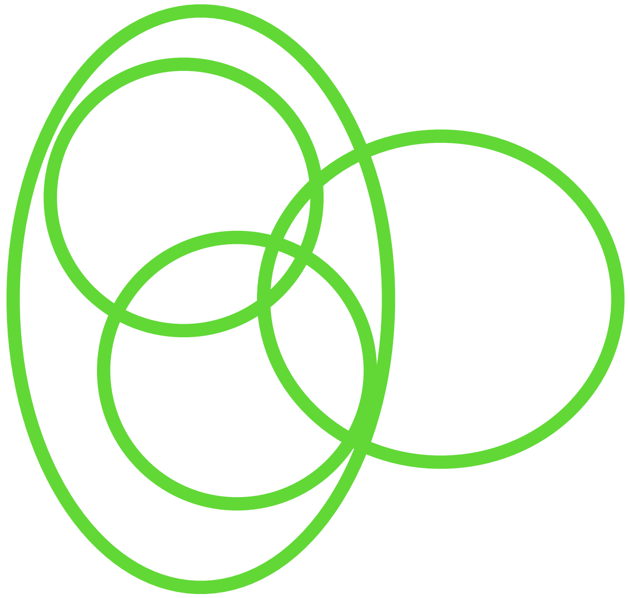
Foresight in submodular optimization

Yuval Filmus, Technion

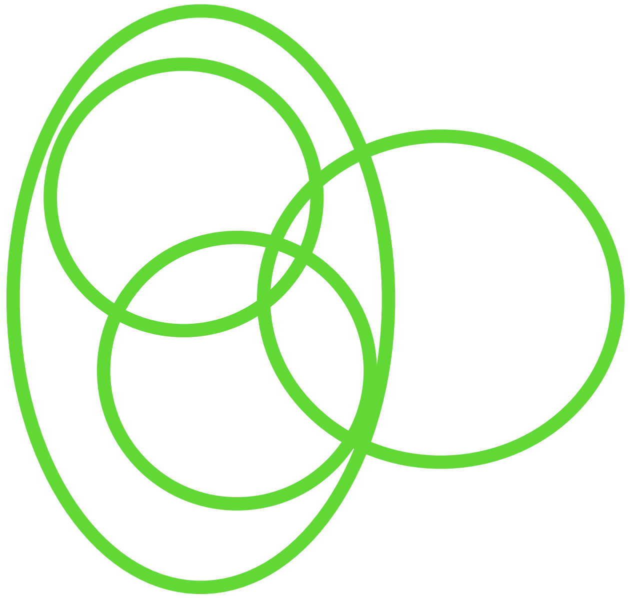
This is not Uri!

Set Cover



How many sets needed to cover entire universe?

Set Cover

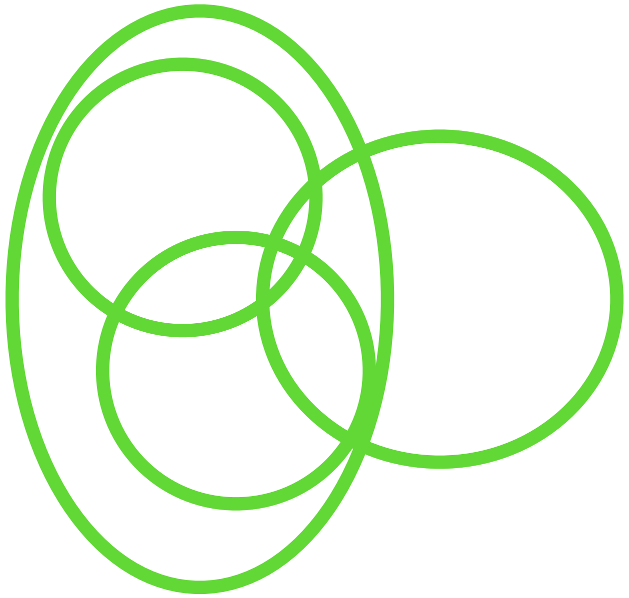


How many sets needed to cover entire universe?

Greedy algorithm:
Repeatedly choose set covering maximum number of new elements

ln n approximation

Set Cover



How many sets needed to cover entire universe?

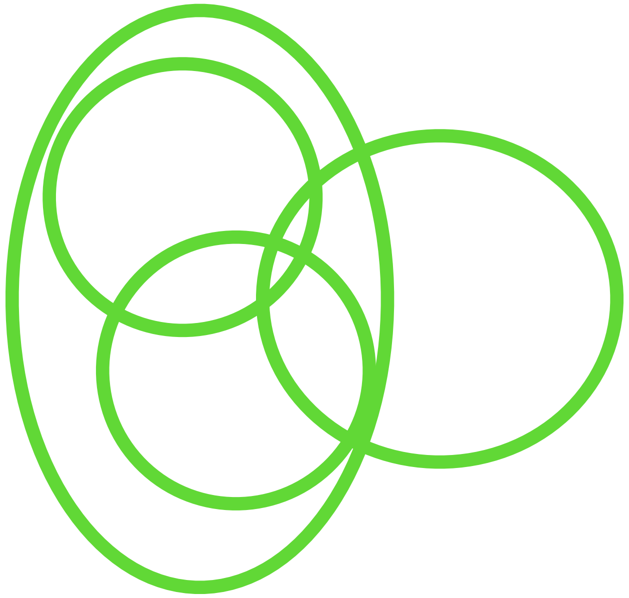
Greedy algorithm:
Repeatedly choose set covering maximum number of new elements

ln *n* approximation

Feige (JACM'98):
Optimal unless NP$\subseteq$TIME($n^{O(\log\log n)}$)

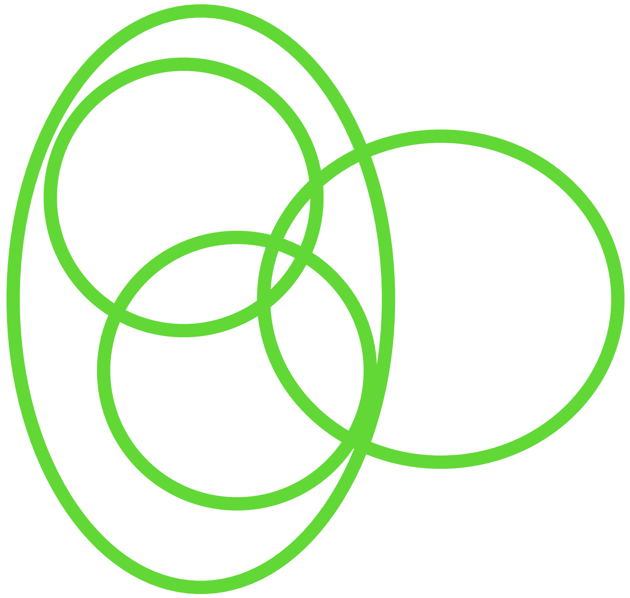
D. Moshkovitz (2014):
Optimal unless P=NP

Max *k*-Cover



How many elements can cover using *k* sets?

Max *k*-Cover

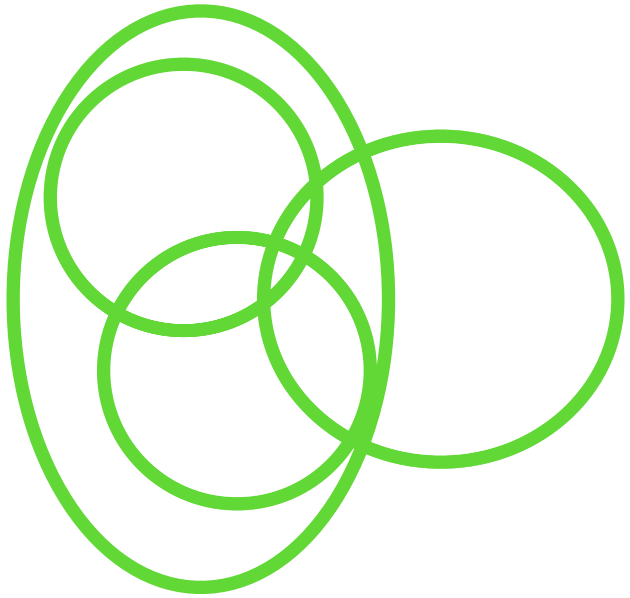


How many elements can cover using *k* sets?

Greedy algorithm:
Repeatedly choose set covering maximum number of new elements

1–1/*e* approximation
(0.632)

Max *k*-Cover



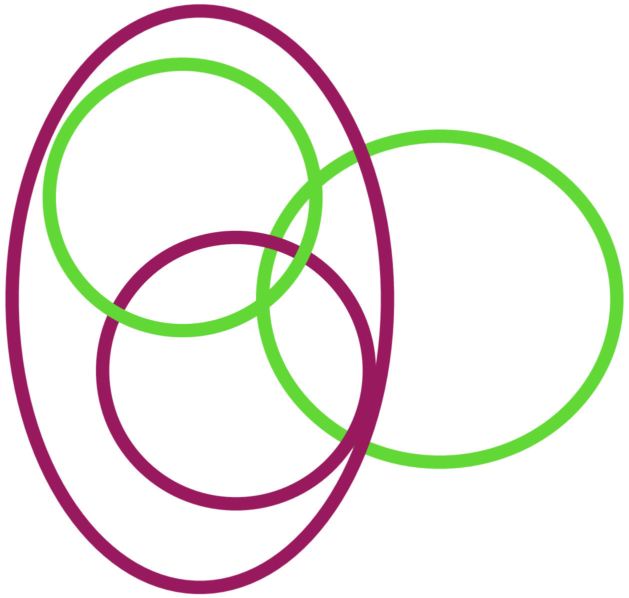
How many elements can cover using *k* sets?

Greedy algorithm:
Repeatedly choose set covering maximum number of new elements

1–1/*e* approximation
(0.632)

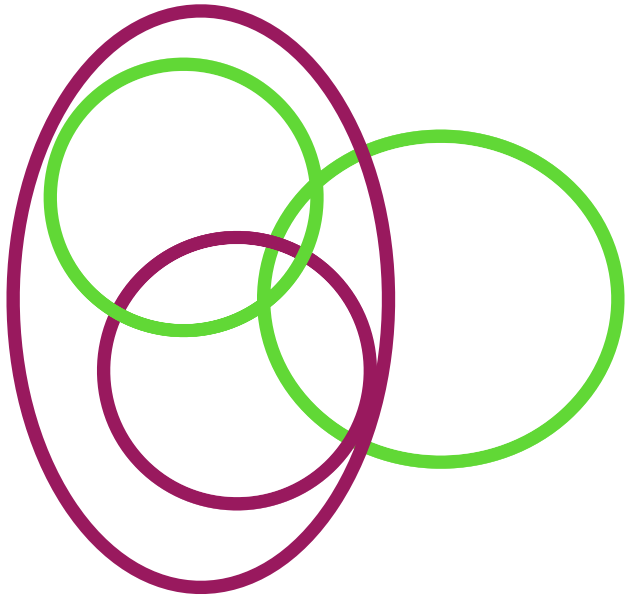
Feige (JACM'98):
Optimal unless P=NP

Partition Max *k*-Cover



How many elements can cover using *k* sets, one of each color?

Partition Max *k*-Cover

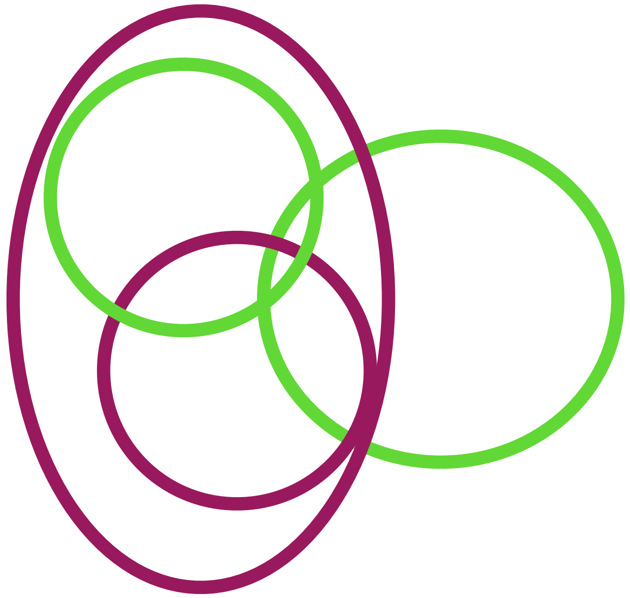


How many elements can cover using *k* sets, one of each color?

Greedy algorithm:
Repeatedly choose set covering maximum number of new elements

1/2 approximation

Partition Max *k*-Cover



How many elements can cover using *k* sets, one of each color?

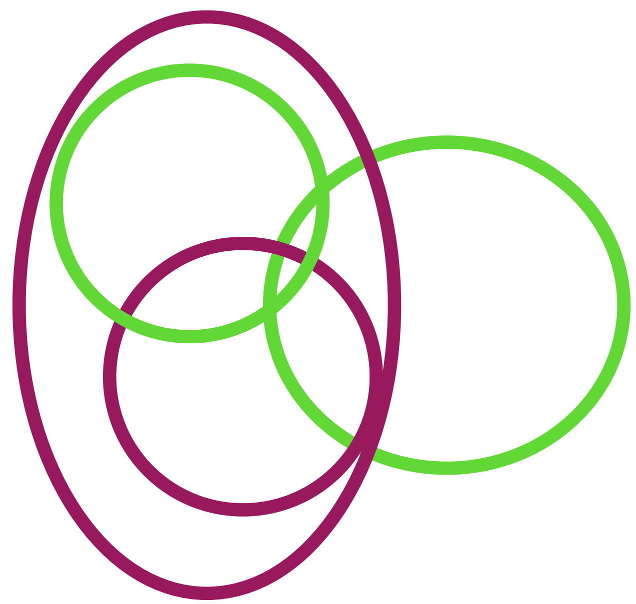
Greedy algorithm:
Repeatedly choose set covering maximum number of new elements

1/2 approximation

Generalizes Max SAT:

- Color: variable
- Set: truth assignment
- Element: clause

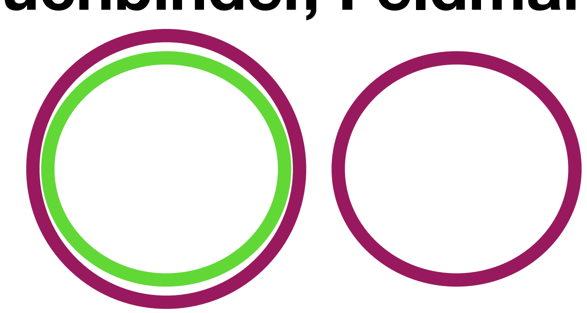
Partition Max *k*-Cover



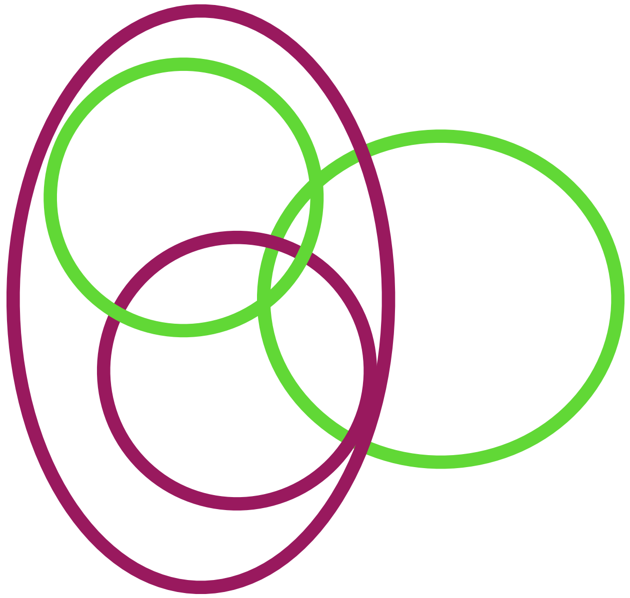
How many elements can cover using *k* sets, one of each color?

Random order greedy:
Choose random order of colors.
Repeatedly choose set covering maximum number of new elements.

≥0.5096 approximation
[Buchbinder, Feldman, F, Garg'19]



Partition Max *k*-Cover



How many elements can cover using *k* sets, one of each color?

Random order greedy:
Choose random order of colors.
Repeatedly choose set covering maximum number of new elements.

≥0.5096 approximation
[Buchbinder, Feldman, F, Garg'19]

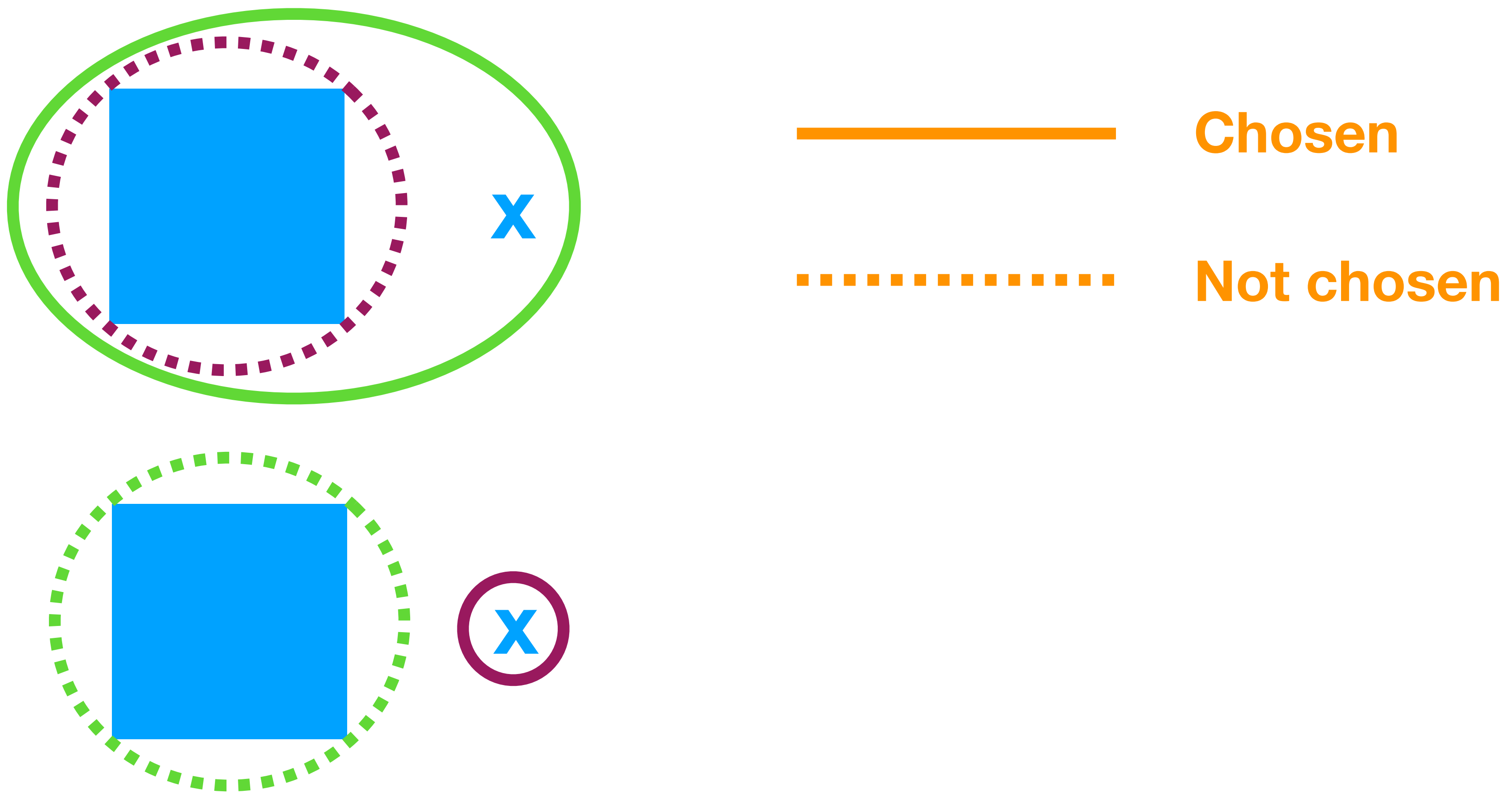
Srinivasan '01,
Ageev, Sviridenko '04:
LP relaxation

Calinescu, Chekuri, Pál, Vondrák '09:
Continuous greedy

1–1/*e* approximation

Not combinatorial!

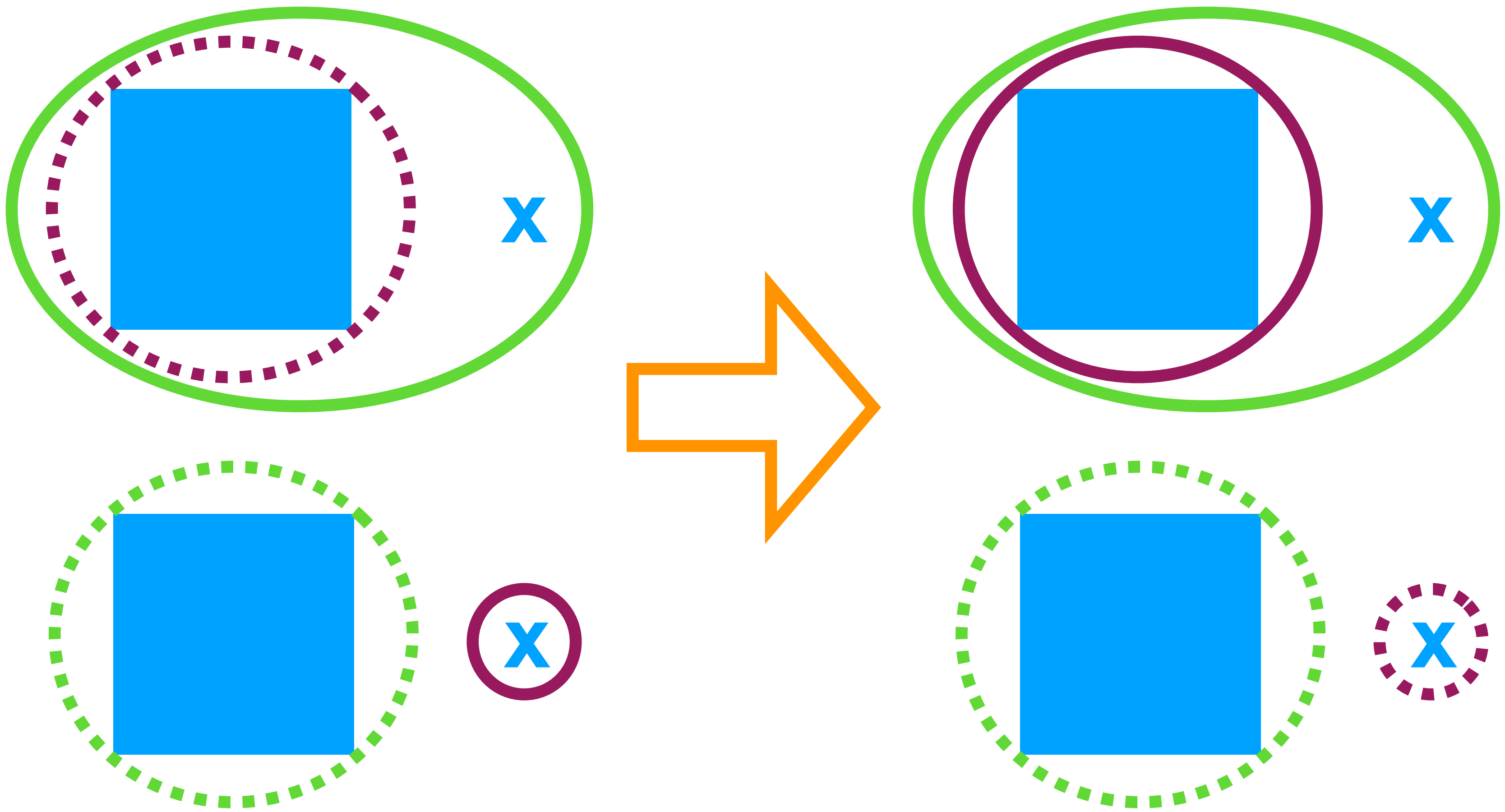
Local search?



Locally optimal!

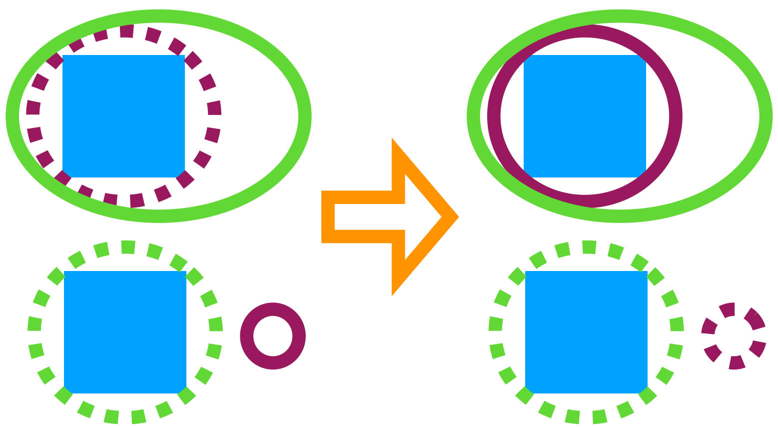
1/2 approximation

Local search?

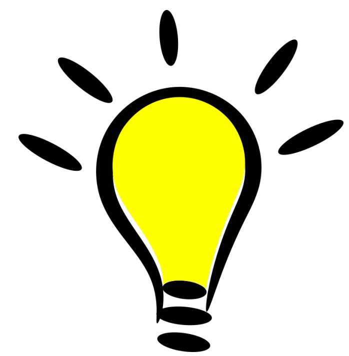


Loss: Cover fewer elements
Gain: Upper square covered twice

Local search!

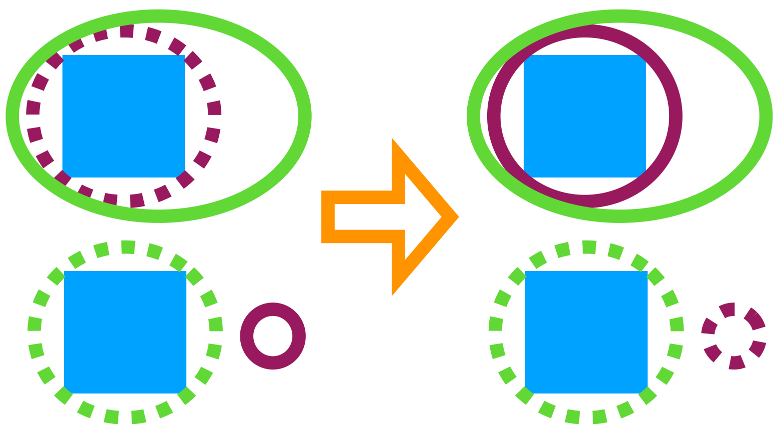


Loss: Cover fewer elements
Gain: Upper square covered twice

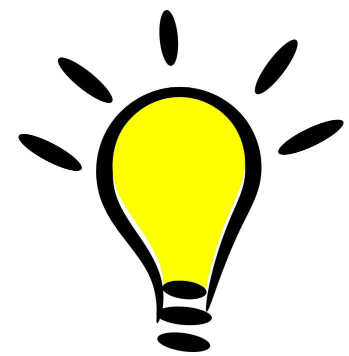


Idea:
Give more weight to elements covered multiple times

Local search!



Loss: Cover fewer elements
Gain: Upper square covered twice



Idea:
Give more weight to elements covered multiple times

F, Ward (2012):
Optimal choice of weights
➜ 1–1/*e* approximation!

Optimal weights found by solving infinite LP

Multiplicity	Weight
0	0
1	1
2	1.418
3	1.672
4	1.852
5	1.991
k	$\approx C \log k$

$$\alpha_{k+1} = (k+1)\alpha_k - k\alpha_{k-1} - \frac{1}{e-1}$$

Analyzing local search

Setup:

- **Local optimum** $S_1,\ldots,S_k$
- **Global optimum** $O_1,\ldots,O_k$

Switching S_i and O_i doesn't improve objective function $cost \Rightarrow$

$$\sum_{i=1}^{k} cost(S_1, \ldots, O_i, \ldots, S_k) \leq k \cdot cost(S_1, \ldots, S_k) \qquad \text{(local optimality)}$$

Analyzing local search

Setup:

- **Local optimum** $S_1,\dots,S_k$
- **Global optimum** $O_1,\dots,O_k$

Switching S_i and O_i doesn't improve objective function $cost \Rightarrow$

$$\sum_{i=1}^{k} cost(S_1,\dots,O_i,\dots,S_k) \le k \cdot cost(S_1,\dots,S_k) \qquad \textbf{(local optimality)}$$

$N(a,b,c)$ – # elements appearing:
a times in $S_1,\dots,S_k$
b times in $O_1,\dots,O_k$
c times in $S_1 \cap O_1,\dots,S_k \cap O_k$

Given weights, can write LP for approximation ratio in variables $N(a,b,c)$:

min $|S_1 \cup \cdots \cup S_k|$ *s.t.* $|O_1 \cup \cdots \cup O_k|=1$ and "local optimality"

Analyzing local search

Setup:

- **Local optimum** $S_1,\ldots,S_k$
- **Global optimum** $O_1,\ldots,O_k$

Switching S_i and O_i doesn't improve objective function $cost \Rightarrow$

$$\sum_{i=1}^{k} cost(S_1, \ldots, O_i, \ldots, S_k) \leq k \cdot cost(S_1, \ldots, S_k) \qquad \textbf{(local optimality)}$$

***N*(*a*,*b*,*c*) – # elements appearing:**
a **times in** $S_1,\ldots,S_k$
b **times in** $O_1,\ldots,O_k$
c **times in** $S_1 \cap O_1,\ldots,S_k \cap O_k$

Given weights, can write LP for approximation ratio in variables *N*(*a*,*b*,*c*):

$$\min |S_1 \cup \cdots \cup S_k| \ \textit{s.t.}\ |O_1 \cup \cdots \cup O_k| = 1 \text{ and "local optimality"}$$

Can find optimal weights using dual LP

Matroids

Setup:

- **Local optimum** $S_1, \ldots, S_k$
- **Global optimum** $O_1, \ldots, O_k$

Switching S_i **and** O_i **doesn't improve objective function** $cost \Rightarrow$

$$\sum_{i=1}^{k} cost(S_1, \ldots, O_i, \ldots, S_k) \leq k \cdot cost(S_1, \ldots, S_k)$$ **(local optimality)**

Matroids

Setup:

- **Local optimum** $S_1,\ldots,S_k$
- **Global optimum** $O_1,\ldots,O_k$

Switching S_i and O_i doesn't improve objective function $cost \Rightarrow$

$$\sum_{i=1}^{k} cost(S_1,\ldots,O_i,\ldots,S_k) \leq k \cdot cost(S_1,\ldots,S_k) \qquad \textbf{(local optimality)}$$

Brualdi's lemma:
If $S_1,\ldots,S_k$ and $O_1,\ldots,O_k$ are two bases in an arbitrary matroid, can rearrange indices so that $S_1,\ldots,O_i,\ldots,S_k$ is a basis for all i

Matroids

Setup:

- **Local optimum** $S_1, \ldots, S_k$
- **Global optimum** $O_1, \ldots, O_k$

Switching S_i and O_i doesn't improve objective function $cost \Rightarrow$

$$\sum_{i=1}^{k} cost(S_1, \ldots, O_i, \ldots, S_k) \leq k \cdot cost(S_1, \ldots, S_k) \qquad \textbf{(local optimality)}$$

Brualdi's lemma:
If $S_1, \ldots, S_k$ and $O_1, \ldots, O_k$ are two bases in an arbitrary matroid, can rearrange indices so that $S_1, \ldots, O_i, \ldots, S_k$ is a basis for all i

Conclusion: algorithm works for arbitrary matroids

Submodular functions

$f(A)$=number of elements covered by sets in A

Submodular functions

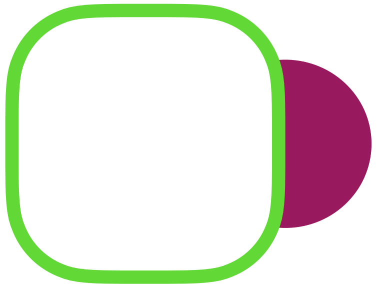
$f(A)$=number of elements covered by sets in A

$f(\varnothing) = 0$ **Normalization**

Submodular functions

f*(*A*)=number of elements covered by sets in *A

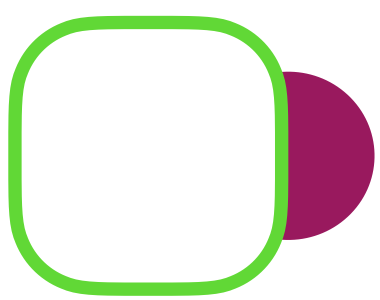
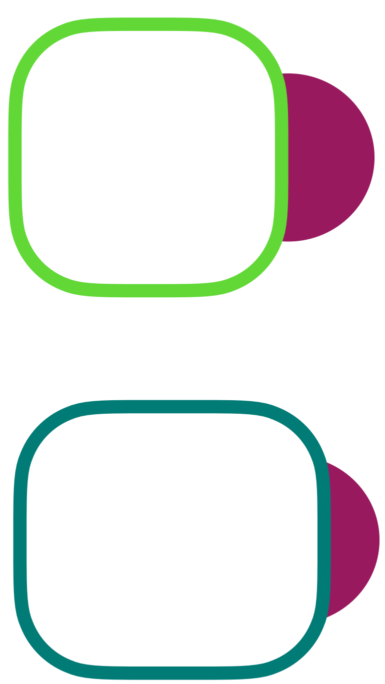
$f(\varnothing) = 0$ **Normalization**



$f(A+x) \geq f(A)$ **Monotonicity**

Submodular functions

$f(A)$=number of elements covered by sets in A

	$f(\emptyset) = 0$	**Normalization**
	$f(A+x) \geq f(A)$	**Monotonicity**
	**If** $A \subseteq B$ **then** $f(A+x) - f(A) \geq f(B+x) - f(B)$	**Submodularity**

Submodular functions

Alternative definitions:

- **Normalization:** $f(\varnothing)=0$
- **Monotonicity:** $f(A+x)-f(A)\geq 0$ $\qquad \partial_x f \geq 0$
- **Submodularity:** $f(A+x+y)-f(A+x)-f(A+y)+f(A)\leq 0$ $\qquad \partial_x \partial_y f \leq 0$

  $f(A)+f(B)\geq f(A\cup B)+f(A\cap B)$

Submodular functions

Alternative definitions:

- **Normalization:** $f(\emptyset)=0$
- **Monotonicity:** $f(A+x)-f(A)\geq 0$ $\qquad \partial_x f \geq 0$
- **Submodularity:** $f(A+x+y)-f(A+x)-f(A+y)+f(A)\leq 0$ $\qquad \partial_x \partial_y f \leq 0$

  $f(A)+f(B)\geq f(A\cup B)+f(A\cap B)$

Coverage functions characterized by:

$$\partial_{x_1}\cdots\partial_{x_k} f \begin{cases} \geq 0 & \text{if } k \text{ odd} \\ \leq 0 & \text{if } k \text{ even} \end{cases}$$

Many algorithms for coverage functions only need condition on first two derivatives

Submodular functions

"Max *k*-cover":
Given monotone submodular *f*,
maximize *f*(*A*) over |*A*|=*k*

Greedy algorithm:
Repeatedly add elements
maximizing value of *f*

1–1/*e* approximation

Submodular functions

“Max *k*-cover”:
Given monotone submodular *f*,
maximize *f*(*A*) over |*A*|=*k*

Greedy algorithm:
Repeatedly add elements
maximizing value of *f*

1–1/*e* approximation

“Partition Max *k*-cover”:
Given monotone submodular *f*,
maximize *f*(*A*) over sets *A*
containing one element
of each color (*k* colors)

Greedy: 1/2
Random order greedy: ≥0.5096
Continuous greedy: 1–1/*e*

Submodular functions

"Max k-cover":
Given monotone submodular f,
maximize $f(A)$ over $|A|=k$

Greedy algorithm:
Repeatedly add elements
maximizing value of f

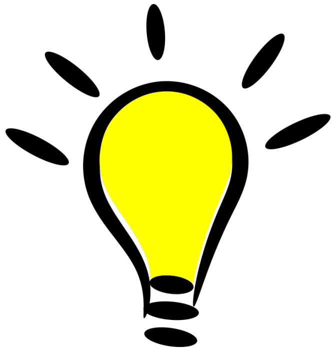
$1-1/e$ approximation

"Partition Max k-cover":
Given monotone submodular f,
maximize $f(A)$ over sets A
containing one element
of each color (k colors)

Greedy: 1/2
Random order greedy: ≥0.5096
Continuous greedy: $1-1/e$

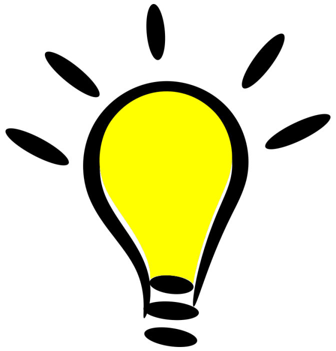
Can we generalize our local search algorithm?

Submodular functions



Idea:
Formulate $cost$**(*A*) in terms of *f*(*B*) for *B*⊆*A***

Submodular functions

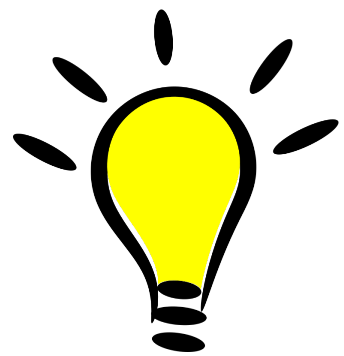


Idea:
Formulate $cost$(*A*) in terms of *f*(*B*) for $B \subseteq A$

Let $f_p(A)$=E[*f*(*B*)], where *B* is chosen by sampling each element of *A* w.p. *p*

De Finetti's theorem: $cost$ is mixture of f_p

Submodular functions



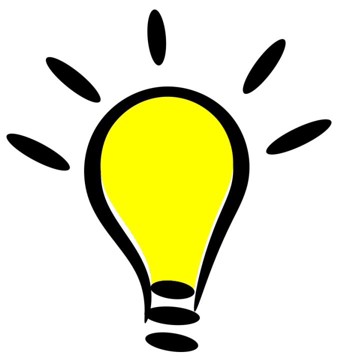
Idea:
Formulate $cost$(*A*) in terms of *f*(*B*) for *B*⊆*A*

Let $f_p(A)$=E[*f*(*B*)], where *B* is chosen by sampling each element of *A* w.p. *p*

De Finetti's theorem: $cost$ is mixture of f_p

$$cost(A) = \frac{1}{e-1}\int_0^1 \frac{e^p}{p} f_p(A)\,\mathrm{d}p \qquad \partial_x cost(A) = \int_0^1 \frac{e^p}{e-1} (\partial_x f)_p(A)\,\mathrm{d}p$$

Submodular functions



Idea:
Formulate $cost$(*A*) in terms of *f*(*B*) for *B*⊆*A*

Let $f_p(A)$=E[*f*(*B*)], where *B* is chosen by sampling each element of *A* w.p. *p*

De Finetti's theorem: $cost$ is mixture of f_p

$$cost(A) = \frac{1}{e-1}\int_0^1 \frac{e^p}{p} f_p(A)\,\mathrm{d}p \qquad \partial_x cost(A) = \int_0^1 \frac{e^p}{e-1}(\partial_x f)_p(A)\,\mathrm{d}p$$

F, Ward (2012): 1–1/*e* approximation for any monotone submodular *f*!

Continuous greedy

Vondrák (2008)

Competing algorithm for submodular max k-cover:

Maintain feasible solution A
At time $p \in [0,1]$, for each color i, at rate $1/p$:
Update color i with x maximizing $(\partial_x f)_p(A)$

Continuous greedy

Vondrák (2008)

Competing algorithm for submodular max *k*-cover:

Maintain feasible solution *A*
At time $p \in [0,1]$, for each color *i*, at rate $1/p$:
Update color *i* with *x* maximizing $(\partial_x f)_p(A)$

Our cost function considers all times at once:

$$cost(A) = \frac{1}{e-1}\int_0^1 \frac{e^p}{p} f_p(A)\,\mathrm{d}p \qquad \partial_x cost(A) = \int_0^1 \frac{e^p}{e-1}(\partial_x f)_p(A)\,\mathrm{d}p$$

Continuous greedy

Vondrák (2008)

Competing algorithm for submodular max *k*-cover:

Maintain feasible solution *A*
At time $p \in [0,1]$, for each color *i*, at rate $1/p$:
Update color *i* with *x* maximizing $(\partial_x f)_p(A)$

Our cost function considers all times at once:

$$cost(A) = \frac{1}{e-1} \int_0^1 \frac{e^p}{p} f_p(A) \, \mathrm{d}p \qquad \partial_x cost(A) = \int_0^1 \frac{e^p}{e-1} (\partial_x f)_p(A) \, \mathrm{d}p$$

Exact connection between the two algorithms is still a mystery!

Open problems

- Deterministic algorithm?

Open problems

- Deterministic algorithm?
- Optimization over perfect matchings in bipartite graph? (More generally, intersection of two or more matroids)

Open problems

- Deterministic algorithm?
- Optimization over perfect matchings in bipartite graph? (More generally, intersection of two or more matroids)
- Tight analysis of random order greedy

Open problems

- Deterministic algorithm?
- Optimization over perfect matchings in bipartite graph?
  (More generally, intersection of two or more matroids)
- Tight analysis of random order greedy
- Random order greedy with foresight?
  (“Secretary” variant of partition max k-cover)

Deterministic algorithm?

Our algorithm is randomized since f_p can only be computed by sampling
Continuous greedy algorithm is randomized for similar reasons

Best known deterministic algorithm:
Residual greedy [Buchbinder, Feldman, Garg '19], 0.5008 approximation

Deterministic algorithm?

Our algorithm is randomized since f_p can only be computed by sampling
Continuous greedy algorithm is randomized for similar reasons

Best known deterministic algorithm:
Residual greedy [Buchbinder, Feldman, Garg '19], 0.5008 approximation

Similar story for unconstrained submodular maximization**:**

Feige, Mirrokni, Vondrák '11: deterministic 1/3 approximation
Buchbinder, Feldman, Naor, Schwartz '15: randomized 1/2 approximation
Buchbinder, Feldman, Naor, Schwartz '15: deterministic 1/3 approximation

Deterministic algorithm?

Our algorithm is randomized since f_p can only be computed by sampling
Continuous greedy algorithm is randomized for similar reasons

Best known deterministic algorithm:
Residual greedy [Buchbinder, Feldman, Garg ’19], 0.5008 approximation

Similar story for unconstrained submodular maximization**:**

Feige, Mirrokni, Vondrák ’11: deterministic 1/3 approximation
Buchbinder, Feldman, Naor, Schwartz ’15: randomized 1/2 approximation
Buchbinder, Feldman, Naor, Schwartz ’15: deterministic 1/3 approximation

Buchbinder, Feldman ’16: deterministic 1/2 approximation

Happy birthday, Uri!

This is also not Uri!