Advanced Graph Algorithms

Approximation Algorithms

Martin Milanič
martin.milanic@upr.si
University of Primorska, Koper, Slovenia

University of Buenos Aires, November 14, 2016

Coping with NP-Hardness

Suppose you need to solve an NP-hard problem X.
Theory says that most likely there is no polynomial-time algorithm for X.

Are you going to give up?

- Probably yes, if the goal is really to find a polynomial-time algorithm.
- Probably not, if your job depends on a solution to the problem.

Coping with NP-Hardness

A naive approach:

- develop smart strategies of searching through the space of all possible solutions;
- an optimal solution is always found;
- no guarantee on running time.

Heuristics:

- intuitive algorithms;
- guaranteed to run in polynomial time;
- no guarantee on quality of solution.

Coping with NP -Hardness

Approximation algorithms:

- guaranteed to run in polynomial time;
- guaranteed to find "high quality" solution, say within 1% / 10% / 50% / a factor of 3 of optimum;
- here we face a **difficulty:** need to prove a solution's value is close to optimum, without even knowing the optimum value!

Approximation Algorithms

- The development of approximation algorithms can thus be seen as one of possible answers to the impossibility of efficiently solving a number of important NP-hard optimization problems.
  We are therefore satisfied with **sufficiently good** feasible solutions, which can be computed **fast enough**.

- The goal is of course to sacrifice as little as possible on optimality, while retaining as good time (and space) complexity of the algorithm as possible.

- The theory of approximation algorithms seeks which relations between quality of solution and running time can be obtained for a given problem.

Approximation Algorithms

We will give an overview of approximation algorithms for selected graph problems.

Algorithms are typically problem-specific, but some general features will also be outlined (when applicable).

Approximation Algorithms

For further reading, there are many possibilities:

- **Vazirani**, Approximation Algorithms, Springer, 2001,
- **Ausiello et al.**, Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties, Springer, 2003,
- **Hochbaum (ed.)**, Approximation Algorithms for NP-Hard Problems, PWS 1997,
- **Williamson, Shmoys**, The Design of Approximation Algorithms, Cambridge University Press, 2010,
- and of course research papers.

Instances of optimization problems

An **instance** (of an optimization problem) is an ordered triple $(\mathcal{S}, f, opt)$, where:

- $\mathcal{S}$ is an (implicitly given) set of feasible solutions
- $f : \mathcal{S} \to \mathbb{R}$ is the objective function
- $opt \in \{\min, \max\}$ type of problem: *minimization* or *maximization*

We are looking for

$$OPT := opt\{f(x) \mid x \in \mathcal{S}\}\,.$$

Instances of optimization problems

Example:
Traveling Salesman:
Input: a distance matrix $D = [d_{ij}]_{i,j=1}^{n}$
$d_{ij} \geq 0$: length of path from i to j

$\mathcal{S} = \{\text{traveling salesman tours}\}$

tour $= \pi \in S_n$ (π is a cyclic permutation of the set $\{1, \ldots, n\}$, a permutation with a unique cycle)
(Equivalently: a Hamiltonian cycle in the complete graph.)

$\pi = (i_1\ i_2\ \cdots\ i_n)$
$f(\pi) = \sum_{k=1}^{n-1} d_{i_k i_{k+1}} + d_{i_n i_1}$
$opt = \min$

The set of all instances for the traveling salesman
= *the traveling salesman problem* ▲

Instances of optimization problems

Example:

Vertex cover:

Input: a graph $G = (V, E)$

$\mathcal{S} = \{C \subseteq V : C \text{ is a vertex cover of } G\}$

vertex cover: a set C of vertices such that every edge $e \in E$ has at least one vertex in C

$f(C) = |C|$

$opt = \min$ ▲

Approximation Algorithms

Let Π be an optimization problem such that for every instance of the problem and every feasible solution $x \in \mathcal{S}$, the objective function value takes **positive value** ($f(x) > 0$).

ρ**-approximation algorithm**:

- An algorithm *A* for an optimization problem Π that runs in polynomial time.
- For every instance of Π, *A* outputs a feasible solution with objective function value within ratio ρ of true optimum for that instance.

ρ = approximation ratio / approximation factor

More specifically:

- for minimization problems:
  for every instance I, we have $f_A(I) \leq \rho \cdot \mathrm{OPT}(I)$, where
  $f_A(I)$ is the value of the solution returned by the algorithm,
  and
  $\mathrm{OPT}(I)$ is the optimal solution value.
- for maximization problems: $f_A(I) \geq \mathrm{OPT}(I)/\rho$.

Approximation Algorithms and Schemes

ρ-approximation algorithm:

- An algorithm A for an optimization problem Π that runs in polynomial time.
- For every instance of Π, A outputs a feasible solution with objective function value within ratio ρ of true optimum for that instance.

Polynomial-time approximation scheme (PTAS):

- A family of approximation algorithms $\{A_\epsilon : \epsilon > 0\}$ for a problem Π.
- A_ϵ is a $(1+\epsilon)$-approximation algorithm for Π.
- For every $\epsilon > 0$, A_ϵ runs in time polynomial in the size of input instance.

Fully polynomial-time approximation scheme (FPTAS):

- PTAS such that A_ϵ runs in time polynomial in the size of input instance and $1/\epsilon$.

Approaches to the Design of Approximation Algorithms

There exist several approaches to the design of approximation algorithms:

- combinatorial algorithms,
- algorithms based on linear programming,
- randomized algorithms,
- algorithms based on geometric ideas,
- etc.

From an abstract viewpoint ideas for development of approximation algorithms are similar as with development of efficient algorithms for polynomially solvable problems:

Find an appropriate combinatorial structure of the problem and develop algorithmic techniques that will exploit this structure.

A remark on the running time of approximation algorithms.

A remark on the running time

We typically require for approximation algorithms that they run **in polynomial time**.

For particularly difficult problems we sometimes also allow **exponential running time**.

Example:

The **bandwidth** of a graph $G = (V, E)$ is defined as

$$\min_{f} \max_{uv=e\in E} |f(u) - f(v)|,$$

where the minimum is taken over all bijections
$f : V \to \{1, \ldots, n\}$.

- A graph G has bandwidth $\leq k$ if and only if there exists a linear ordering of its vertex set such that the resulting adjacency matrix of G has nonzero elements s only on diagonals “close” to the main diagonal.

Theorem

There exists a 2-approximation algorithm for the bandwidth problem running in time $O(2^n)$.

(Fürer, Gaspers, Kasiviswanathan, 2013)

There are $n!$ feasible solutions, which is significantly more than 2^n.

The result becomes interesting in view of the fact that

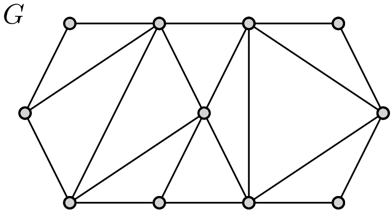
an arbitrary constant-factor polynomial time approximation of the bandwidth problem is NP-hard, even for trees.

(Dubey, Feige, Unger, 2011)

Approximation Algorithms for the Vertex Cover Problem

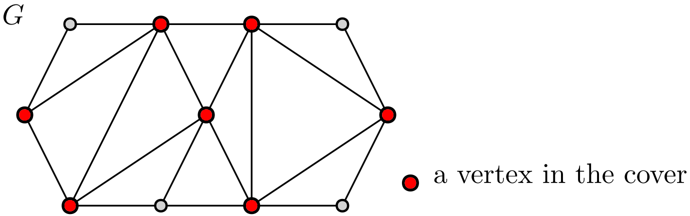
The Vertex Cover Problem

Recall:
vertex cover in a graph $G = (V, E)$:
a subset $C \subseteq V$ such that for all $e \in E$, $e \cap C \neq \emptyset$



The Vertex Cover Problem

Recall:
vertex cover in a graph $G = (V, E)$:
a subset $C \subseteq V$ such that for all $e \in E$, $e \cap C \neq \emptyset$



The Vertex Cover Problem

Consider the optimization version of the VERTEX COVER problem:

MINIMUM VERTEX COVER	
Input:	Graph $G = (V, E)$.
Task:	Find a minimum vertex cover in G.

In bipartite graphs, the problem can be solved optimally in polynomial time.
For general graphs, the problem is NP-hard.

Recall: a **matching** in a graph is a subset of pairwise disjoint edges.

2-Approximation Algorithm for Vertex Cover

Consider the following algorithm:

Approx-Cover:
$C := \emptyset$;
while $(\exists e = uv \in E)(u, v \in V \setminus C)$ **do**
    $C := C \cup \{u, v\}$
end while
return C.

The algorithm computes an inclusion-wise maximal matching M and returns the union of all edges in the matching.

2-Approximation Algorithm for Vertex Cover

Claim

Approx-Cover *is a* 2*-approximation algorithm for the* MINIMUM VERTEX COVER *problem.*

Proof:

The stopping criterion of the **while** loop guarantees that C is a cover.

Clearly, the algorithm can be implemented to run in polynomial time.

Let M be the maximal matching consisting of all edges chosen by the algorithm.

Every vertex cover must contain at least one vertex of each edge of M, hence $\mathrm{OPT} \geq |M|$

and consequently

$$|C| = 2|M| \leq 2 \cdot \mathrm{OPT}\,.$$

□

Can the factor of 2 in the analysis be improved?

No: it can happen that we get a 2-approximation and nothing better.

Example:
Let $G = K_{n,n}$.
The algorithm always returns the whole vertex set as a vertex cover, $C = V(K_{n,n})$. This is of size $2n$.
However, any optimal solution is of size n
(either part of the bipartition). ▲

Inapproximability issues

Is it possible to approximate the problem better?

- If there exists a polynomial 1.36-approximation algorithm for MINIMUM VERTEX COVER, then P = NP (Dinur-Safra 2005).
- No ρ-approximation algorithm for MINIMUM VERTEX COVER is known with $\rho < 2$.

Greedy approximation

Consider also the following simple algorithm:

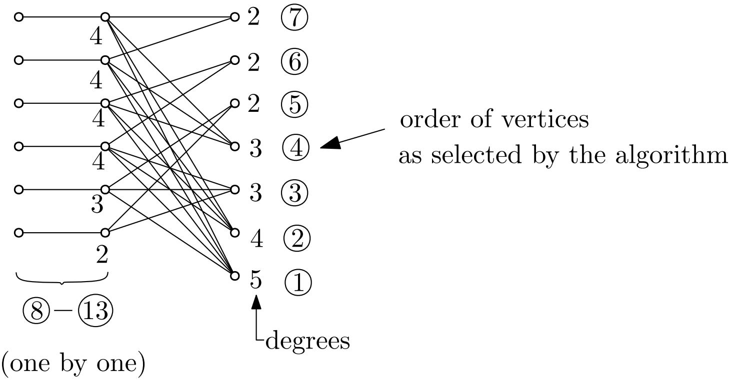
GREEDY APPROXIMATION
Input: Graph $G = (V, E)$.
Output: A cover C.
$H := G$
$C := \emptyset$
while $(E(H) \neq \emptyset)$
    Let u be a vertex of maximum degree in H.
    $C := C \cup \{u\}$.
    $H := H - u$.
end while
return C.

Greedy approximation

Set C returned by the algorithm is clearly a vertex cover.

The approximation ratio of this algorithm can be arbitrarily large.

Example:



The greedy approximation can take 13 vertices, the optimal value is 6.

Greedy approximation

The construction can be generalized, showing that the greedy approximation has no constant approximation ratio.
(See Korte-Vygen, Combinatorial Optimization, p. 396–397.)

For $n \geq 3$ and $i \leq n$ let $A_n^i := \sum_{j=2}^{i} \lfloor \frac{n}{j} \rfloor$.

$V(G_n) = \{a_1, a_2, \ldots, a_{A_n^{n-1}}, b_1, \ldots, b_n, c_1, \ldots, c_n\}$.

$E(G_n) = \{b_i c_i \mid i = 1, \ldots, n\} \cup$
$\bigcup_{i=2}^{n-1} \bigcup_{j=A_n^{i-1}+1}^{A_n^i} \left\{ a_j b_k \mid (j - A_n^{i-1} - 1)i + 1 \leq k \leq (j - A_n^{i-1})i + 1 \right\}$.

The algorithm will choose $A_n^{n-1} + n$ vertices, while $\{b_1, \ldots, b_n\}$ is a vertex cover of size n.

$$A_n^{n-1} \geq nH(n-1) - n - (n-2)\,,$$

where $H(n) = \sum_{i=1}^{n} \frac{1}{i}$ is the nth harmonic number.
(For every positive integer n, we have $H(n) > \ln n$. In the example above we had $n = 6$.)

Approximating the Set Cover problem

The SET COVER problem

SET COVER

Input: A ground set $U = \{u_1, \dots, u_n\}$,
a family $\mathcal{F} = \{S_1, \dots, S_m\}$ of subsets of U,
(we assume $S_1 \cup \dots \cup S_m = U$)
positive costs of subsets $c(S_1), \dots, c(S_m)$.

Task: Find a cheapest covering subfamily $\mathcal{F}' \subseteq \mathcal{F}$.

A subfamily $\mathcal{F}' = \{S_{i_1}, \dots, S_{i_k}\}$ is said to be **covering** (or: a **cover**) if $S_{i_1} \cup \dots \cup S_{i_k} = U$.

Example:

$U = \{1, 2, 3, 4, 5, 6\}$

$S_1 = \{1, 2, 3, 4\}$, $c(S_1) = 9$
$S_2 = \{1, 2, 5\}$, $c(S_2) = 5$
$S_3 = \{2, 3, 4\}$, $c(S_3) = 3$
$S_4 = \{2, 3, 6\}$, $c(S_3) = 4$
$S_5 = \{5, 6\}$, $c(S_3) = 2$

Cheapest cover: $\{S_2, S_3, S_5\}$

Cost of the cover: $c(S_2) + c(S_3) + c(S_5) = 10$.

Greedy method for SET COVER

C: set of already covered elements of U

$\overline{C} = U \setminus C$: set of not yet covered elements of U

effective cost of a set $S := c(S)/|S \cap \overline{C}|$

Greedy-Cover$(U, S_1, \ldots, S_m, c)$:
$C \leftarrow \emptyset$, $F \leftarrow \emptyset$
while $C \neq U$ **do**
    $S \leftarrow$ set with minimum effective cost
    $F \leftarrow F \cup \{S\}$, $C \leftarrow C \cup S$
end while
return F

C: set of already covered elements of U

$\overline{C} = U \setminus C$: set of not yet covered elements of U

effective cost of a set $S := c(S)/|S \cap \overline{C}|$

For the purpose of the analysis, we introduce a cost for each newly covered element:

Greedy-Cover$(U, S_1, \ldots, S_m, c)$:
$C \leftarrow \emptyset$, $F \leftarrow \emptyset$
while $C \neq U$ **do**
    $S \leftarrow$ set with minimum effective cost
    $\alpha \leftarrow c(S)/|S \cap \overline{C}|$
    for each $u \in S \cap \overline{C}$ **do** $cost(u) = \alpha$
    $F \leftarrow F \cup \{S\}$, $C \leftarrow C \cup S$
end while
return F

The analysis

We may assume that the algorithm covers elements $u_1, \ldots, u_n$ in this order.

Claim

For all $k = 1, \ldots, n$ we have:

$$cost(u_k) \leq \frac{\mathrm{OPT}}{n - k + 1}\,.$$

Proof:

Let $\overline{C}$ be the set of uncovered elements just before element u_k gets covered.

Elements in $\overline{C}$ can be covered with at most $|\overline{C}|$ sets of total cost $\leq \mathrm{OPT}$. Hence, there exists a set S with effective cost $\leq \frac{\mathrm{OPT}}{\overline{C}}$.

It follows:

$$cost(u_k) \leq \frac{\mathrm{OPT}}{\overline{C}} \leq \frac{\mathrm{OPT}}{n - k + 1}\,.$$

□

The analysis

Proposition

Greedy-Cover *is an $H(n)$-approximation algorithm for* SET COVER, *where*

$$H(n) = 1 + 1/2 + 1/3 + \ldots + 1/n \leq \ln n + 1 \,.$$

Proof:

$$c(F) = \sum_{k=1}^{n} cost(u_k) \leq \sum_{k=1}^{n} \frac{\mathrm{OPT}}{n-k+1} = \mathrm{OPT} \cdot \left(\sum_{k=1}^{n} \frac{1}{k} \right) \,.$$

Most likely, this is best possible:

- For any $\epsilon > 0$, if there exists an approximation algorithm for SET COVER with approximation ratio $(1-\epsilon)\ln n$, then P = NP (Dinur-Steurer 2014).

Applications to Graph Problems: Variants of Domination

The dominating set problem

a **dominating set** in a graph $G = (V, E)$:
a set $S \subseteq V$ such that every vertex is either in S or has a neighbor in S

DOMINATING SET	
Input:	A graph $G = (V, E)$
Task:	Compute a dominating set of minimum size.

DOMINATING SET is a well known NP-hard problem.

How well can it be approximated?

We can model DOMINATING SET as a special case of SET COVER.

Let us say that a vertex v is **dominated** by a set S if either v is in S or v has a neighbor in S.

Then, placing a vertex x in S dominates all elements of its closed neighborhood, defined as

$$N[x] = \{x\} \cup N(x) .$$

So we can take:

- the ground set $U = V$,
- the set family $\mathcal{F} = \{S_v : v \in V\}$ where $S_v = N[v]$,
- the cost function $c(S_v) = 1$ for all $v \in V$.

Indeed, we then have:

A set $D \subseteq V$ is a dominating set in G if and only if the set $\{S_v : v \in D\}$ is a covering subfamily of $\mathcal{F}$. And conversely, every covering subfamily arises this way.

Corollary

The **Dominating Set** *problem can be approximated to within a factor of* $\ln n + 1$ *on n-vertex graphs.*

Two remarks:

1. This is essentially **best possible.**
   - The inapproximability result of Dinur and Steurer for the SET COVER problem implies a similar result for **Dominating Set**.

2. The same approach can be used to model many other variants of domination, for example:
   - **total domination**: every vertex has a neighbor in the set
   - **distance-k domination**: every vertex is at distance at most k from a vertex in the set
   - **vertex cover**
     (here, as we know, one can do better: there is a 2-approx.)

Another Variant of Domination: Vector Domination

Vector domination in graphs

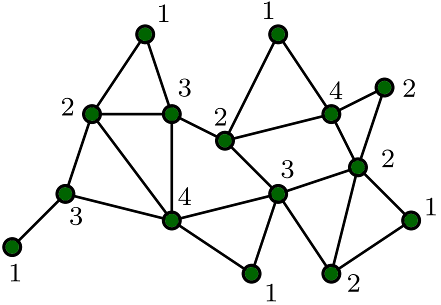
Given: a graph $G = (V, E)$
For every vertex v, an integer $r(v)$

A set $S \subseteq V$ is a vector dominating set for (G, r) if every vertex in $V \setminus S$ has at least $r(v)$ neighbors in S.

Vector domination in graphs

Given: a graph $G = (V, E)$
For every vertex v, an integer $r(v)$

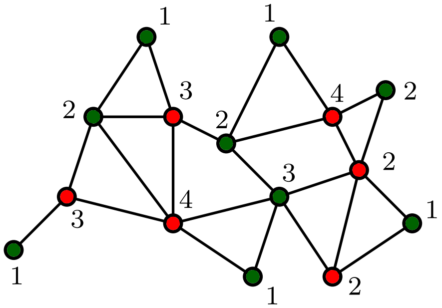
A set $S \subseteq V$ is a vector dominating set for (G, r) if every vertex in $V \setminus S$ has at least $r(v)$ neighbors in S.



Vector domination in graphs

Given: a graph $G = (V, E)$
For every vertex v, an integer $r(v)$

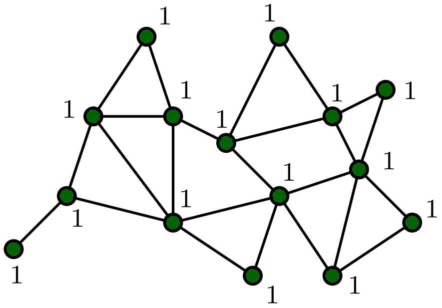
A set $S \subseteq V$ is a vector dominating set for (G, r) if every vertex in $V \setminus S$ has at least $r(v)$ neighbors in S.



Vector domination in graphs

Vector domination generalizes:

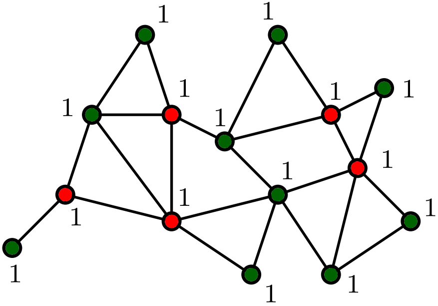
- **domination:** $r(v) = 1$ for all v



Vector domination in graphs

Vector domination generalizes:

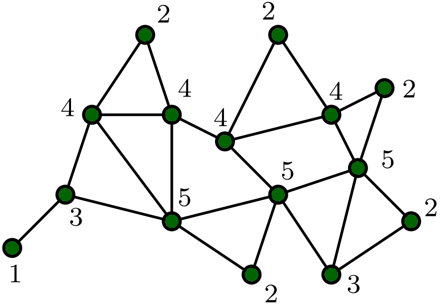
- **domination:** $r(v) = 1$ for all v



Vector domination in graphs

Vector domination generalizes:

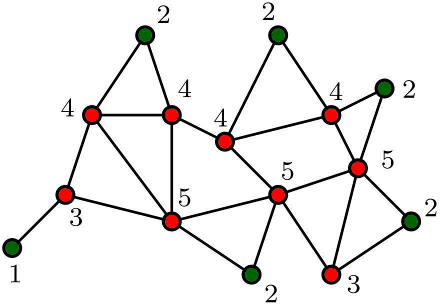
- domination: $r(v) = 1$ for all v
- **vertex cover:** $r(v) = d(v)$ for all v



Vector domination in graphs

Vector domination generalizes:

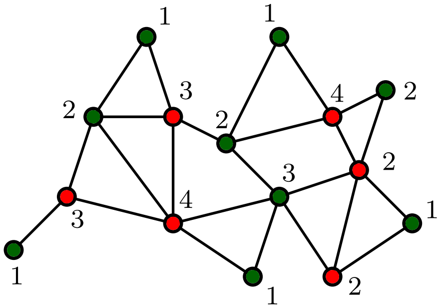
- domination: $r(v) = 1$ for all v
- **vertex cover:** $r(v) = d(v)$ for all v



Vector domination in graphs

Given: a graph $G = (V, E)$
For every vertex v, an integer $r(v)$

A set $S \subseteq V$ is a vector dominating set for (G, r) if every vertex in $V \setminus S$ has at least $r(v)$ neighbors in S.



The vector domination problem

VECTOR DOMINATION

Input: A graph $G = (V, E)$, an function $r(v) : V \to \mathbb{Z}^+$
Task: Compute a minimum vector dominating set for (G, r).

There is an extension of SET COVER called SET MULTICOVER, where each element needs to be covered multiple times. This problem can also be approximated greedily, with a ratio of $H(\Delta)$ where Δ is the maximum size of a set in the family (Dobson 1982).

Bad news:
It is not clear how to model VECTOR DOMINATION in this setting.

Good news:
We can use a different result from 1982 to solve this problem!

The vector domination problem

We will obtain the following:

Theorem

The VECTOR DOMINATION *problem can be approximated in polynomial time to within a factor of* $\ln(2\Delta(G)) + 1$*, where* $\Delta(G)$ *is the maximum degree of* G*.*

First, note that we may assume that for all $v \in V$, we have $r(v) \leq d(v)$, where $d(v)$ is the degree (the number of neighbors) of v in G:

- If $r(v) > d(v)$, then v must be contained in every vector dominating set.
  Thus, we can set $r(w) \leftarrow r(w) - 1$ for all $w \in N(v)$ and add v to an optimal (or approximate) solution for the reduced problem on $G - v$.

Approximating vector domination

Greedy Strategy

- start with $S = \emptyset$
- if S is not a vector dominating set, keep on adding to S a vertex $v \in V \setminus S$ maximizing $f(S \cup \{v\}) - f(S)$

$$\text{argmax}_{v \in V} \left(f(S \cup \{v\}) - f(S) \right)$$

What is f?

$$f(X) = \sum_{v \in V} f_v(X)\,, \text{ for all } X \subseteq V, \text{ and}$$

$$f_v(X) = \begin{cases} \min\{|X \cap N(v)|, r(v)\} & \text{if } v \notin X; \\ r(v) & \text{if } v \in X. \end{cases}$$

$|X \cap N(v)|$ = the number of already chosen neighbors of v

Approximating vector domination

$$f(X) = \sum_{v \in V} f_v(X)\,,\ \text{for all } X \subseteq V, \text{ and}$$

$$f_v(X) = \begin{cases} \min\{|X \cap N(v)|, r(v)\} & \text{if } v \notin X; \\ r(v) & \text{if } v \in X. \end{cases}$$

Note that:

- $f(V) = \sum_{v \in V} r(v)$
- $f(X) = f(V)$ if and only if $X \subseteq V$ is a vector dominating set for (G, r).
- Hence, the VECTOR DOMINATION problem asks for a smallest set $X \subseteq V(G)$ with $f(X) = f(V)$.

Approximating vector domination

It can be shown that f is a (non-decreasing, integer-valued) **submodular set function.**

Submodularity is a discrete analog of concavity:

$$X \subseteq Y \quad \Rightarrow \quad f(X \cup \{v\}) - f(X) \geq f(Y \cup \{v\}) - f(Y)\,.$$

f is *non-decreasing* if $X \subseteq Y \Rightarrow f(X) \leq f(Y)$

Hence, the vector domination problem is a special case of the **Minimum Submodular Cover** problem:

Input: A finite set V and an integer-valued non-decreasing submodular set function f on subsets of V (given by an oracle).

Task: Find a smallest set $X \subseteq V$ such that $f(X) = f(V)$.

Approximating vector domination

By a result of [Wolsey, 1982] on minimum submodular cover, the greedy strategy approximates OPT by a factor of at most $H(\max f(\{y\}))$.

For every $y \in V$, we have

$$f(\{y\}) = \sum_{v \in V \setminus \{y\}} f_v(\{y\}) + f_y(\{y\}) \leq d(y) + r(y) \leq 2d(y)\,.$$

Hence $\max_{y \in V} f(\{y\}) \leq 2\Delta(G)$ and the greedy strategy approximates OPT by a factor of at most

$$H(2\Delta(G)) \leq \ln(2\Delta(G)) + 1\,,$$

as claimed. □