Network monitoring: detecting node failures

Monitoring failures in (communication) DS

- A major activity in DS consists of monitoring whether all the system components work properly
- To our scopes, we will concentrate our attention on DS which can be modelled by means of a MPS, thus embracing all those real-life applications which make use of an underlying communication graph $G=(V,E)$
- Here, we have to monitor nodes and links (mal)functioning, through the use of a set of sentinel nodes, which will periodically return to a network administrator a certain set of information about their neighborhood

Formalizing the k-node-monitoring problem

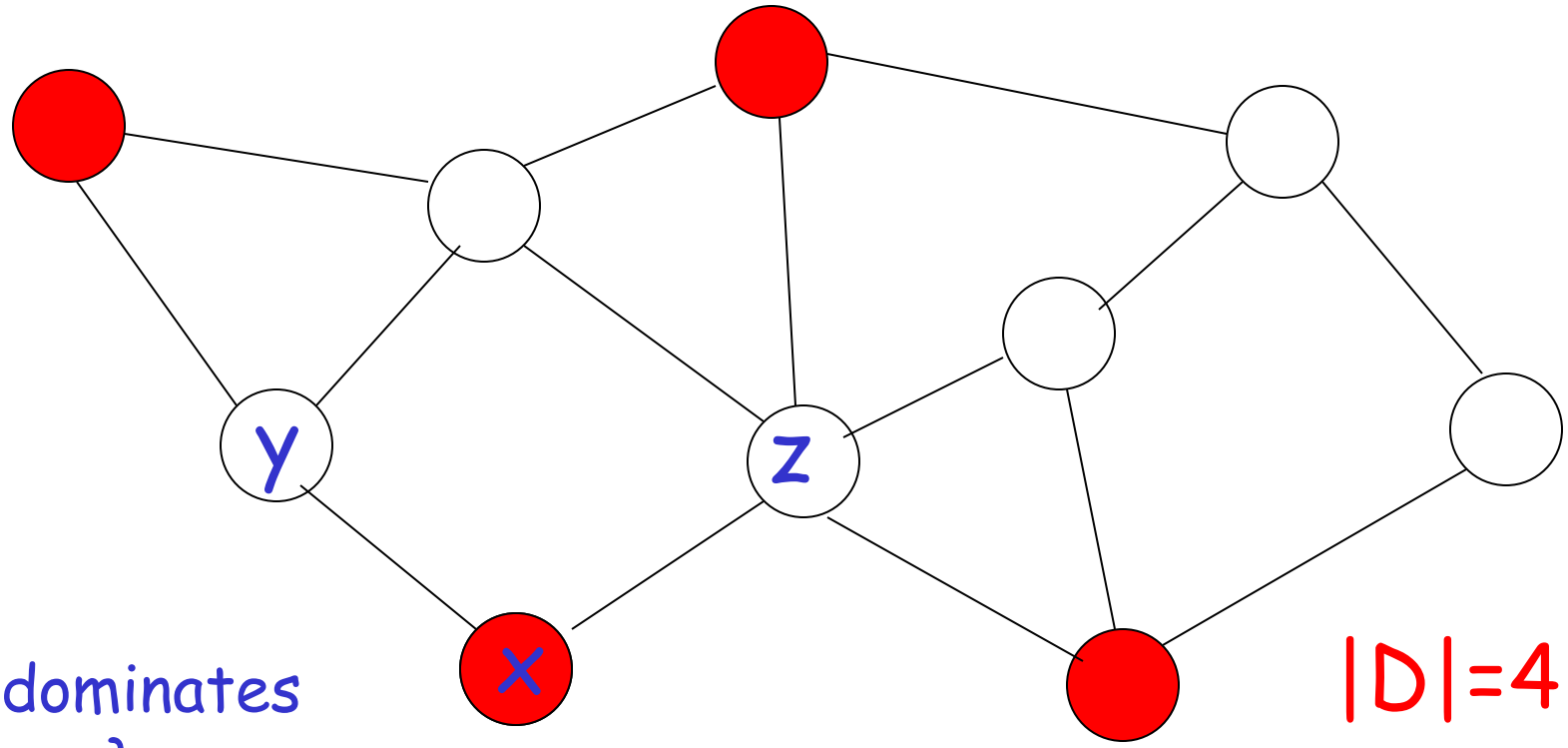
- **Input**: A graph $G = (V,E)$ modeling a MPS, a parameter k, and a query model Q, namely a formal description of the entire process through which a sentinel node x reports its piece of information to the network administrator (i.e., (1) which nodes are queried by x, and (2) which type of information x can return to the system as a result of the queries);
- **Goal**: Compute a minimum-size subset of sentinel nodes $S \subseteq V$ allowing to monitor G with respect to the *simultaneous* failure of at most k nodes in G, i.e., such that the composition of the information reported by the nodes in S to the network administrator is sufficient to identify the precise set of crashed nodes, for any such set of size at most k.

Network monitoring and dominance in graphs

- The simplest possible query models are those in which each sentinel node communicates with its neighbors only, and thus a sentinel node can report a set of information about its neighborhood $\Rightarrow$ the monitoring problem in this case is naturally related with the concept of dominance in graphs, i.e., with the activity of selecting a set of nodes (dominators) in a graph in order to have all the nodes of the graph within distance at most 1 from at least a dominator
- These query models are then further refined on the basis of the type of messages that sentinel nodes exchange with their neighbors and with the network administrator

Dominating Set

Given a graph $G=(V,E)$, a dominating set of G is a set of nodes $D \subseteq V$ such that every node of G is at distance at most 1 from D

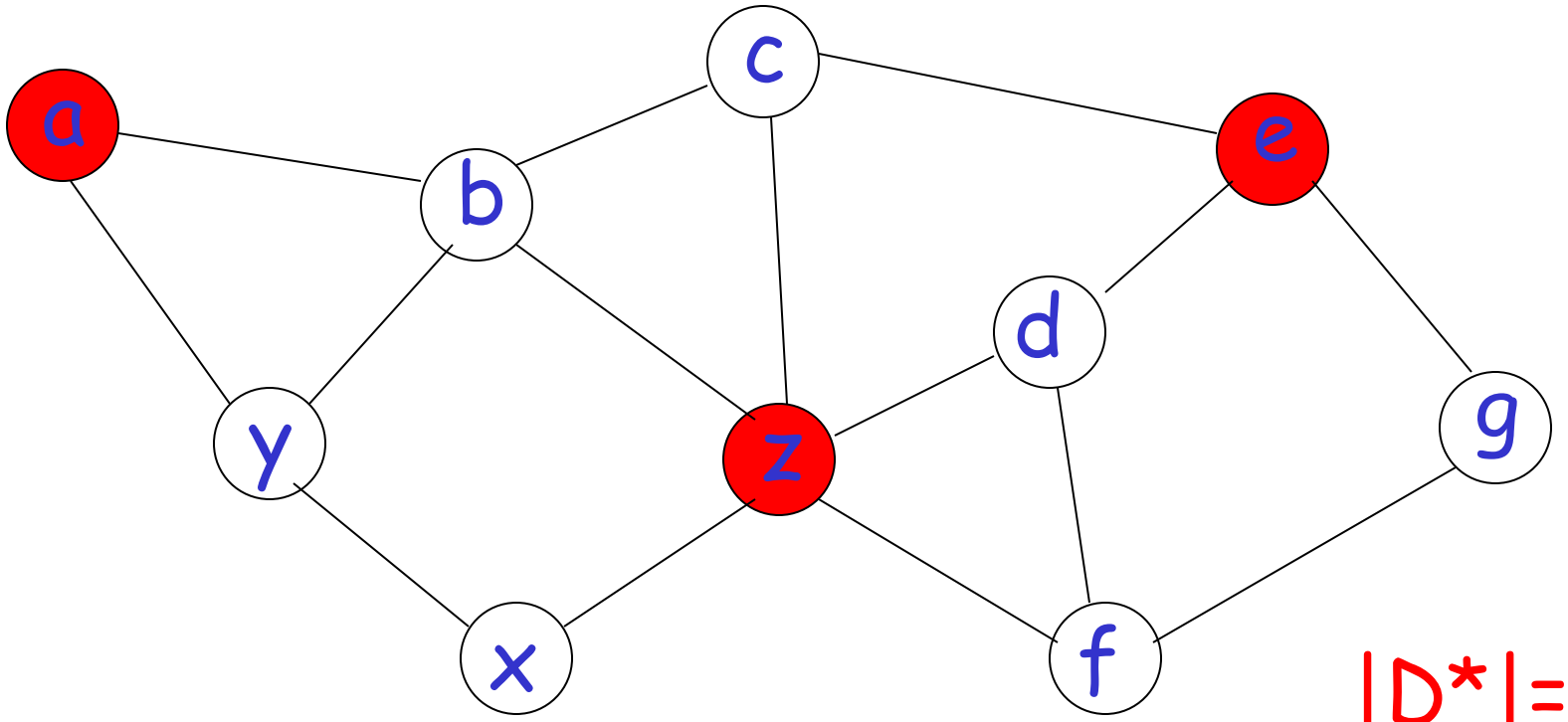


x dominates {x,y,z}

|D|=4

Minimum Dominating Set (MDS):

This is a dominating set of minimum size



$|D^*|=3$

Network monitoring and MDS

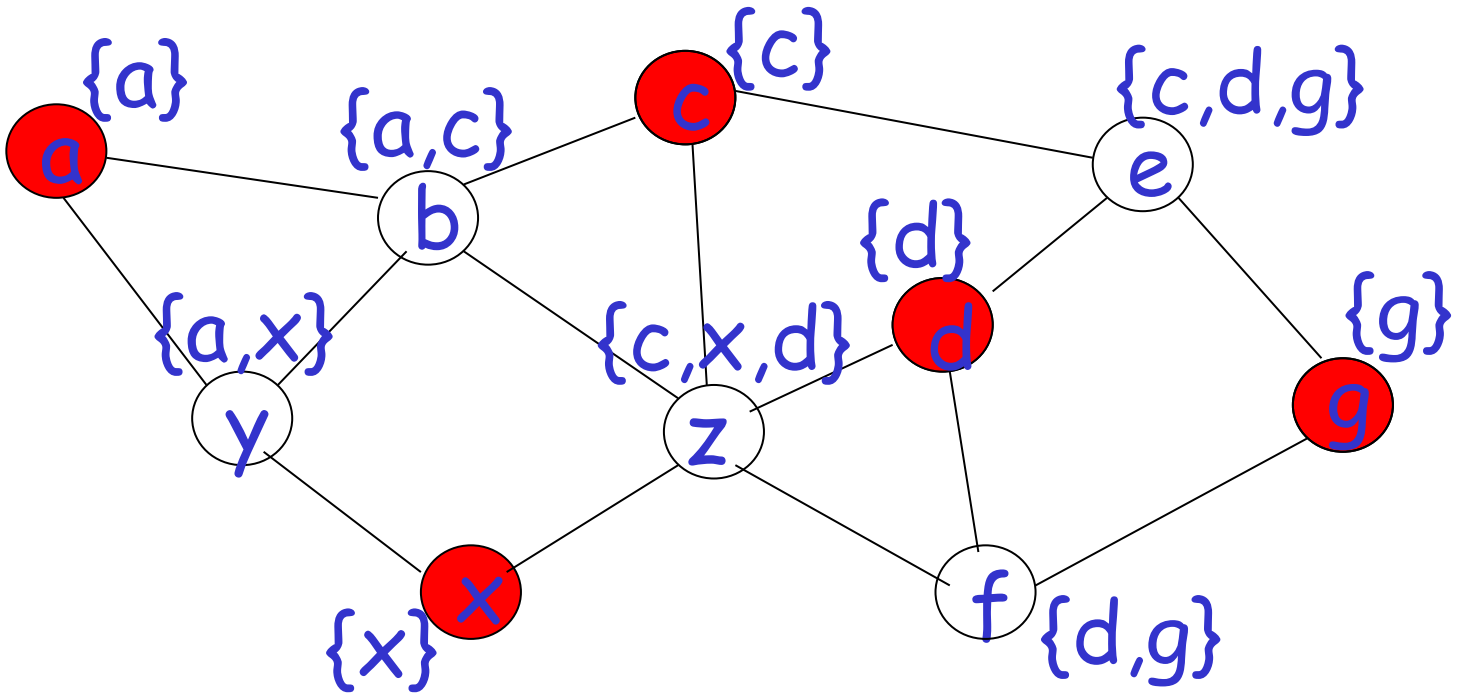
In a query model in which a sentinel node:

1. Sends a ping to each adjacent node and waits for a reply;
2. Reports to the network administrator the id (i.e., the system is **non-anonymous**) of the set of adjacent nodes which did not reply;

a MDS D^* of a graph $G=(V,E)$ is an optimal solution for the 1-node-monitoring problem (indeed, if a node in $V \setminus D^*$ fails, its id will be detected by at least a node in D^*, while if a node in D^* fails, then the system directly detects it since of the unreported monitoring).

A special type of Dominating Set: the Identifying Code (IC)

This is a dominating set D in which every node v is dominated by a distinct set of nodes in D (this is called the identifying set of v)



A Minimum IC (MIC) is an IC of smallest cardinality.

Network monitoring and MIC

In a query model in which a sentinel node:

1. Sends a ping to each adjacent node and waits for a reply;
2. Reports to the network administrator an alarm bit (0 if all the adjacent replied, 1 otherwise - notice that in this case the system might be **anonymous**);

a MIC C^* of a graph $G=(V,E)$ is an optimal solution for the 1-node-monitoring problem (indeed, if a node in $V \setminus C^*$ fails, it will be exactly detected by the unique set of node in C^* that dominate it, while if a node in C^* fails, then the system directly detects it since of the unreported monitoring).

Our problems

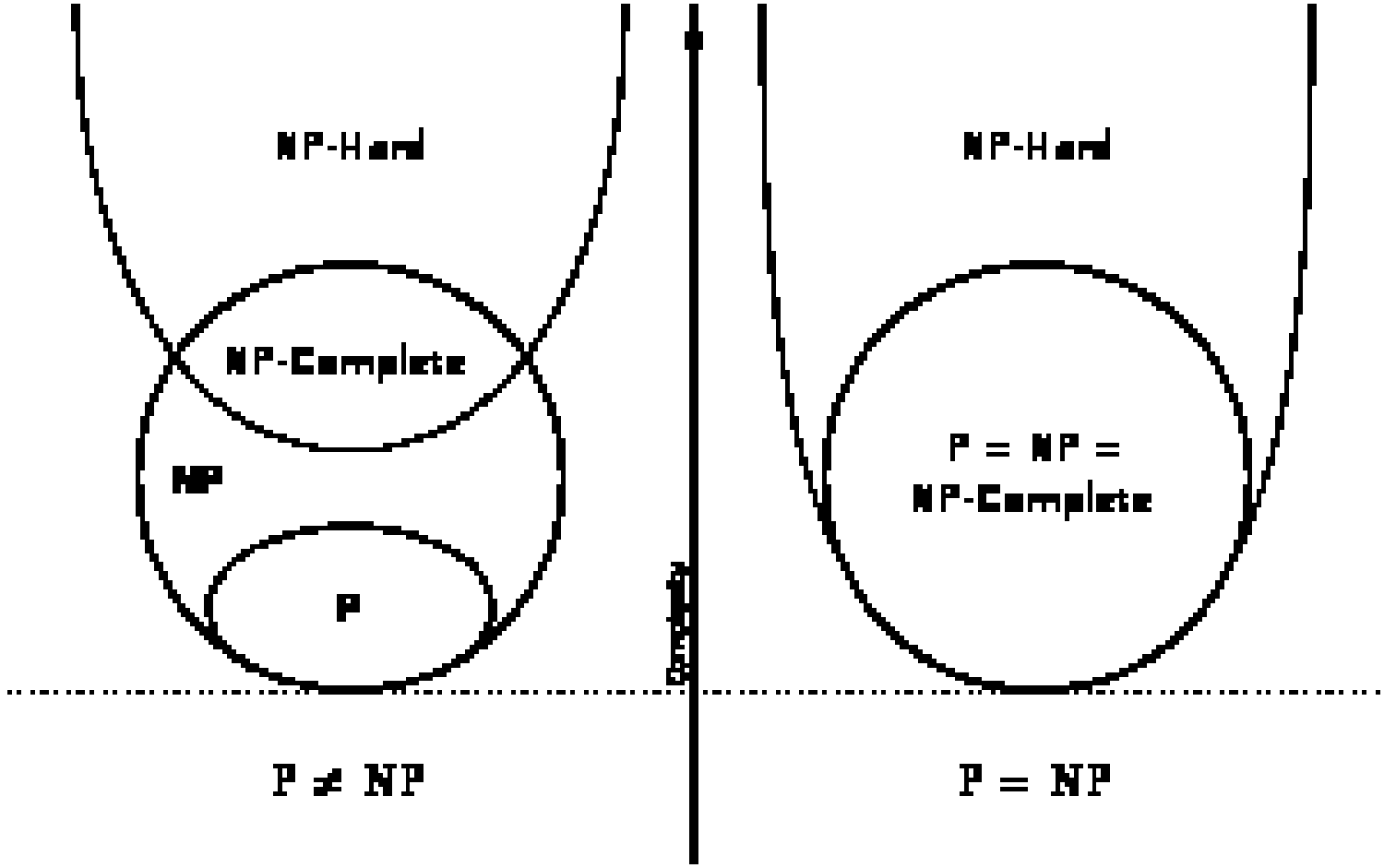
- We will then study the monitoring problem for **single** node failures (i.e., node crashes) w.r.t. the two following query models:
  1. Sentinels are able to return the id of the adjacent failed node $\Rightarrow$ we will search for a MDS of the network (MDS problem)
  2. Sentinels are only able to return an alarm bit about the neighborhood (i.e., a warning that an adjacent node has failed) $\Rightarrow$ we will search for a MIC of the network (MIC problem)
- **Main questions**: Are MDS and MIC problems easy or NP-hard? If so, can we provide efficient (distributed) approximation algorithms to solve them?
- We will show that MDS and MIC problems are NP-hard, and that they are both not approximable within $o(\ln n)$; we will also provide an $\Theta(\ln n)$-approximation distributed algorithm for MDS and MIC (only a sketch)

Reminder: being NP-hard

- A decision problem Π is NP-hard iff one can reduce in polynomial time any NP-complete problem Π' to it, i.e., there exists a polynomial-time algorithm that maps an instance of Π' to an instance of Π, and such that the YES-instances of Π' will be mapped to the YES-instances of Π, and vice-versa (this is a.k.a. Karp reduction, denoted by $\Pi' \leq_K \Pi$)
- Of course, if we could solve an NP-hard problem in polynomial time, then P=NP
- Notice that MDS and MIC are optimization problems, since we search for solutions of minimum size, and so they are not encompassed by the above NP-hardness definition
- However, it is easy to provide a decision version of an optimization problem, without affecting its intrinsic complexity: it suffices to add a threshold value to the input, and then asking whether a solution either above or below that threshold does actually exist

  $\Rightarrow$ In the following, we will assume that the class NP-hard contains both decision and optimization problems

The class NP-hard: a picture



Reminder: optimization problems and approximability

- An optimization problem A is a quadruple (I, F, c, g), where
  - I is a set of instances;
  - given an instance $x \in I$, $F(x)$ is the set of feasible solutions;
  - given a feasible solution $y \in F(x)$, $c(y)$ denotes the cost of y, which is usually a positive real;
  - g is the goal function, and is either min or max.
  - The goal is then to find for some instance x an optimal solution, that is, a feasible solution y with

$$c(y) = g\{c(y') \mid y' \in F(x)\}.$$

- For NP-hard optimization problems, unfortunately we do not know polynomial-time solving algorithms, thus we resort to approximation algorithms: Given a minimization (resp., maximization) problem A, let $OPT_A(x)$ denote the cost of an optimal solution for A w.r.t. the instance x; then, we say that A is ρ-approximable, with $\rho \geq 1$ (resp., $\rho \leq 1$), if there exists a polynomial-time algorithm for A which for **any instance** $x \in I$ returns a feasible solution whose measure is at most (resp., at least) $\rho \cdot OPT_A(x)$.
- Moreover, we say that A is ρ-inapproximable, if under some reasonable assumptions (typically, $P \neq NP$), A is not ρ-approximable

Reducibility between (NP-hard) optimization problems

- In a Karp reduction, the focus is just on a mapping between YES-instances of the two involved problems, so no any attention is paid w.r.t. the measure of the mapped solutions, which is instead crucial for optimization problems
- In other words, a Karp reduction transforms an NP-hard optimization problem into another NP-hard problem, but the two problems may be completely different from an approximability point of view
- For optimization problems, we can then use a special (stronger) type of reduction, namely the L-reduction, which linearly preserves the degree of approximability of a problem; such a type of reduction is also useful to prove NP-hardness, as we will see soon

L-reduction: definition

Let A and B be optimization problems and c_A and c_B be their respective cost functions. A pair of functions f and g is an L-reduction from A to B (we write $A \leq_L B$) if all of the following conditions are met:

1. functions f and g are computable in polynomial time;
2. if x is an instance of problem A, then $f(x)$ is an instance of problem B (and so f is used to transform **instances** of A into instances of B);
3. if y is a feasible solution of $f(x)$, then $g(x,y)$ is a feasible solution of x (and so g is used to transform **solutions** of B into solutions of A);
4. there exists a positive constant α such that for every instance x of A
$$OPT_B(f(x)) \leq \alpha\, OPT_A(x);$$
(informally, the cost of an optimal solution of the transformed instance is close to the cost of an optimal solution of the original instance)
5. there exists a positive constant β such that for every instance x of A and every solution y of $f(x)$
$$|OPT_A(x) - c_A(g(x,y))| \leq \beta |OPT_B(f(x)) - c_B(y)|$$
(informally, the distance from an optimal solution of the transformed solution $g(x,y)$ is close to the distance from an optimal solution of the solution found for the transformed instance $f(x)$).

L-reduction: consequences

Let us focus on minimization problems (all results hold for maximization problems as well):

Fact 1: If A is L-reducible to B and B admits a $(1+\delta)$-approximation algorithm, with $\delta>0$, then A admits a $(1+\delta\alpha\beta)$-approximation algorithm, where α and β are the constants associated with the L-reduction (and so in particular if $\alpha=\beta=1$, the approximation ratio is equal). Indeed, by dividing both sides of the inequality in Point 5 by $OPT_A(x)$:

$$|1 - c_A(g(x,y))/OPT_A(x)| \leq$$

$$\beta|OPT_B(f(x))/OPT_A(x) - (1+\delta)\ OPT_B(f(x))/OPT_A(x)|$$

and since from Point 4 $OPT_B(f(x)) \leq \alpha\ OPT_A(x)$

$$|1 - c_A(g(x,y))/OPT_A(x)| \leq \alpha\ \beta|1 - (1+\delta)\ | \Rightarrow c_A(g(x,y))/OPT_A(x) \leq 1+\delta\alpha\beta.$$

Fact 2: A corollary of Fact 1 is that if A is L-reducible to B and A is not approximable within a factor of $1+\delta$, with $\delta>0$, then B is not approximable within a factor of $1+\delta/\alpha\beta$ (since otherwise by Fact 1 A would admit a $(1+\delta)$-approximation algorithm). In particular, if $\delta=\omega(1)$, then a δ-inapproximability of A implies a δ-inapproximability of B

L-reduction: consequences (2)

Fact 3: If A is L-reducible to B and A is NP-hard, then B is also NP-hard (this comes from the first three items of the definition, since any reduction from an NP-complete problem Π to A can be easily extended in polynomial time to a reduction to B).

Fact 4: If A is L-reducible to B and B is L-reducible to A, then A and B are asymptotically equivalent in terms of (in)approximability.

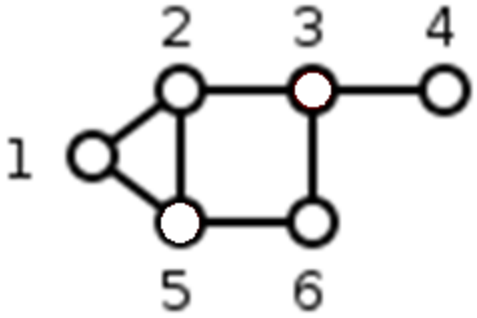
The Set Cover problem

We will show that the (Minimum) Set Cover Problem and the Minimum Dominating Set Problem are **asymptotically equivalent** in terms of (in)approximability (i.e., we show an L-reduction in both directions). Even more, we will show that in both directions of the reduction, $\alpha=\beta=1$.

- First of all, we recall the definition of the Set Cover (SC) problem. An instance of SC is a pair $(U=\{o_1,\ldots,o_m\}, S=\{S_1,\ldots,S_n\})$, where U is a universe of objects, and S is a collection of subsets of U. The objective is to find a minimum-size collection of subsets in S whose union is U. In the following, w.l.o.g., we assume that $n=\Theta(m)$
- SC is well-known to be NP-hard, and to be not approximable within $(1-o(1))\ln n$, unless P=NP.
- On the positive side, the greedy heuristic (i.e., at each step, and until exhaustion, choose the so-far unselected set in S that covers the largest number of uncovered elements in U) provides a (essentially tight) $\Theta(\ln n)$ approximation ratio.

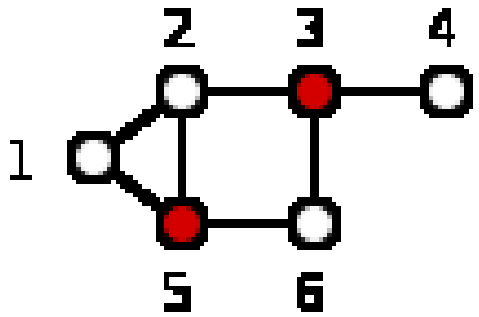
L-reduction from MDS to SC (1/3)

- **Instance tranformation:** Given an instance x of MDS, namely a graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$, construct (in polynomial time) an SC instance (U, S) as follows: the universe U is V, and the family of subsets is $S = \{S_1, S_2, \ldots, S_n\}$ such that S_i consists of the vertex i and all the vertices adjacent to i in G (this is the function f, and notice that the two instances are such that $|V|=|S|=n$).
- For example, given the graph $G = (V,E)$ shown below, we construct a set cover instance with the universe $U = \{1, 2, \ldots, 6\}$ and the subsets $S_1 = \{1, 2, 5\}$, $S_2 = \{1, 2, 3, 5\}$, $S_3 = \{2, 3, 4, 6\}$, $S_4 = \{3, 4\}$, $S_5 = \{1, 2, 5, 6\}$, and $S_6 = \{3, 5, 6\}$.



L-reduction from MDS to SC (2/3)

- **Solution tranformation:** It is easy to see that if $C = \{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$ is a feasible solution of SC, then $D = \{i_1, i_2, \ldots, i_k\}$ is a dominating set for G, (since all the objects in U, namely all the vertices of G, are covered), with $|D| = |C|$ (this is the poly-time computable function g, and notice that the two solutions have the same size, i.e., $c_A(g(x,y)) = c_B(y)$).
- Moreover, notice that if D is a dominating set for G, then $C = \{S_i : i \in D\}$ is a feasible solution of SC (since all the vertices of G, namely all the objects in U, are covered), with $|C| = |D|$. Hence, from the previous point, an optimal solution of MDS for G equals the size of a Minimum Set Cover (MSC) for (U,S)
- In our example, $D = \{3, 5\}$ is a dominating set for G, and this corresponds to the set cover $C = \{S_3, S_5\}$ of the universe. For example, vertex 4 is dominated by vertex $3 \in D$, and the element $4 \in U$ is contained in the set S



L-reduction from MDS to SC (3/3)

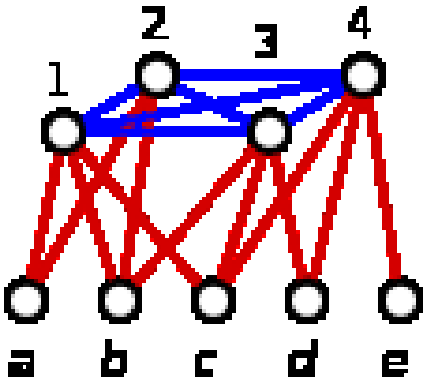
- From the previous two points, we have $OPT_B(f(x)) = OPT_A(x)$, and so $\alpha=1$, and moreover since as said before, $c_A(g(x,y)) = c_B(y)$, we have that

  $|OPT_A(x) - c_A(g(x,y))| = |OPT_B(f(x)) - c_B(y)|$, i.e., $\beta=1$.

- $\Rightarrow$ Therefore, MDS $\leq_L$ SC with $\alpha=\beta=1$, and thus from Fact 1, a ρ-approximation algorithm for SC provides exactly a ρ-approximation algorithm for MDS.

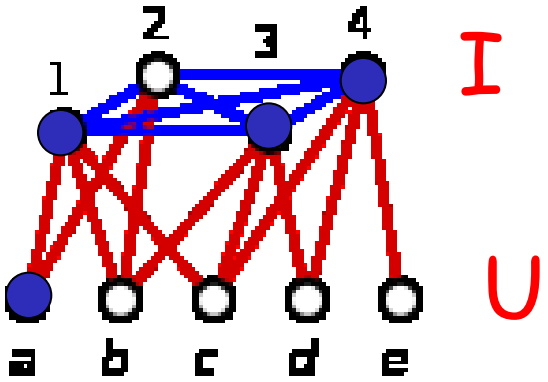
L-reduction from SC to MDS (1/4)

- **Instance tranformation:** Let now (U,S) be an instance of SC with the universe $U=\{o_1,\ldots,o_m\}$, and the family of subsets $S=\{S_1,\ldots,S_n\}$, and let $I=\{1,\ldots,n\}$; w.l.o.g., we assume that U and the index set I are disjoint. Construct (in polynomial time) a graph $G = (V,E)$ as follows: the set of vertices is $V = I \cup U$, there is an edge $\{i,j\} \in E$ between each pair $i, j \in I$, and there is also an edge $\{i,o\}$ for each $i \in I$ and $o \in S_i$. That is, G is a **split graph**: I is a clique and U is an independent set. This is the function f, and notice that the two instances are now such that $n=|S| \neq |V|=n+m$).
- For example, let be given the following instance of SC: $U = \{a, b, c, d, e\}$, $S_1 = \{a, b, c\}$, $S_2 = \{a, b\}$, $S_3 = \{b, c, d\}$, and $S_4 = \{c, d, e\}$, and so $I = \{1, 2, 3, 4\}$. This provides the graph $G=(I \cup U,E)$ below.



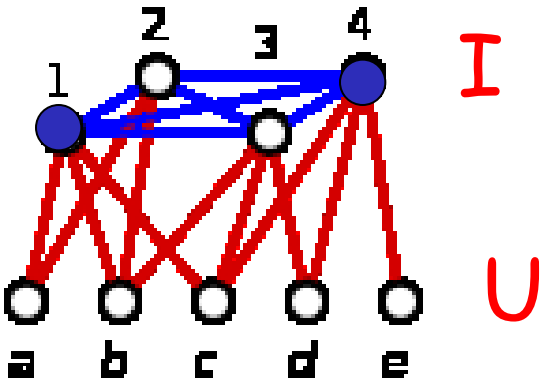
L-reduction from SC to MDS (2/4)

- **Solution tranformation**: Now let D be a dominating set for G. Then it is possible to construct another dominating set X such that $|X| \leq |D|$ and $X \subseteq I$: simply replace each $o \in D \cap U$ by a neighbour $i \in I$ of o. Then $C = \{S_i : i \in X\}$ is a feasible solution of SC, with $|C| = |X| \leq |D|$. This is the poly-time computable function g, but notice that the two solutions have not the same size in general (i.e., $c_A(g(x,y)) \leq c_B(y)$).
  Remark: Notice however that if D were a minimum dominating set for G, then clearly $|X| = |D|$ (indeed remember that X is a dominating set)
- For instance, in our example, where $U = \{a, b, c, d, e\}$, $S_1 = \{a, b, c\}$, $S_2 = \{a, b\}$, $S_3 = \{b, c, d\}$, and $S_4 = \{c, d, e\}$, given any dominating set for the graph G, say $D = \{a, 3, 4\}$, we can construct a dominating set $X = \{1, 3, 4\}$ which is not larger than D and which is a subset of I. The dominating set X now corresponds to the set cover $C = \{S_1, S_3, S_4\}$.



L-reduction from SC to MDS (3/4)

- Conversely, if $C = \{S_i : i \in D \subseteq I\}$ is a feasible solution of SC, then D is a dominating set for G, with $|D| = |C|$: First, for each $o \in U$ there is an $i \in D$ such that $o \in S_i$ (since D is a SC), and since by construction o and i are adjacent in G, this implies that all the nodes in U are dominated. Second, since $D \subseteq I$ must be nonempty, each $i \in I$ is adjacent to a vertex in D (since G[I] is a clique)
- For instance, in our example, $U = \{a, b, c, d, e\}$, $S_1 = \{a, b, c\}$, $S_2 = \{a, b\}$, $S_3 = \{b, c, d\}$, and $S_4 = \{c, d, e\}$, and so $C = \{S_1, S_4\}$ is a set cover; this corresponds to the dominating set $D = \{1, 4\}$.



L-reduction from SC to MDS (4/4)

- From the previous point and the remark on slide (2/4), it follows that

$$OPT_B(f(x)) = OPT_A(x), \text{ and so } \alpha=1.$$

- Moreover, as we said before, $c_A(g(x,y)) \leq c_B(y)$, and moreover $c_A(g(x,y)) \geq OPT_A(x)$ and $c_B(y) \geq OPT_B(f(x))$ (they are minimization problems), and so we have that

$$|OPT_A(x) - c_A(g(x,y))| \leq |OPT_B(f(x)) - c_B(y)|$$

and so $\beta=1$.

$\Rightarrow$ Then, SC $\leq_L$ MDS with $\alpha=\beta=1$, and from Fact 2, a $\omega(1)$-inapproximability of SC implies a $\omega(1)$-inapproximability for MDS.

Consequences on the approximability of MDS

- From the bidirectional L-reduction from SC, it follows that MDS is as hard to approximate as SC.
- This means, MDS is NP-hard and cannot be approximated within $(1-o(1)) \ln n$, unless P=NP.
- On the positive side, the greedy heuristic for SC (i.e., at each step, and until exhaustion, choose the so-far unselected set in S that covers the largest number of uncovered elements in U) can be reformulated for MDS in order to provide a (essentially tight) $\Theta(\ln n)$ approximation ratio.

Recap: the MDS and the MIC problems

- MDS: Given a graph $G=(V,E)$, find a dominating set of G (i.e., is a set of nodes $D \subseteq V$ such that every node of G is at distance at most 1 from D) of minimum size $\Rightarrow$ a MDS is useful to monitor node failures when sentinel nodes are able to report the ID of an adjacent failing node
- MIC: Given a graph $G=(V,E)$, find an identifying code of G (i.e., is a set of nodes $D \subseteq V$ such that every node v of G is at distance at most 1 from a univocal set of nodes $D_v \subseteq D$) of minimum size $\Rightarrow$ a MIC is useful to monitor node failures when sentinel nodes are able to only report an alarm bit that an adjacent node failed
- MDS is as hard to approximate as SC, i.e., it is NP-hard and cannot be approximated within $(1-o(1)) \ln n$, unless P=NP, but admits a (tight) $\Theta(\ln n)$ approximation algorithm

Centralized MDS Greedy Algorithm (1/4)

Greedy Algorithm (GA): For any node v of the given graph G, let N(v) denote its neighborhood in G, i.e., the set of adjacent nodes, and define span(v) to be the number of non-dominated nodes in {v} U N(v). Then, start with empty dominating set D, and so initially span(v)=1+|N(v)|, and at each step add to D node v with maximum span, until all nodes are dominated.

Theorem: The GA is $H(\Delta+1)$-approximating, where Δ is the degree of G, and $H(k) = 1+1/2+1/3+...+1/k \leq 1+\ln k$, i.e., the GA is $(1+\ln(\Delta+1))$-approximating, or $(1+\ln n)$-approximating (since $\Delta \leq n-1$).

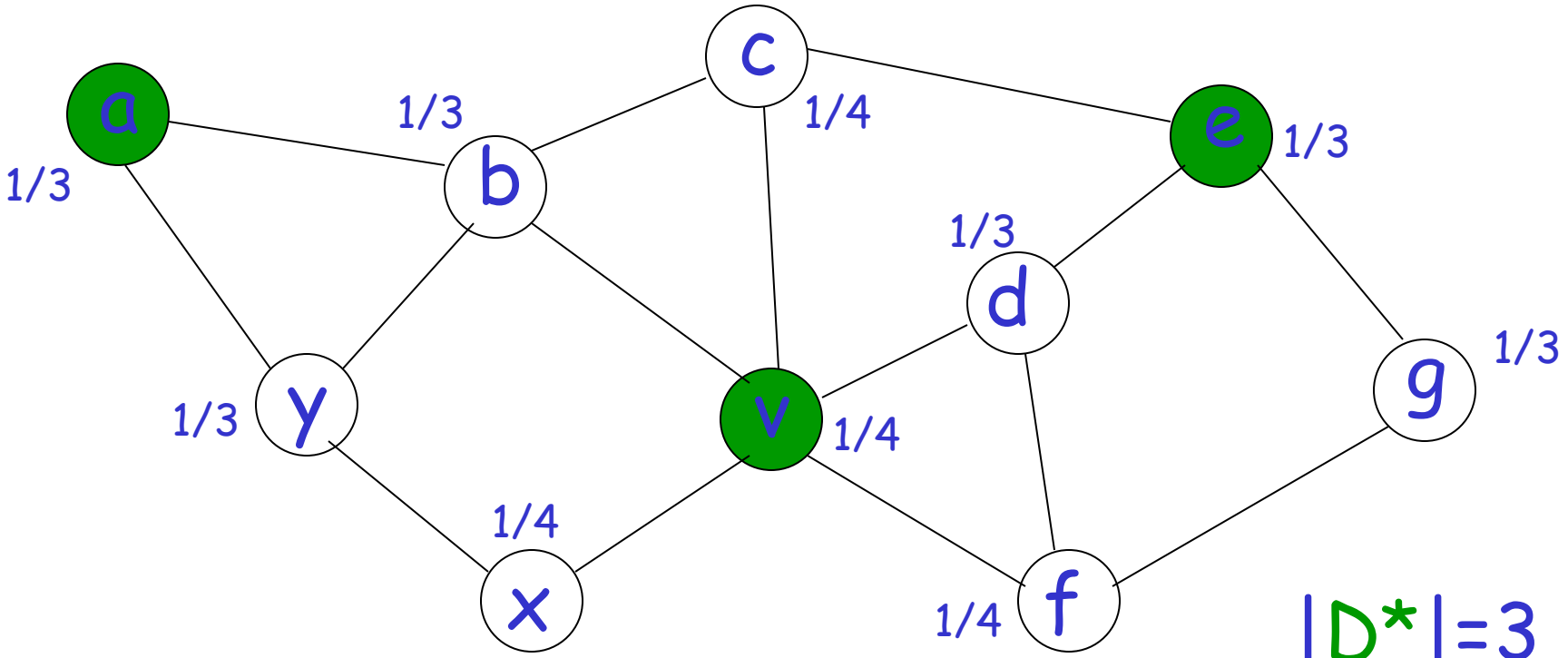
Centralized MDS Greedy Algorithm (2/4)

Proof: We prove the theorem by using amortized analysis. We call black the nodes in D, grey the nodes which are dominated (neighbors of nodes in D), and white all the non-dominated nodes. Each time we choose a new node of the dominating set (each greedy step), we have a cost increasing of 1, (since one node is added to the solution), but instead of assigning the whole cost to the node we have chosen, we distribute the cost equally among all newly dominated nodes.

Now, assume that we know a MDS D*. By definition, to each node which is not in D*, we can assign a neighbor from D*. By assigning each node to exactly one node of D*, the graph is decomposed into stars, each having a dominator (node in D*) as center, and non-dominators as leaves. Clearly, the cost of a MDS is 1 for each such star, or, in other words, each node of a star of k+1 nodes centered at $v \in D^*$ and of degree k (i.e., with k leaves) will cost 1/(k+1). But what the cost of such a star will be in the solution found by the GA?

Creating the stars

D* is is a dominating set of minimum size



Three stars centered at D* are created…

…and the corresponding costs are visualized

Centralized MDS Greedy Algorithm (3/4)

- Let us look at a single star with center v in D*. Assume that at the beginning of the current step of the *GA*, v is not black (i.e., it is either white or grey), and let w(v) be the number of current white nodes in the star of v in D*. First of all, notice that span(v)≥w(v), since w(v) considers only a subset of nodes adjacent to v.

- If the *GA* selects in this step a node v', some of these white nodes may become grey, so they will get charged a cost of 1/span(v') (observe this can happen iff v and v' are at distance at most 2 in *G*).

- By the greedy condition of the algorithm, span(v') ≥ span(v) ≥ w(v), since otherwise the algorithm should rather have chosen v for D instead of v'. Therefore, a white node of v becoming grey/black in the current step is charged by at most 1/w(v).

- Notice that after becoming grey/black, nodes do not get charged any more. Notice also that the cost that will be charged in the future to the remaining (if any) white nodes in the star of v will be **larger**, since w(v) is non-increasing w.r.t. the steps of the *GA*.

Centralized MDS Greedy Algorithm (4/4)

As a consequence, in the worst case (i.e., to maximize the cost charged to the star of v), no two nodes in the star of v become grey/black at the same step of the GA. Thus, in the worst case, denoting by $k \leq \delta(v)$ the degree of the star of v in D*, the first node gets charged by at most $1/(k+1)$, the second node gets charged by at most $1/k$, and so on. Thus, the total amortized cost of a star for the GA is at most

$$1/(k+1) + 1/k + \ldots + 1/2 + 1 = H(k+1) \leq H(\delta(v)+1) \leq H(\Delta+1) \leq 1 + \ln(\Delta+1)$$

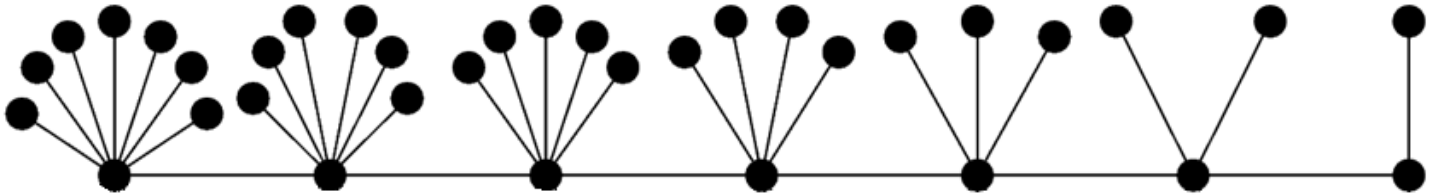
against a cost of 1 for the optimum. ■

Distributing (synchronously) the GA (1/2)

- Synchronous, non-anonymous, uniform MPS
- Proceed in phases, initially no node is in D
- Each phase has 3 steps:
  1. each node calculates its current span, by testing adjacent nodes (2 rounds);
  2. each node sends (span, ID) to all nodes within distance 2 (2 rounds);
  3. each node joins the dominating set D iff its (span, ID) is lexicographically higher than all others within distance 2 (1 round to notify neighbors); in this case, it will exit from the computation

Distributing (synchronously) the GA (2/2)

- It can be easily proven that the distributed algorithm has the same approximation ratio as the greedy algorithm: indeed, the analysis of the GA only involves nodes which are at distance at most 2 in G, as we have observed in the proof, which is exactly the tested neighborhood of the distributed algorithm
- However, the algorithm can be quite slow, since it can take $O(|D|)$ phases to terminate, where D is the returned dominated set. Look for instance at the following caterpillar graph (path of decreasing degrees) of n nodes:



$\Rightarrow$ Nodes along the "backbone" (of length $\Theta(\sqrt{n})$) add themselves to D sequentially from left to right $\Rightarrow$ $\Theta(\sqrt{n})$ phases (and rounds) are needed!

$\Rightarrow$ Via randomization, the greedy algorithm can be modified so as to terminate w.h.p. in $O(\log \Delta \log n)$ rounds, with an expected $O(\log \Delta)$-approximation ratio.

(In)approximability of MIC

- Concerning the MIC problem, the situation is very similar to MDS.
- More precisely, MIC is NP-hard and cannot be approximated within $(1-o(1)) \ln n$, unless P=NP.
- On the positive side, there exists a sequential $(1+\ln n)$-approximation algorithm for MIC.
- Moreover, the distributed GA for the MDS problem can be easily modified to solve the MIC problem in a distributed setting (it will essentially explore the 3-neighborood of a node instead of its 2-neighborood), and it will run in $O(|IC|)$, where IC is the returned identifying code.

Assignment

1. Provide a message complexity analysis of the distributed GA for the MDS problem
2. Run the greedy algorithm for the MDS problem on the following graph (the optimum is given by red nodes), and compute the apx ratio