is dynamic?

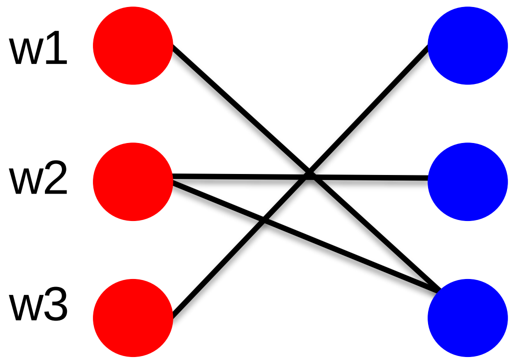
Online Matching Problems

Online Bipartite Matching



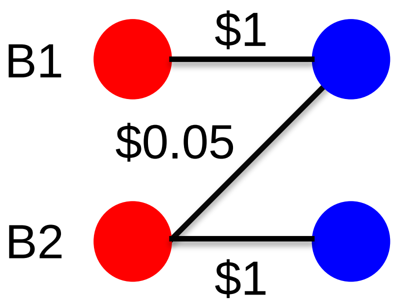
- Simplest problem
- No weights
- Maximize number of vertices that get matched

Vertex Weighted Matching



- Weights on LHS vertices
- Maximize sum of weights of LHS vertices that get matched

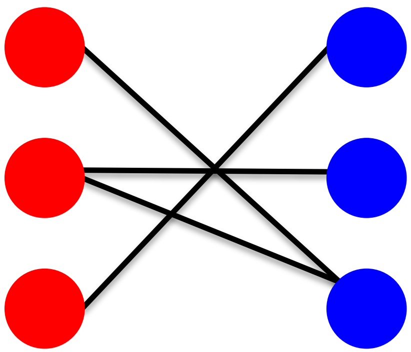
Adwords



- Budgets on LHS vertices (advertisers)
- Bids denoted by edge weights
- Maximize total amount of money spent by advertisers constrained by their budgets

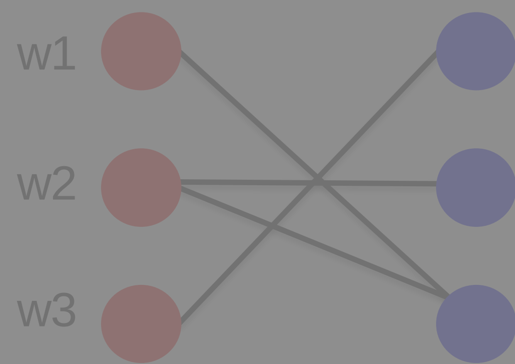
Online Matching Problems

Online Bipartite Matching



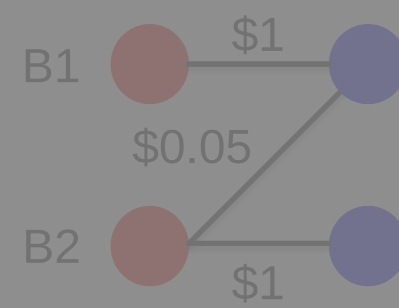
- Simplest problem
- No weights
- Maximize number of vertices that get matched

Vertex Weighted Matching



- Weights on LHS vertices
- Maximize sum of weights of LHS vertices that get matched

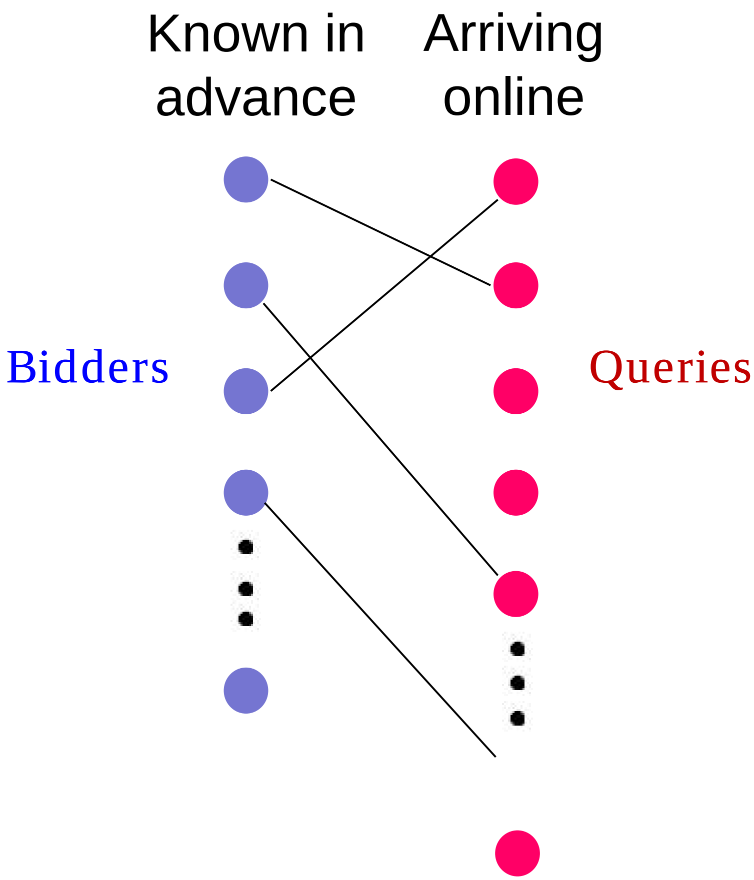
Adwords



- Budgets on LHS vertices (advertisers)
- Bids denoted by edge weights
- Maximize total amount of money spent by advertisers constrained by their budgets

Online Bipartite Matching

Example: Matching bidders to a incoming stream of search queries.



- Bidders known in advance.
- Stream of queries arrive online

- Decision to match a bidder to the arrived applicable query needs to be done on the spot

- Once two vertices are matched, it cannot be revoked

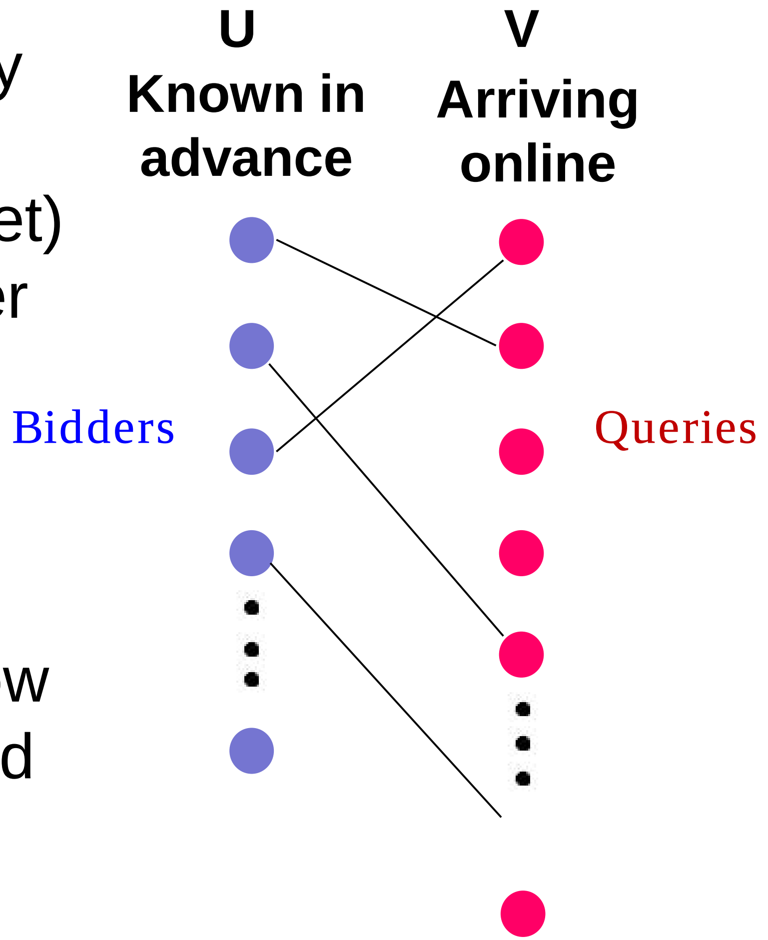
Online Bipartite Matching

We assume no knowledge of the query sequence V (RHS vertex set)

- No knowledge of V or E (edge set)
- No knowledge of the arrival order of V

The algorithm begins with only the knowledge of U (LHS vertex set)

- At any point in time, we only know the vertices in V that have arrived and the edges incident on them.



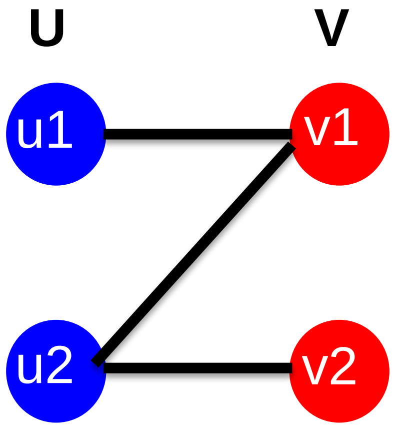
Greedy Algorithm

Greedy Algorithm:

For every arriving vertex $v \in V$

Match v to any available neighbor u in U (if any)

(break ties lexicographically)



Difficulty

Example:

- u1 and u2 are indistinguishable
- If u2 matches to v1,

then the next arriving vertex v2 will be left unmatched.

Greedy Algorithm

Competitive ratio = $\text{MIN}_{\text{all possible inputs}}(|M_{greedy}|/|M_{optimal}|)$

Claim:

Greedy has a competitive ratio of 1/2

Proof outline:

1. Show that any maximal matching is has a lower bound size of ½ the maximum (optimal) matching
2. Show that Greedy always produces a maximal matching
3. Show that ½ is also the worst-case upper bound.

Greedy Algorithm

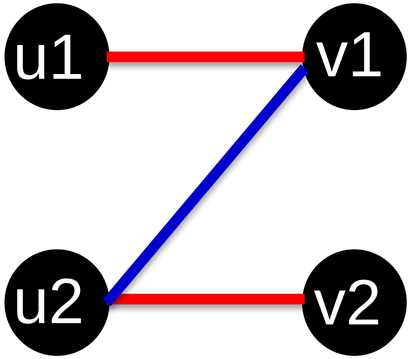
Lemma: If ***M*** is a maximal matching, and ***M**** a maximum matching, then |***M***| ≥ ½ | ***M**** |.

Proof:

- Take any edge (u, v) in M*
- Either u or v must be matched in M (otherwise M is not maximal)
- Therefore, the number of vertices matched in M, V(M), is at least half as many as in M*:

$$V(M) \geq \frac{1}{2} V(M^*).$$

- Note that V(M) = 2|M|

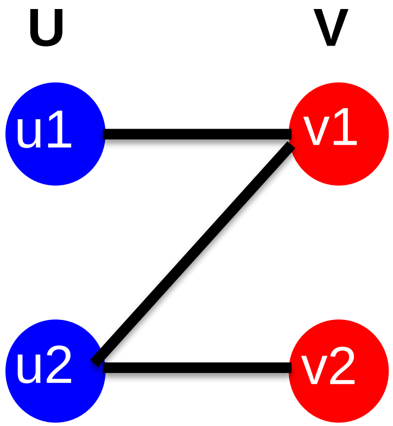


Greedy Algorithm

Lemma: Greedy always constructs a maximal matching

Proof:

- Let M be the matching produced by Greedy
- Suppose M is not maximal
- Then, there is an edge (u*,v*) that could be added to M and still be a matching
- Then, v* must have been unmatched
    - Implies u* was matched (contradiction)

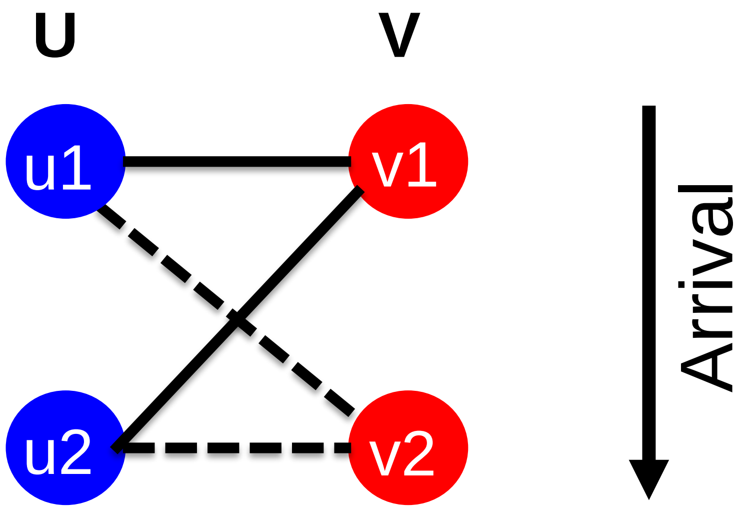


Greedy Algorithm

Lemma: Greedy (deterministic) has a worst-case upperbound of ½ the optimal solution

Proof:

- Consider the following graph:



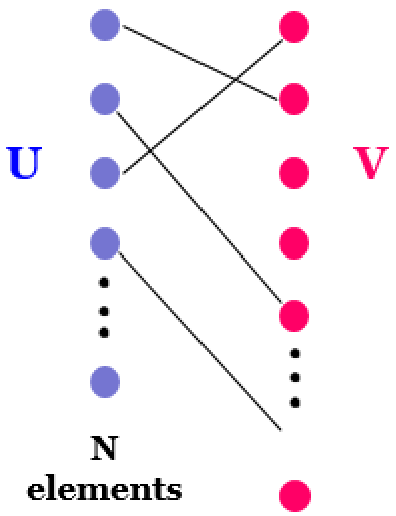
- v2 can be connected to u1 or u2, depending on how Greedy deterministically selects out of a tie.

Ranking Algorithm

- ***Karp, Vazirani and Vazirani [3]*** introduced the Ranking algorithm which uses a random partition of the vertices of the left partition.

Algorithm

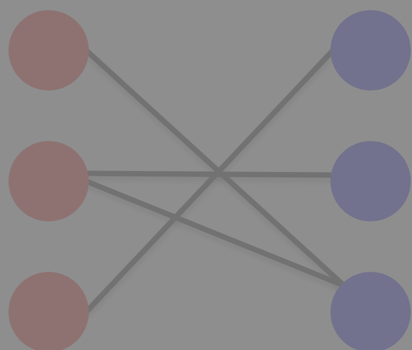
1. Create a random permutation π of the vertices on the LHS.
2. Match each incoming vertex in V to the available neighbor in U with the smallest value of $\pi(u)$.
3. If v has no available neighbors, it remains unmatched.



- This algorithm has an optimal ratio of **1 – 1/e**
- Interestingly, they also showed that no online algorithm can have a tighter bound than **1 – 1/e**.

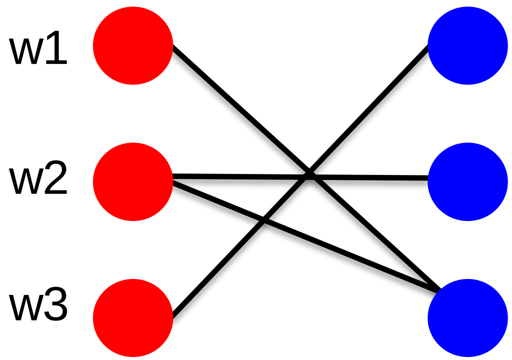
Online Matching Problems

Online Bipartite Matching



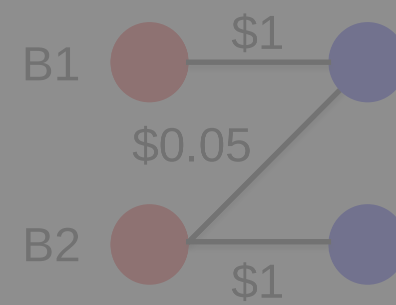
- Simplest problem
- No weights
- Maximize number of vertices that get matched

Vertex Weighted Matching



- Weights on LHS vertices
- Maximize sum of weights of LHS vertices that get matched

Adwords



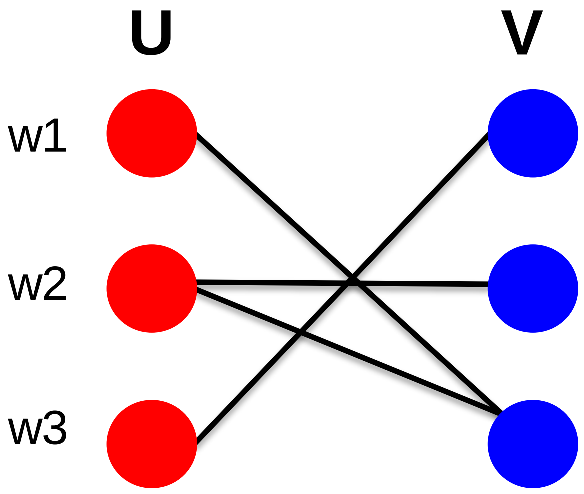
- Budgets on LHS vertices (advertisers)
- Bids denoted by edge weights
- Maximize total amount of money spent by advertisers constrained by their budgets

Vertex-Weighted Matching

Given $G(U,V,E)$,

- U is known in advance
- Each vertex $u \in U$ has a non-negative weight w_u, which is known in advance.
- $v \in V$ arrive online and reveal their neighbors in U, as before.

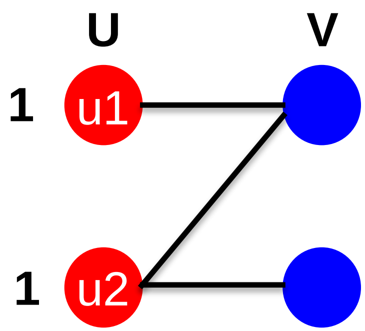
The goal is to **maximize the sum of weights** of vertices in U that get matched.



Vertex-Weighted Matching

Example 1

When u1 and u2 are equal weights,

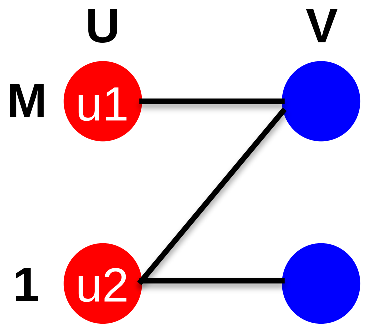


- Greedy achieves a ratio of 1/2.
- Ranking achieves a ratio of 3/4 (1-1/e in general).

Example 2

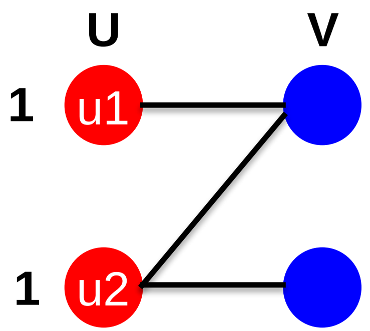
When u1 and u2 are different weights,

- Greedy achieves a ratio of 1.
- Ranking achieves a ratio of 1/2 (one can construct larger examples in which the ratio of Ranking goes to 0).



Vertex-Weighted Matching

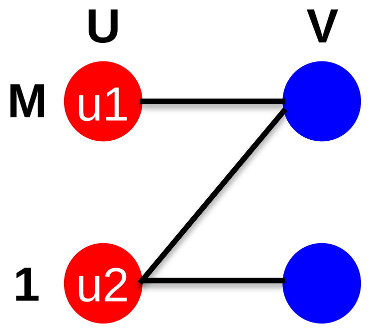
- **Ranking** works optimally for uniformly or near-uniformly weighted graphs,
- Fails badly when the vertex weights are highly skewed.



- **Greedy** does well for highly skewed weights
- But achieves ratio of only 1/2 when the weights are equal.

Solution

- Hybrid approach combining the two strategies



Vertex-Weighted Matching

A possible solution (hybrid approach)

Use **Greedy** on perturbed weights [8] (which is also a strict generalization of **Ranking**). Define a function,

$$\psi(x) = 1 - e^{x-1}$$

For each vertex v, assign a random value X_v from a uniform distribution [0, 1]

Instead of picking the highest weighted neighbor like in Greedy, pick the neighbor with the highest perturbed weight i.e.,

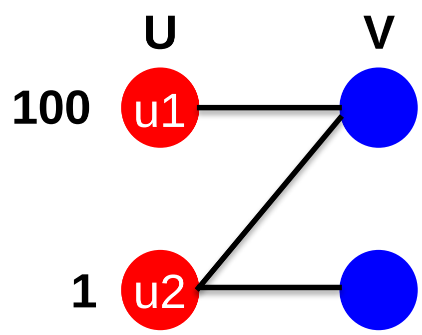
$$w_v * \psi(x_v)$$

Note,

- when the weights are highly skewed (say, if we have exponentially increasing weights) then the algorithms performs very similarly to **Greedy**.
- when all the weights are equal, then the algorithm is precisely **Ranking**

Vertex-Weighted Matching

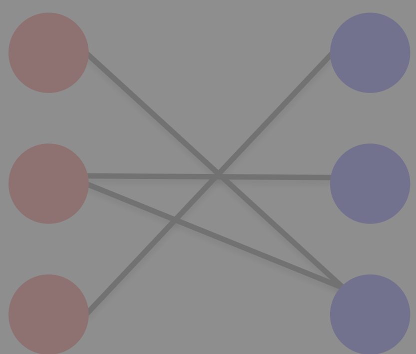
Perturbed weight = $w_v * \psi(x_v)$ $\psi(x) = 1 - e^{x-1}$



U	Xv	PERTURBED WEIGHT
u1	0.75	$\left(1 - e^{-0.25}\right) * 100 = 22.12$
u2	0.25	$\left(1 - e^{-0.75}\right) * 1 = 0.527$

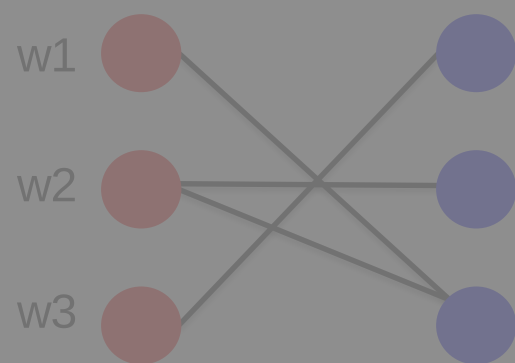
Online Matching Problems

Online Bipartite Matching



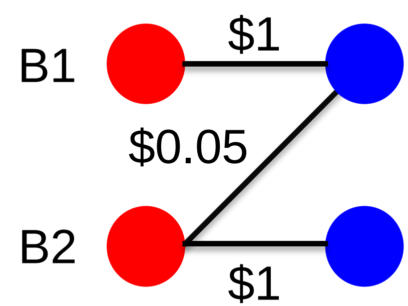
- Simplest problem
- No weights
- Maximize number of vertices that get matched

Vertex Weighted Matching



- Weights on LHS vertices
- Maximize sum of weights of LHS vertices that get matched

Adwords



- Budgets on LHS vertices (advertisers)
- Bids denoted by edge weights
- Maximize total amount of money spent by advertisers constrained by their budgets

Online Advertisement or Adwords

- Advertisers
  - bid on keywords (search queries)
  - specify maximum daily budget per queries
  - are charged when an ad is clicked on.
- Search engine earns when a user clicks on the ad.
  - Google's total advertising revenues were USD **$59.06** billion in 2014.
- Ads are ordered on the search page based on a revenue estimation per ad mechanism called **Ad Rank**.
  - Bids, Click Through Rate(CTR), ad relevance, Landing page experience, etc. all affect Ad Rank score.

Objective: Maximize the total revenue while respecting the daily budgets of the bidders.
Solution: Calculate expected value per bid (CTR*bid)

Simplification of the problem

(a) There is one ad shown for each query.

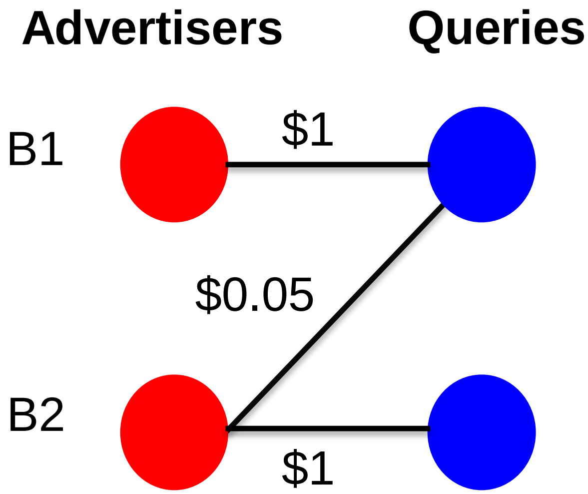
(b) All click-through rates are the same.

(c) All advertisers have the same budget.

(d) All bids are either 0 or 1.

Adwords Problem

- Given N advertisers/bidders
- Each advertiser has a fixed daily budget B_i
- Q is a set of known query keywords that the advertisers will bid on
- Each bidder specifies a bid for b_i for every query $q \in Q$.
- Sequence of q_1, q_2, q_3 .. , q_n arrive online
- Each query q_i must be assigned to an advertiser b_i without knowing the future bids.
- Objective is to maximize total revenue.



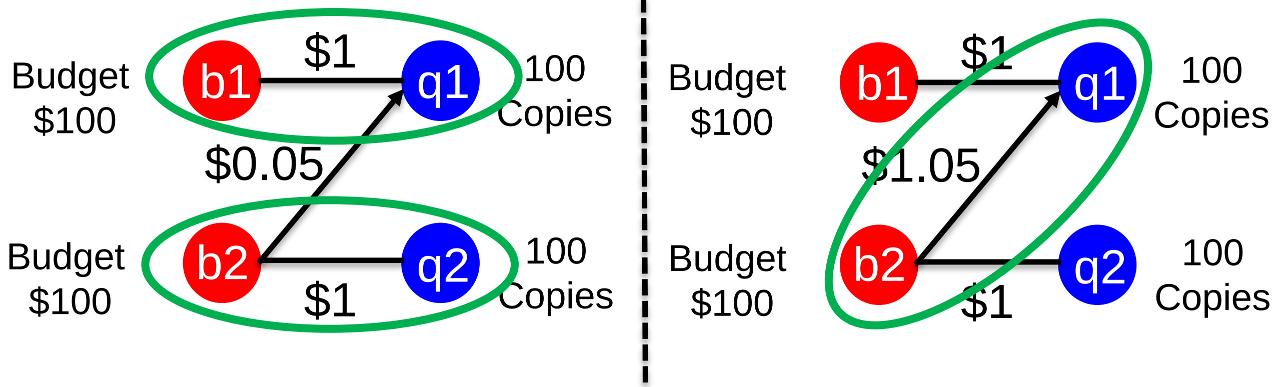
Competitive Ratio

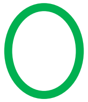
Competitive ratio is the minimum total revenue for an algorithm, on any sequence of search queries, divided by the revenue of the optimum off-line algorithm for the same sequence of search queries.

Competitive ratio = $\mathrm{MIN}_{\text{all possible inputs}}\left(|M_{greedy}|/|M_{optimal}|\right)$

Case 1: Ratio = 1

Case 2: Ratio = 1/2



 = matching

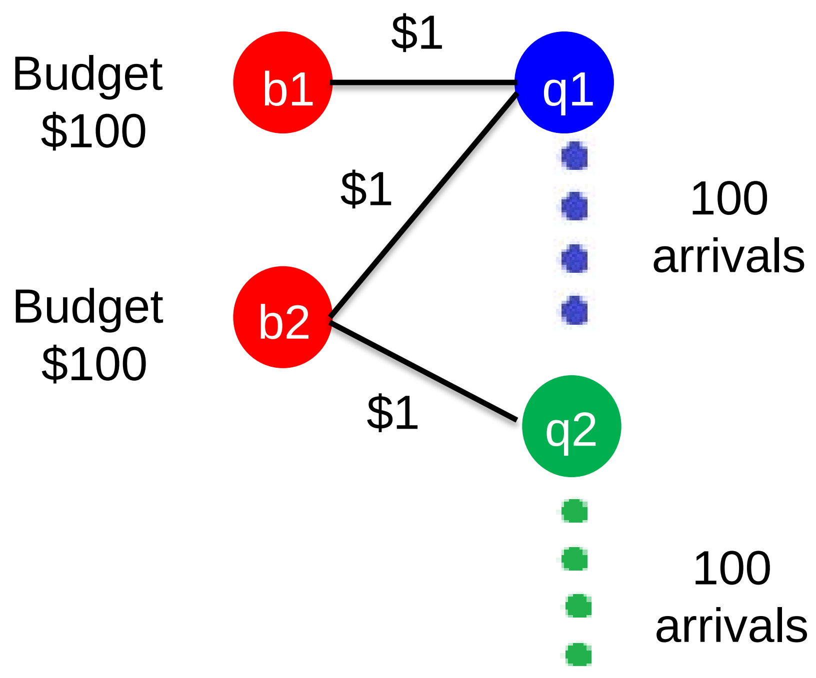
Greedy Matching

B1 bids on keyword **q1= shoes**

B2 bids on keyword **q2= beddings** and **q1**

Worst case: **100 b2** bids assigned to **q1**

Optimal: **100 b1s** assigned to **q1** followed by **100 b2s** assigned to **q2.**

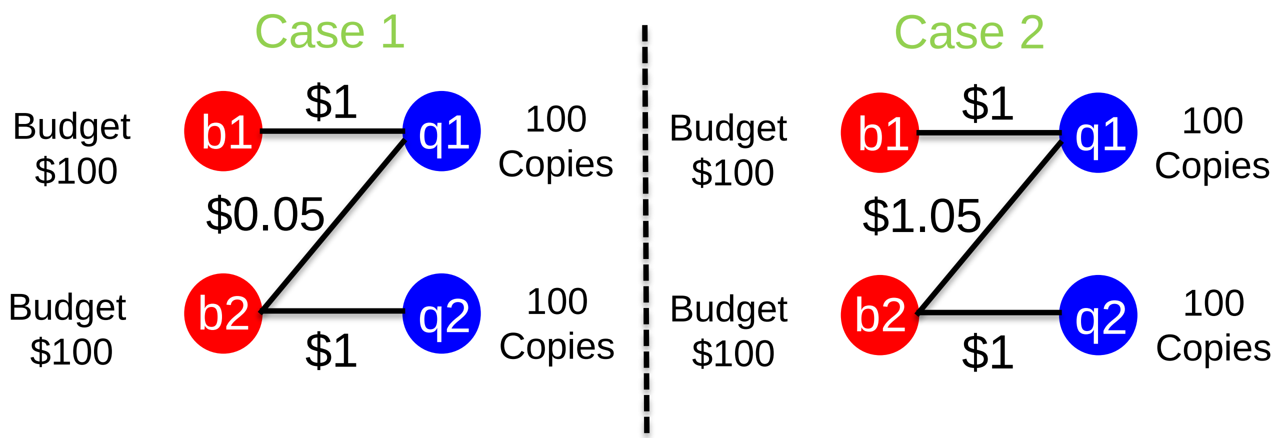


Competitive ratio = 1/2

Complications - Budgets

Honoring advertisers budget limit adds extra complications

- Estimating future search queries.
- Maximizing money spent by advertisers, while honoring budgets.



In case of simple greedy approach:

- Case 1 Bidder 1 and 2 will have both $100 spent.
- Case 2 Bidder 1 will have 0 dollars spent. Can we do better?

Balance Algorithm

- **Balance algorithm** [4] awards the query keyword to that interested advertiser ***who has the highest unspent budget***.

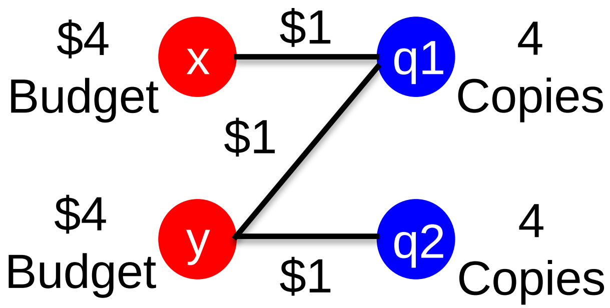
At first we discuss a specific case of the adwords problem:

- each advertiser has a daily budget of B dollars
- makes only **0/1 dollar** bids on each query.

With this assumptions, this algorithm has a competitive ratio of 1 – 1/e.

Balance Algorithm

- If four q1's arrive followed by four q2's
- Optimal choice would be **xxxxyyyy**
- Balance Algorithm will assign **xyxyyy,_,_**

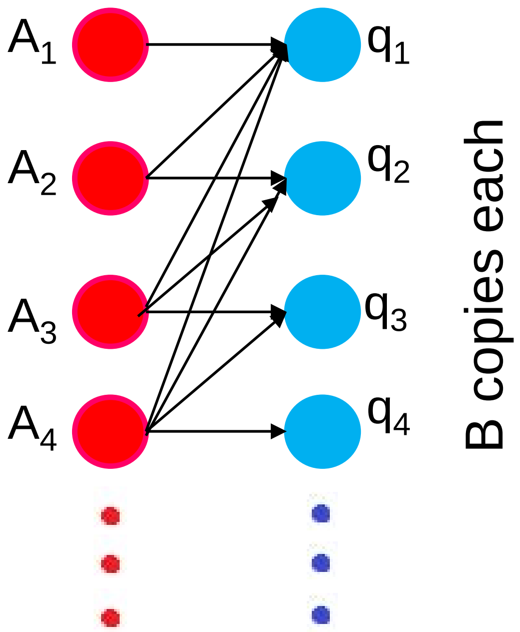


In this case we get a factor of 6/8=3/4 off the optimal solution. **In fact, for two advertisers, the competitive ratio for this algorithm is 3/4.**

Now we shall see that as the number of advertisers grows, the competitive ratio lowers to 0.63 (actually 1−1/e) but no lower.

Balance Algorithm, N bidders

When there are many advertisers, the competitive ratio for the Balance Algorithm can be under 3/4, but not below 1 – 1/e.



Worst-Case Situation:

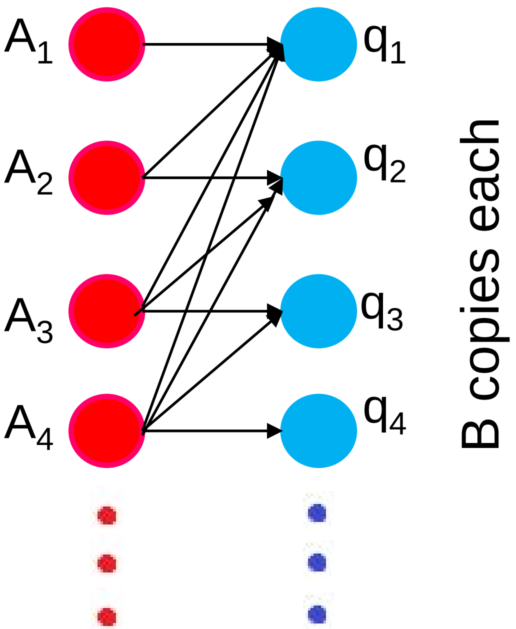
1. There are N advertisers, $A_1, A_2, \ldots, A_N$
2. Each advertiser has a budget B > N
3. There are N queries $q_1, q_2, \ldots, q_N$, each with B occurrences
4. Advertiser A_i bids on queries $q_1, q_2, \ldots, q_i$ and no other queries.

Optimal off-line algorithm assigns the B queries q_i in the i^{th} round to A_i for all *i*. **Optimal revenue N*B**

Balance Algorithm, N bidders

Balance Algorithm assigns each of the queries in round 1 to the N advertisers equally, because all bid on q_1.

- ***Round 1:*** Each advertiser gets *B/N* of the queries q1.
- ***Round 2:*** All but A_1 bids on these queries,
  B is divided among A_2 through A_N,
  N −1 bidders get B/(N −1) queries each.
- ***Round i*:** A_i through A_n get B/(N − i) queries each.



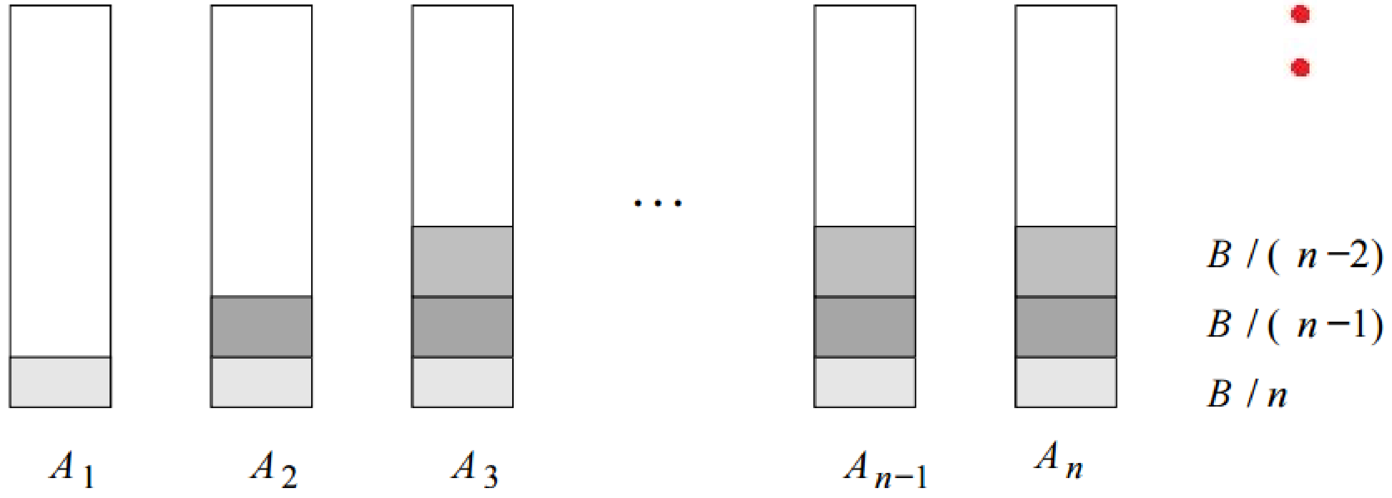
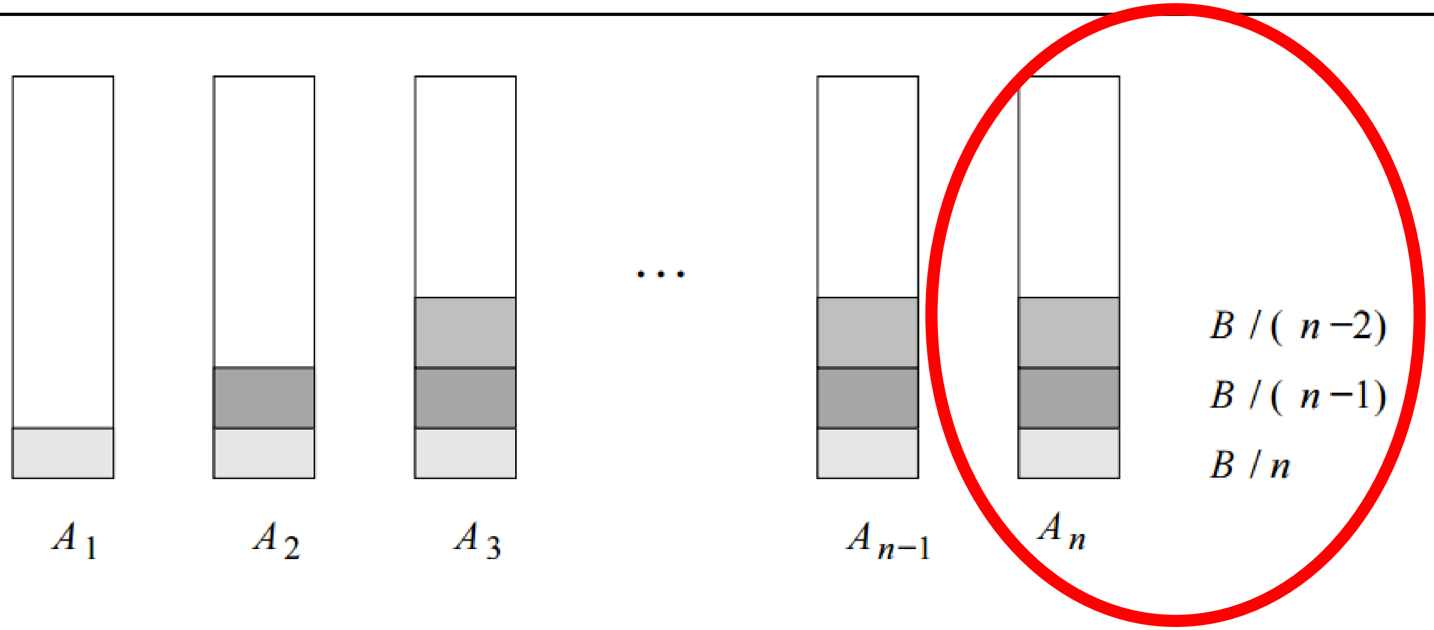


Image from: http://infolab.stanford.edu/~ullman/mmds/ch8a.pdf

Balance Algorithm, N bidders



Eventually, the budgets of the higher numbered advertisers will be exhausted, and no more queries will be able to be allocated in the subsequent rounds. This occurs at the lowest round j where:

$$B\left(\frac{1}{N}+\frac{1}{N-1}+\cdots+\frac{1}{N-j+1}\right)\geq B$$

$$\left(\frac{1}{N}+\frac{1}{N-1}+\cdots+\frac{1}{N-j+1}\right)\geq 1$$

Balance Algorithm, N bidders

Euler showed that as *N* gets large, $\sum_{i=1}^{N} 1/i$ approaches *log*$_e$.

$$\left(\frac{1}{N}+\frac{1}{N-1}+\cdots+\frac{1}{N-j+1}\right)=log_e N - loge(N-j)$$

$$= log_e \frac{N}{N-j}$$

$$log_e \frac{N}{N-j}=1 \Rightarrow \frac{N}{N-j}=e \Rightarrow \boldsymbol{j=N(1-1/e)}$$

$\Rightarrow$ ***Competitive ratio*** **=** BN(1 − 1/e)/BN = **1 – 1/e**

Generalized Balance Algorithm

- **Balance** works well when all bids are 0 or 1.
- It falls apart when bids and budgets are arbitrary.

Example issue with Balance Algorithm

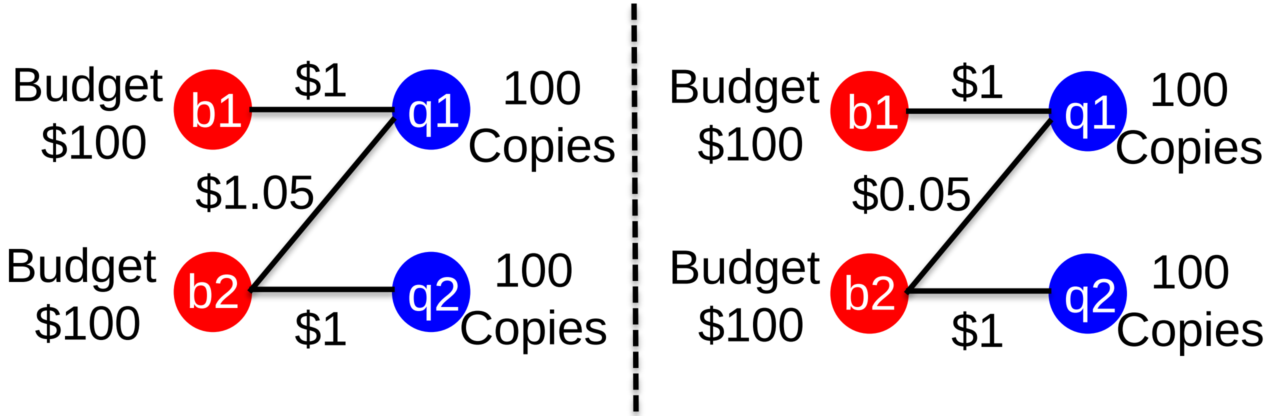
- There are two advertisers A_1 and A_2
- They both bid on one query *q*
- There are **10 occurrences of *q***

Bidder	Bid	Budget
A_1	$1	$110
A_2	$10	$100

- Balance will assign all the queries to A1 (since it has a larger budget), giving a revenue of $10
- Optimal algorithm would be $100 (assign all queries to A2)

Generalized Balance Algorithm [4]

Intuition A hybrid algorithm that combines these Greedy and Balance algorithms, so as to perform well in all instances.



Greedy: ~ 0.5
Balance: 0.75

Greedy: 1
Balance: ~0.5

Solution

- Bias the choice of ad in favor of higher bids.
- Scale the bid of an advertiser as a function of the fraction of its budget spent, instead of using the remaining budget.
- Run Greedy on the scaled bids

Generalized Balance Algorithm

Let

- x_u be the fraction of advertiser *u*'s budget that has been spent
- *v* be the next query to arrive.
- $\Psi(x_u) = 1 - e^{X_u - 1}$

The ***scaled bid*** of *u* for *v*, b_{uv}', *is defined as:*

$$b_{uv}\Psi(x_u)$$

Algorithm MSVV

When the next vertex $v \in V$ **arrives:**

Allocate *v* to the bidder *with the largest scaled bid*

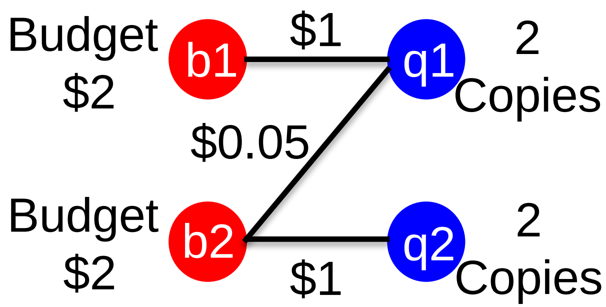
Proof of competitive ratio = 1 – 1/e is given in their paper [4]

MSVV Algorithm

Algorithm MSVV

When the next vertex $v \in V$ arrives:

Allocate v to the bidder *with the largest scaled bid*



Greedy: 1
Balance: ~0.5
MSVV: 1

		B1		B2	
TIME	Q	BID	SCALED BID	BID	SCALED BID
1	q1	\$1	$\mathbf{(1-e^{-1}) * \$1 = \$0.63}$	\$0.05	$(1-e^{-1}) * \$0.05 = \$0.03$
2	q1	\$1	$\mathbf{(1-e^{-0.5}) * \$1 = \$0.39}$	\$0.05	$(1-e^{-1}) * \$0.05 = \$0.03$
3	q2	-	-	1	$\mathbf{(1-e^{-1}) * \$1 = \$0.63}$
4	q2	-	-	1	$\mathbf{(1-e^{-0.5}) * \$1 = \$0.39}$

Summary

- The Adwords problem is modeled as a **bipartite graph matching problem**.
- Online (**unweighted) bipartite matching** is a special case of Adwords where all budgets are 1 and edges (bids) are 1.
- Online **vertex-weighted bipartite matching** is a special case of Adwords where the budget is spent on each ad slot.
- **Competitive ratio** is a common metric for evaluating a matching algorithms
- We discussed three algorithms for solving the adwords problem, the third **MSVV** being a hybrid of the first two (greedy and balanced) and having a competitive ratio of 1-1/e.

References

[1] J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

[2]. B. Kalyanasundaram and K.R. Pruhs, "An optimal deterministic algorithm for b-matching," Theoretical Computer Science 233:1–2, pp. 319 325, 2000.

[3] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, "An optimal algorithm for on-line bipartite matching," in *STOC*, pp. 352–358, 1990.

[4]. A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani, "Adwords and generalized online matching," *Journal of ACM*, vol. 54, no. 5, 2007.

[5]. Online Matching and Ad Allocation By Aranyak Mehta

[6]. J. Feldman, A. Mehta, V. S. Mirrokni, and S. Muthukrishnan, "Online stochastic matching: Beating 1-1/e," in *FOCS*, pp. 117–126, 2009.

[7]. B. Haeupler, V. Mirrokni, and M. Zadimoghaddam, "Online stochastic weighted matching: Improved approximation algorithms," *Internet and Network Economics*, pp. 170–181, 2011.

[8] G. Aggarwal, G. Goel, C. Karande, and A. Mehta, "Online vertex-weighted bipartite matching and single-bid budgeted allocations," in *SODA*, pp. 1253–1264, 2011.