Designs of Algorithms and Programming for Massive Data

DS8001

Andriy Miranskyy

Sep. 26, 2016

Outline

- Types of problems
- Computability
- Complexity Classes
- Approximate solutions of NP-complete problems: examples

Types of problems

Problems

- Optimization problems
  - Given multiple solutions with a value associated with it, find a solution with min (or max) value
  - E.g., find a shortest or longest path in a graph
- Decision problem
  - The answer is Yes or No {1,0}
  - E.g., will my software program ever halt?

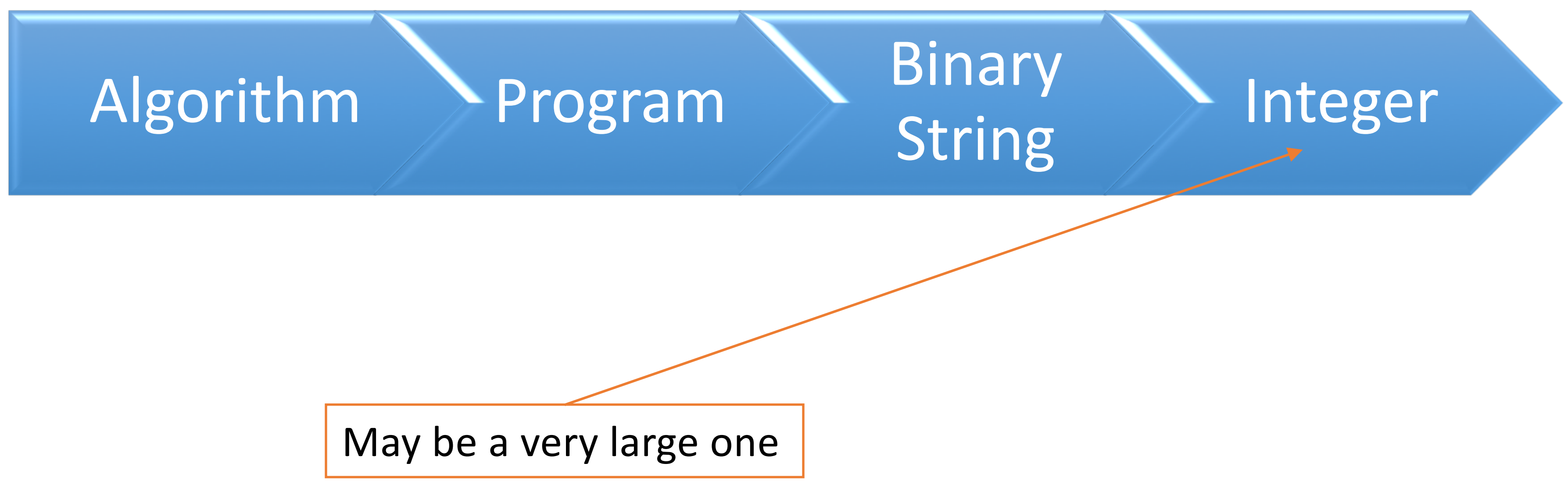
Problems

- Most optimization problems can be reformulated as a decision problem
- Example
    - Optimization: Find the shortest path in the graph
    - Decision:
        - Is the shortest path of length 4?
        - Is the shortest path of length 5?
        - Is the shortest path of length n?

- *Thus, we will focus on decision problems*

Computability

Gödel's Theorem: Problem



Gödel's Theorem: Decision Problem

- Map program (integer) to {Yes, No}

```
      . 0 1 0 1 1 0 …
        ↑ ↑ ↑ ↑ ↑ ↑
        0 1 2 3 4 5 …
```

- The top row is an infinite string of bits
  - Real number of infinite length
  - By adding a binary point, we get real number between 0 and 1
- The bottom row is a finite string of integers
- Which set is bigger: R or N?

Cantor’s Diagonal Argument

- Shows that |R| >> |N|
  - N -- countably infinite, cardinality $\aleph_0$
  - R -- uncountably infinite, cardinality $\aleph_1$
  - Unproven continuum hypothesis states that $\aleph_1 = 2^{\aleph_0}$
- Thus,
  - There are more problems then programs solving them
  - I.e., most problems will not have a corresponding program, i.e. are unsolvable

Gödel's Theorem: Corollary

- Corollaries*
  - Computational: large class of problems are undecidable
    - No computable function exists that gives correct answer to every question
  - Proof-theoretic: large number of statements that are neither provable nor refutable
    - Epiphany/luck is important

* Some people disagree with them

Complexity Classes

If a problem is computable, how hard are the computations?

Complexity Classes

- There are a lot of various classes and subclasses
  - https://complexityzoo.uwaterloo.ca/Complexity_Zoo
    - 533 classes and counting…
- We will focus on a couple of core ones

Uncomputable / Undecidable

- Decision problems unsolvable/undecidable by a Turing machine
- Halting problem*
  - Given computer program and a finite input, does it ever stop running (return result) or runs forever (e.g., in an infinite loop)?
- This does not imply that we should not try to find an approximate answer
  - We can do if for a small class of programs. E.g., we can construct a program that will verify these listings:
    - Compare

```
while (true) {
  print "Hello world!"
}
```

    - with

```
for (i = 1 to 100,000,000,000,000) {
  print "Hello world!"
}
```

  - Approximations: static code analysis, heuristics

* A good summary can be found at: http://stanford.edu/~jbooher/expos/computability_promys.pdf
Technically, halting problem belongs to Recursively Enumerable (RE) or Turing-recognizable set

R – Recursive Languages

- Decision problems solvable by a Turing machine
  - Albeit it may take a very long time: e.g., O (n!) or O(n^n) .

EXPTIME

- Decision problems that have an exponential runtime (or faster)
  - 2^{n^k}, where *k* in as integer
- A lot of complex games are in this category, e.g., Chess or Go
- For example, given 8x8 chess board and a set of figures on the board, will black win?
  - Solution: Play out all possible strategies and find out (exhaustive search)

P - polynomial

- Problems <u>solvable</u> in polynomial time
- We saw some simple examples in the previous lecture (e.g., sorting)
- Another example is finding the shortest path in the graph (think GPS navigator)
  - Dijkstra's algorithm: worst case performance is O[|E| + |V|log(|V|)], where IEI is the number of edges and |V| is the number of vertices.

NP – non-deterministic polynomial

- If a solution resulting in <u>“yes”</u> answer is given to me, then I can verify it in a polynomial time
  - If the answer is “no”, then it is a different class called co-NP
  - Size of the solution (a.k.a. certificate) should be polynomial
- Alternatively: it is solvable in polynomial time by *a <u>theoretical</u> non-deterministic Turing machine (NTM)*
  - NTM can take many computational paths simultaneously, but parallel paths cannot communicate.
    - Possibly using more compute resources that are available on Earth
- From practical perspective: think of it as a “lucky box” that, given a decision tree, will guide you to “yes” answer if at least one “yes” answer exists.
  - Essentially I am luckily guessing a sequence of steps that lead me to “yes” answer

Subset sum problem

- Given a set of integers of length *n*, is there a subset of integers that sums up to 0?
- Example
  - Set: {4, 2, 1, -6, 3}
  - Subset: 4+2-6 = 0
  - Answer: Yes
- Naïve solution using Deterministic Turing machine (TM):
  - Exhaustive search -- go through all possible subsets
  - Solution Complexity (worst-case): $O(2^n n)$
    - go over 2^n subsets, summing at most *n* elements per subset
  - Verification Complexity (worst-case): $O(n)$

* Better algorithm of order $O(2^{n/2})$ exists [2].

Subset sum problem

- Given *n* elements in a set
- Using NTM
  - Non-deterministic:
    - <u>Guess</u> first element of the subset
    - Guess second element of the subset
    - …
    - Guess *m*-th element of the subset
  - Deterministic:
    - Verify by summing up elements 1 through *m*

- What is the complexity?

Subset sum problem

- Note that we cannot use the approach to quickly find if the answer is “no”:
    - We will have to go through all subsets to make sure that none of them sum up to 0
    - We are back to exhaustive search

Reductions, NP-hard, and NP-complete

NP-hard

- It’s as least as hard as any problem in NP
- What does “as hard” means?
  - This is defined in terms of reductions

Reduction - P

- Solve a new decision problem A in poly-time, given a known decision problem B that can be solved in poly-time
- Given an instance of problem A convert it to an instance of a known problem B using a polynomial time reduction algorithm
    - Say, a particular graph consisting of specific edges and vertices
- We use "easiness" of B to prove "easiness" of A

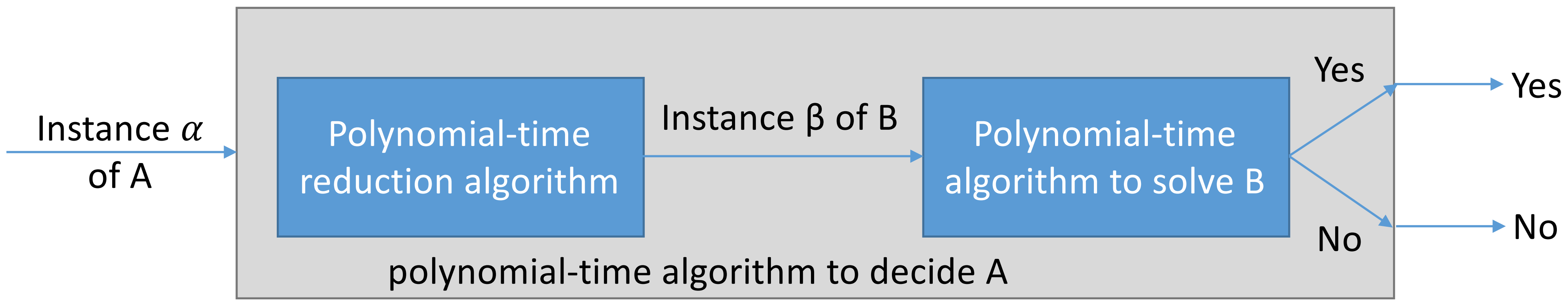

Instance α of A
Polynomial-time reduction algorithm
Instance β of B
Polynomial-time algorithm to solve B
Yes
No
Yes
No
polynomial-time algorithm to decide A

Figure adopted from [1, Fig 34.1]

Reduction - NP

- To show that problem is at least as hard as NP, we go the opposite way
    - since we focus on “as hard” rather than on “easy”
- Suppose that we have a known problem A for which we know that no poly-time solution exists
- Suppose that we have a reduction converting instances of A into instances of a new problem B in poly time
- Then a simple proof by contradiction shows that no polynomial time algorithm can exists for B, i.e., that it is “as hard” as A
    - This will be your homework (hint: use figure from the previous slide)

Reduction

- "Chicken and egg" conundrum:
  - Suppose that we have <u>a known problem A</u> for which we know that no poly-time solution exists.
  - But where do we get the first know problem?
- Cook's theorem (1971) [3] states that Boolean satisfiability problem is NP-Complete
  - We will get to NP-complete in a couple of slides
- Karp (1972) [4] provided a list of 21 NP-complete problems using reduction techniques

First NP problem: 3-satisfiability

- (x_1 or x_2 or x_3) and (not x_3 or x_4 or x_5) and (not x_5 or x_6 or x_7) and …
    - Each clause consists of three literals
        - I.e. has three terms
    - The number of clauses is finite
    - x_i is either true of false
- How to select all x_is, such that the output of the formula is true?

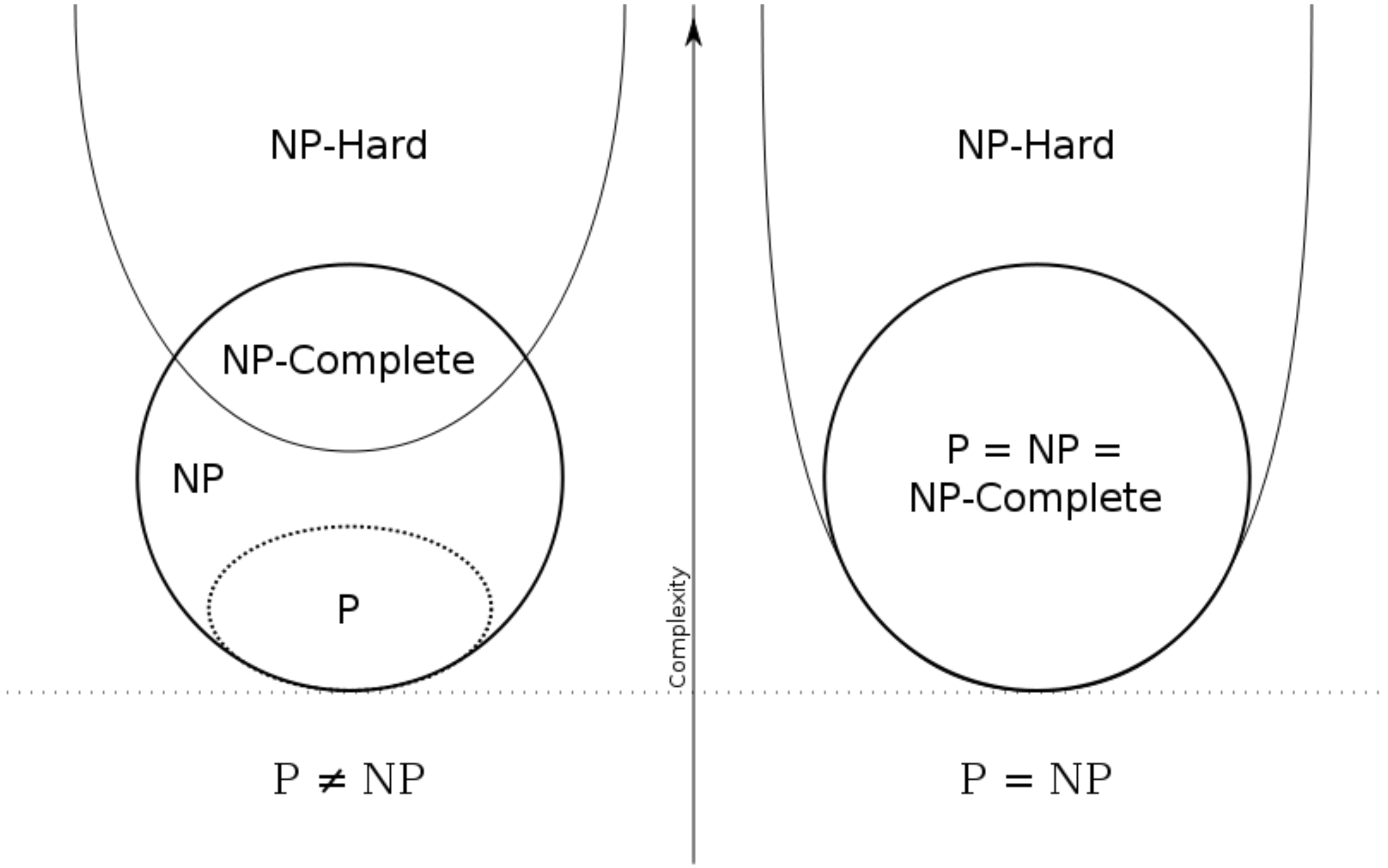
NP-complete

- Problem X is NP-complete if it is in NP and is NP-hard
- X is exactly as hard as everything else in NP – no harder but no easier
- 3-SAT and those to which it can be reduced are in NP-complete set
    - For details of how reduction is performed see [1, Ch. 34]

Unsolved problems: P ? NP

- Most believe that P != NP, but so far no one succeeded in proving or refuting this

Crude relation between P, NP, NP-complete, and NP-hard



Have to be careful if we start adding other complexity classes

Image taken from https://en.wikipedia.org/wiki/NP-completeness#/media/File:P_np_np-complete_np-hard.svg

A lot of open questions, e.g.:

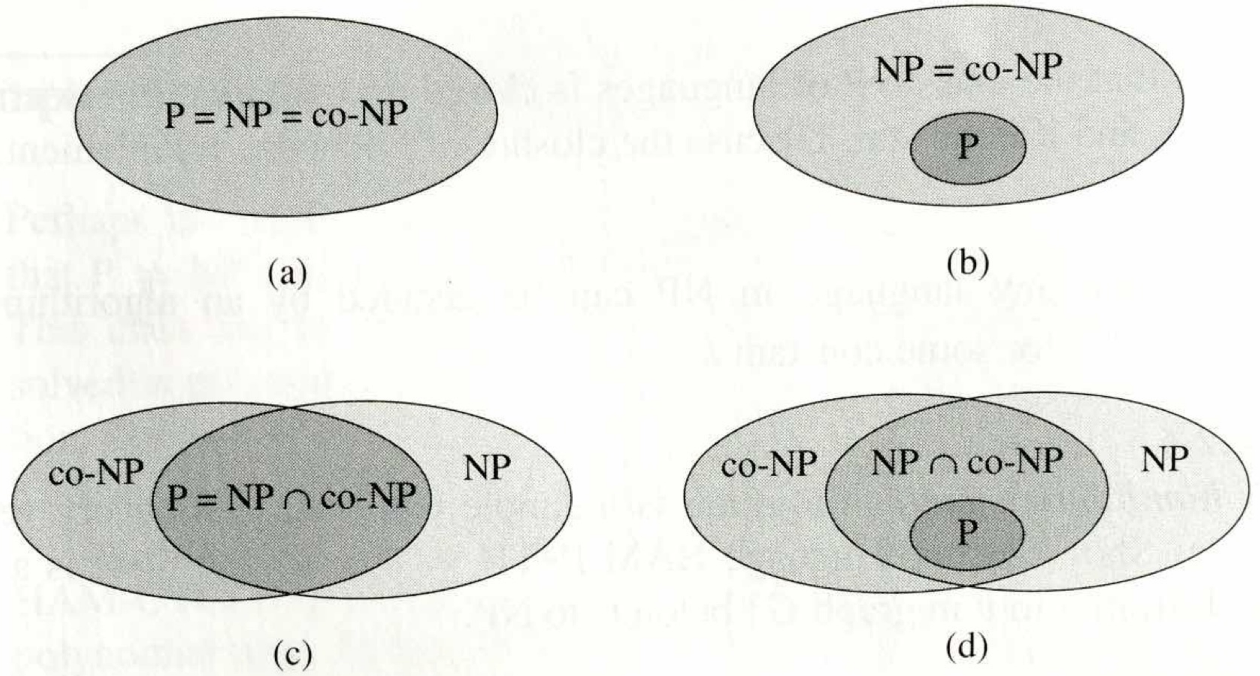


Figure 34.3 Four possibilities for relationships among complexity classes. In each diagram, one region enclosing another indicates a proper-subset relation. **(a)** P = NP = co-NP. Most researchers regard this possibility as the most unlikely. **(b)** If NP is closed under complement, then NP = co-NP, but it need not be the case that P = NP. **(c)** P = NP∩co-NP, but NP is not closed under complement. **(d)** NP ≠ co-NP and P ≠ NP ∩ co-NP. Most researchers regard this possibility as the most likely.

Taken from [1, p. 1065]

Intermezzo: Quantum Computer

- Quantum computer can be modeled by deterministic Turing machine (TM), which implies that it cannot solve problems that TM cannot solve (e.g., halting problem)
- However, there exists BQP (bounded error, quantum, polynomial time) class of problems (in NP, but not NP-complete) that can be solved in polynomial time with high probability, e.g.,
    - integer factorization (prime factorization is of particular interest to cryptographers)
    - discrete logarithm (find integer k, solving $b^k = g$)
- We do not know if quantum computer can solve NP-complete problems, but it is generally believed to be false [5].

Intermezzo: Quantum Computer

- Assuming
  - P ≠ NP
  - NP ≠ PSPACE

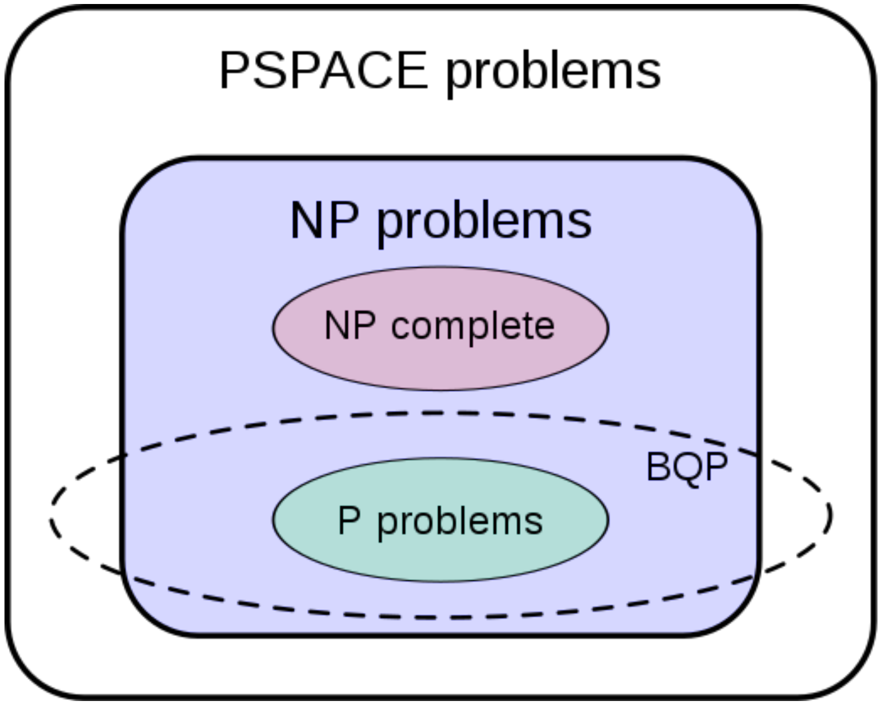


Image taken from https://en.wikipedia.org/wiki/BQP#/media/File:BQP_complexity_class_diagram.svg

Approximate solutions of NP-complete problems

Optimization

Greedy Heuristic and Linear Programming

The set-covering problem

- Given a finite set X and a family of subsets F, find the smallest number of subsets covering each point in X at least once
    - Each point from X is covered by one or more subsets in F
- The problem is NP-complete

Approximation

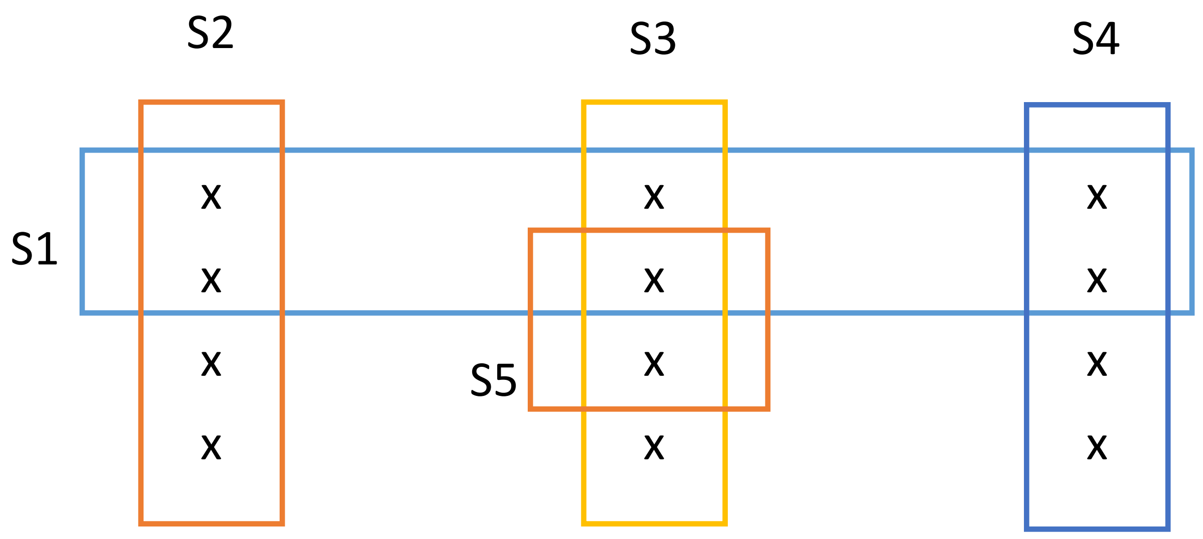
Greedy Heuristic

The set-covering problem Greedy Approximation

- All the points are marked as non-covered
- Selects set with the largest number of non-covered points
  - Largest → Greedy
- Once the set is selected, the set's points are marked as covered
- The process repeats itself until all the points are covered at least once

- Time complexity: polynomial

Example



Optimal solution: S2, S3, S4
Greedy solution: S1, S2, S3, S4

How bad is the approximation?

- One can show* that the approximation ratio between the greedy and optimal covering is given by harmonic sum

$$H(n) = \sum_{i=1}^{n} 1/i \leq \ln(n) + 1, H(0) = 0,$$

  where *n* is the number of points in X.

- E.g., if my set X has 100 data points, then my approximation, at worst, will be ln(100) + 1 $\approx$ 5.6 times larger than the optimal one.

* See [1, Sec. 35.3] for the proof

Approximation

Linear Integer Programming

Formulation

Minimize the number of subsets,	$Min \sum_{S \in F} S_i$
such that	such that
each point in X is covered at least once	$PS \geq 1$
Each subset takes the value of 0 (exclude) or 1 (include)	binary variables: *S*

Example

- Suppose we have four sets
  - S_1, S_2, S_3, and S_4
- Covering five points
  - p_1, p_2, p_3, p_4, and p_5

Points	**Sets**			
	S_1	S_2	S_3	S_4
p_1		1		
p_2	1	1	1	
p_3	1		1	
p_4			1	1
p_5				1

Matrix P above, contains 0s in empty cells

Minimize $S_1 + S_2 + S_3 + S_4$,

subject to

p_1: $S_2 \geq 1$,
p_2: $S_1 + S_2 + S_3 \geq 1$,
p_3: $S_1 + S_3 \geq 1$,
p_4: $S_3 + S_4 \geq 1$,
p_5: $S_4 \geq 1$,

Solution: S_1=0, S_2=1, S_3=1, and S_4=1;
i.e., to cover all points we need {S_2, S_3, S_4}.

Complexity of the approximation

- In general, solution of Integer Programming problems are NP-hard.
- Solution of Binary Integer Programming problems are NP-complete.
    - One of 21 Carp’s problems
- However, if a constraint matrix P is totally unimodular and the right hand side of constraints consists of integer values, then the problems can be solved using Simplex.
- Simplex’s worst-case time complexity is exponential, but it is remarkably efficient in practice.
- Approximation ratio is of order log(n) [6, Sec. 13]

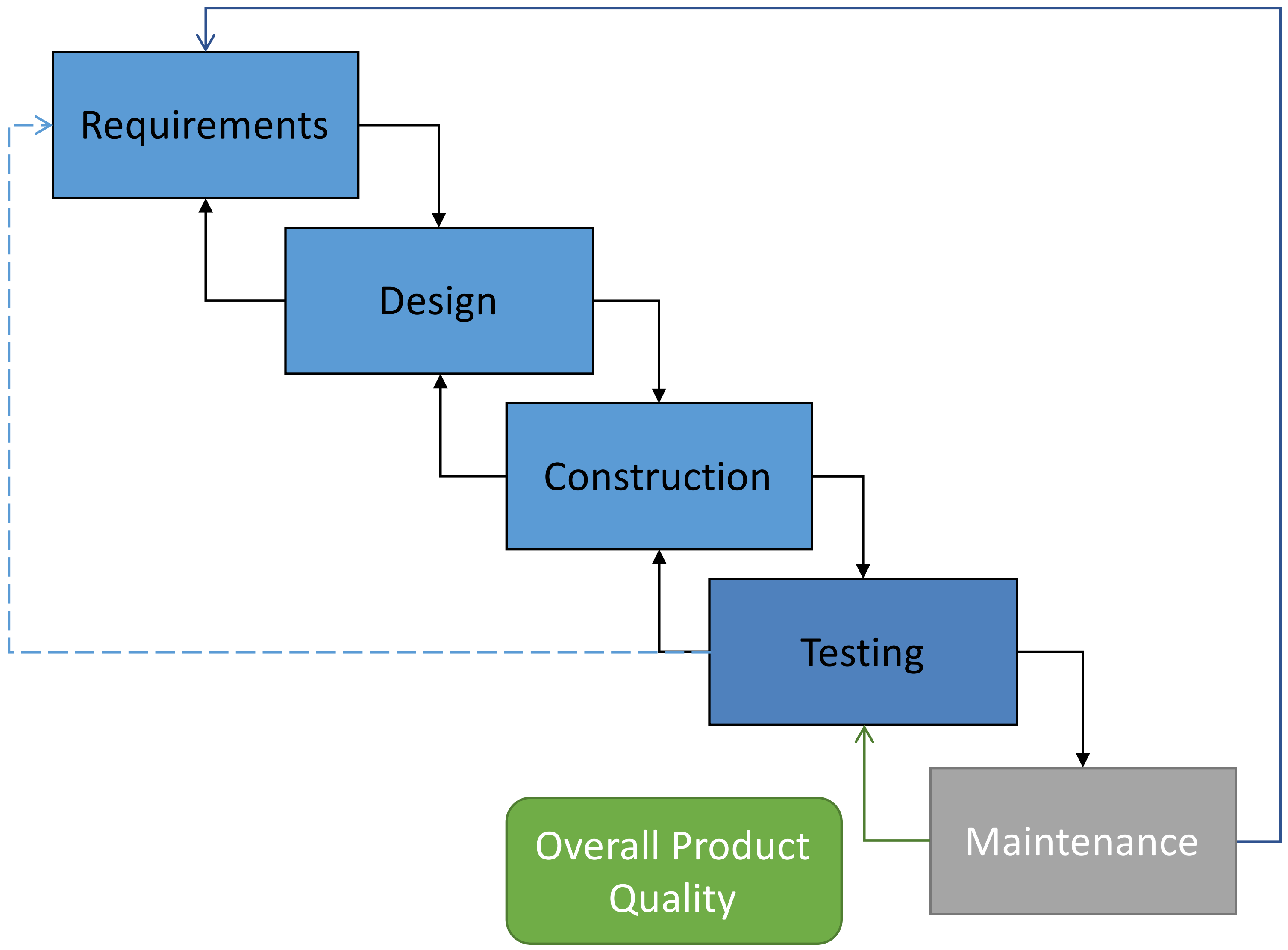
Selection and Prioritization of Customers for Operational and Usage Profiling

Sample Case Study

Outline

- What is customer profiling?
- Why is it important?
- Types of profiles
- Selection and prioritization of customers for profiling

Software Engineering: Conceptual Model



Customer Profiling: Goal

- Understand
  - Customer environment / configuration
  - Usage patterns
  - Workloads
- Identify
  - Gaps in the processes
  - Missing features
- Feed the info back to requirement engineers and testers

Outline

- What is customer profiling?
- Why is it important?
- Usage Profiles
- Selection of customers for profiling

Usage Profile: Example

- Questionnaire
  - How is software used?
  - What is the environment?
  - What are the workloads?

Outline

- What is customer profiling?
- Why is it important?
- Types of Profiles
- Selection of customers for profiling
    - Criteria
    - Selection Process

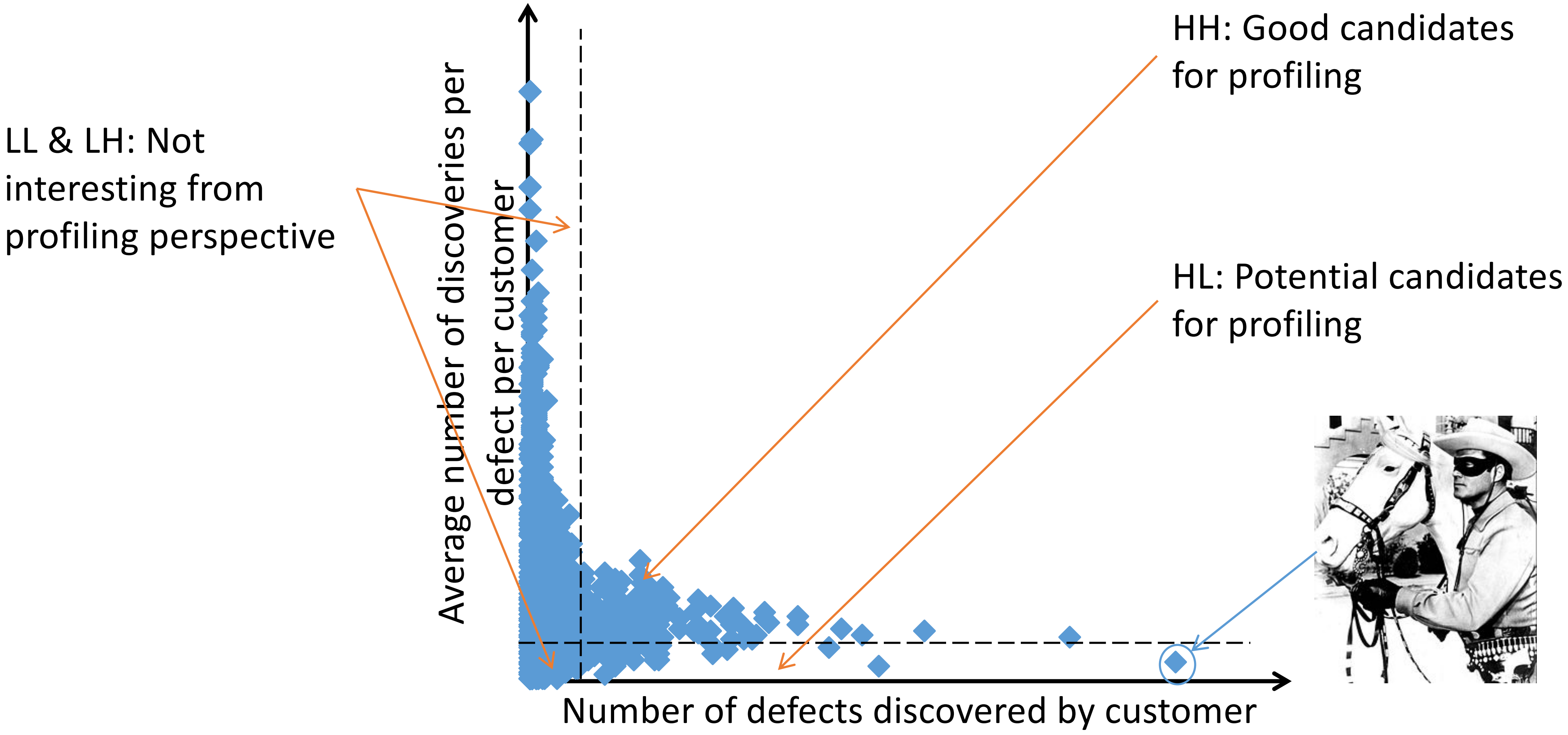
Prioritization Dimensions

1. Total number of defects found by a given customer
2. Average number of discoveries per defect found by a given customer

Customer Prioritization Criteria

Average number of discoveries per defect found by a given customer	Total number of defects found by a given customer	
	Low	**High**
High	**LH:** Not interesting from profiling perspective, but defects are	**HH:** Ideal candidate for profiling
Low	**LL:** Not interesting from profiling perspective	**HL:** Potential candidate for profiling

Customer Prioritization Criteria: Example (Industrial Data [7])



Outline

- What is customer profiling?
- Why is it important?
- Types of Profiles
- Selection of customers for profiling
    - Criteria
    - Selection Process

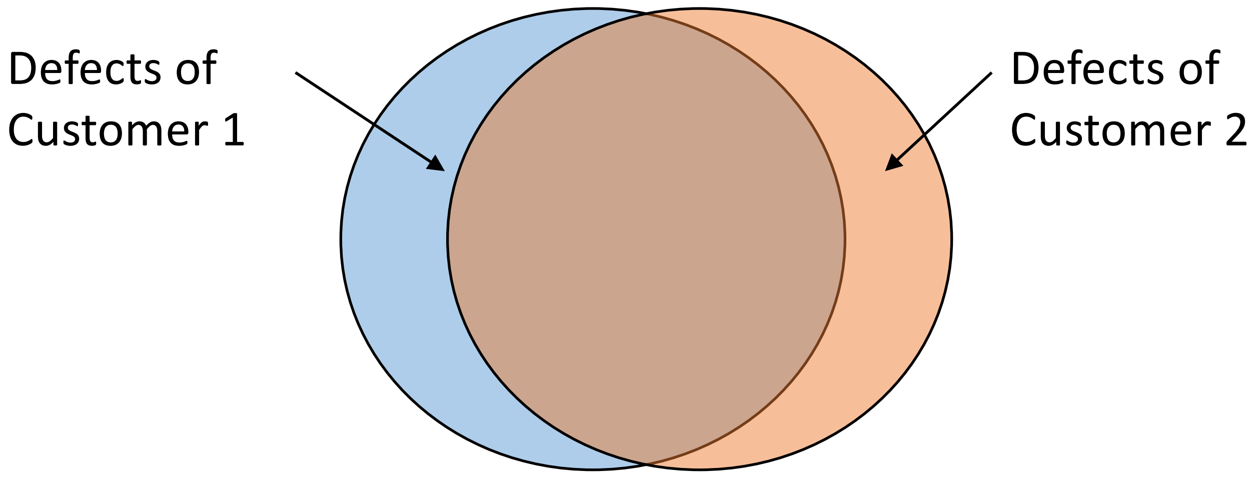
Exhaustive Approach

- An exhaustive approach is to profile all customers
  - Difficult for large customer base
    - Storage requirements, time and resource constraints
    - Moreover, customers may refuse to provide access to their systems due to confidentiality and legal reasons
  - Duplicate information being gathered

Idea: information about customer defects can help us to narrow down a list of candidate customers to profile

Manual Customer Selection Issues: Example

- Avoid duplication of efforts
  - Two customers discovered a large number of defects that are also frequently discovered by other customers
  - Are the defect sets overlapping?



Manual Customer Selection Issues

- Manual selection of customers can become cumbersome if a product is used by thousands of customers discovering hundreds of defects
- We need to find techniques that will allow us to:
  - Minimize the number of customers for profiling while maximizing the total number of discovered defects
  - Prioritize a minimal set of customers (once identified) by the total number of discovered defects per customer

Minimization of Customer Set

- Identify a minimal set of customers
  - Such that all defects are encountered by at least one customer in the minimal set
- You can use Linear Programming (Simplex) to solve the problem
  - Formulate minimization task as a Binary Integer Programming (BIP)*
  - Due to special formulation, the problem can be solved using standard simplex algorithm

* BIPs are, in general, NP-complete

Linear Programming Solution

Minimization of Customer Set

$$
\begin{aligned}
&\text{Minimize } w_1c_1 + w_2c_2 + \ldots + w_Nc_N\,,\\
&\text{subject to}\\
&d_1 : p_{1,1}c_1 + p_{1,2}c_2 + \ldots + p_{1,N}c_N \ge 1,\\
&d_2 : p_{2,1}c_1 + p_{2,2}c_2 + \ldots + p_{2,N}c_N \ge 1,\\
&\ldots\\
&d_M : p_{M,1}c_1 + p_{M,2}c_2 + \ldots + p_{M,N}c_N \ge 1,\\
&\text{binary variables: } c_1, c_2, \ldots, c_N\,;
\end{aligned}
$$

- $N \equiv$ total number of customers;
- $M \equiv$ total number of defects;
- $c_i \equiv$ *i*-th customer ($i = 1 \ldots N$), $c_i = 1$ if the *i*-th customer is included in the minimal set of customers to profile and is 0 otherwise;
- $d_j \equiv$ *j*-th defect (j=1…M);
- $p_{i,j} \equiv$ binary variable showing discovery of the *j*-th defect by the *i*-th customer, $p_{i,j}$=1 if the *i*-th customer discovered the *j*-th defect and is 0 otherwise;
- $w_i \equiv$ *i*-th customer weight.

Minimization of Customer Set. Customer weight

- If we want to emphasize “importance” of the *i*-th customer, then we should decrease weight w_i relative to the weight of the remaining customers.
    - For example, w_i can be proportional to the difficulty of gathering information from the *i*-th customer and inversely proportional to the average number of discoveries per defect found by the *i*-th customer.
    - If all customers are considered equal then w_i =1 for all *i.*

Minimization of Customer Set. Complexity

- In short form BIP problem can be written as

$$\text{Minimize } w^T c,$$

$$\text{subject to } pc \geq 1,$$

$$\text{binary variables: } c.$$

- This approach should provide us with optimal solution.
- In general, solution of BIP problems is NP-complete.
- However, if a constraint matrix *p* is totally unimodular and the right hand side of constraints consists of integer values, then the problems can be solved efficiently.
- Our problem formulation falls into this category of BIP problems.

Minimization of Customer Set: Example

- Suppose we have four customers
  - c_1, c_2, c_3, and c_4
- The customers discovered five defects
  - d_1, d_2, d_3, d_4, and d_5

Defects	Customers			
	***c*₁**	***c*₂**	***c*₃**	***c*₄**
d_1		×		
d_2	×	×	×	
d_3	×		×	
d_4			×	×
d_5				×

Minimize $c_1 + c_2 + c_3 + c_4$,

subject to

$$d_1 : c_2 \geq 1,$$

$$d_2 : c_1 + c_2 + c_3 \geq 1,$$

$$d_3 : c_1 + c_3 \geq 1,$$

$$d_4 : c_3 + c_4 \geq 1,$$

$$d_5 : c_4 \geq 1,$$

binary variables: c_1, c_2, c_3, c_4.

Solution: c_1=0, c_2=1, c_3=1, and c_4=1;
i.e., the minimal set of customers for profiling that cover all defects is {c_2, c_3, c_4}.

Greedy Heuristic Solution

Prioritization of Customers within the Minimal Set

- The customer prioritization heuristic greedily selects customer with the largest number of non-covered defects
- Once the customer is selected, the customer’s defects are marked as covered
- The process repeats itself until all the defects are covered at least once

We avoid applying this heuristic to the initial set of customers directly (skipping the BIP step) because of the sub-optimality of the heuristics

Prioritization of Customers within the Minimal Set: Example

- The minimal set of customers for prioritization is $\{c_2, c_3, c_4\}$. "Non-covered" defects per customer are $\{\{d_1, d_2\}, \{d_2, d_3, d_4\}, \{d_4, d_5\}\}$, respectively.
- Since c_3 discovered the largest number of defects, pick c_3 as the first customer to profile.
- Mark defects d_2, d_3, and d_4 as "covered" and remove them from the "non-covered" list.
- The defects list is changed to $\{\{d_1\}, \{\emptyset\}, \{d_5\}\}$: customers c_2 and c_4 have one uncovered defect.
- Arbitrarily pick c_2 as the second customer for profiling and c_4 as the third one.

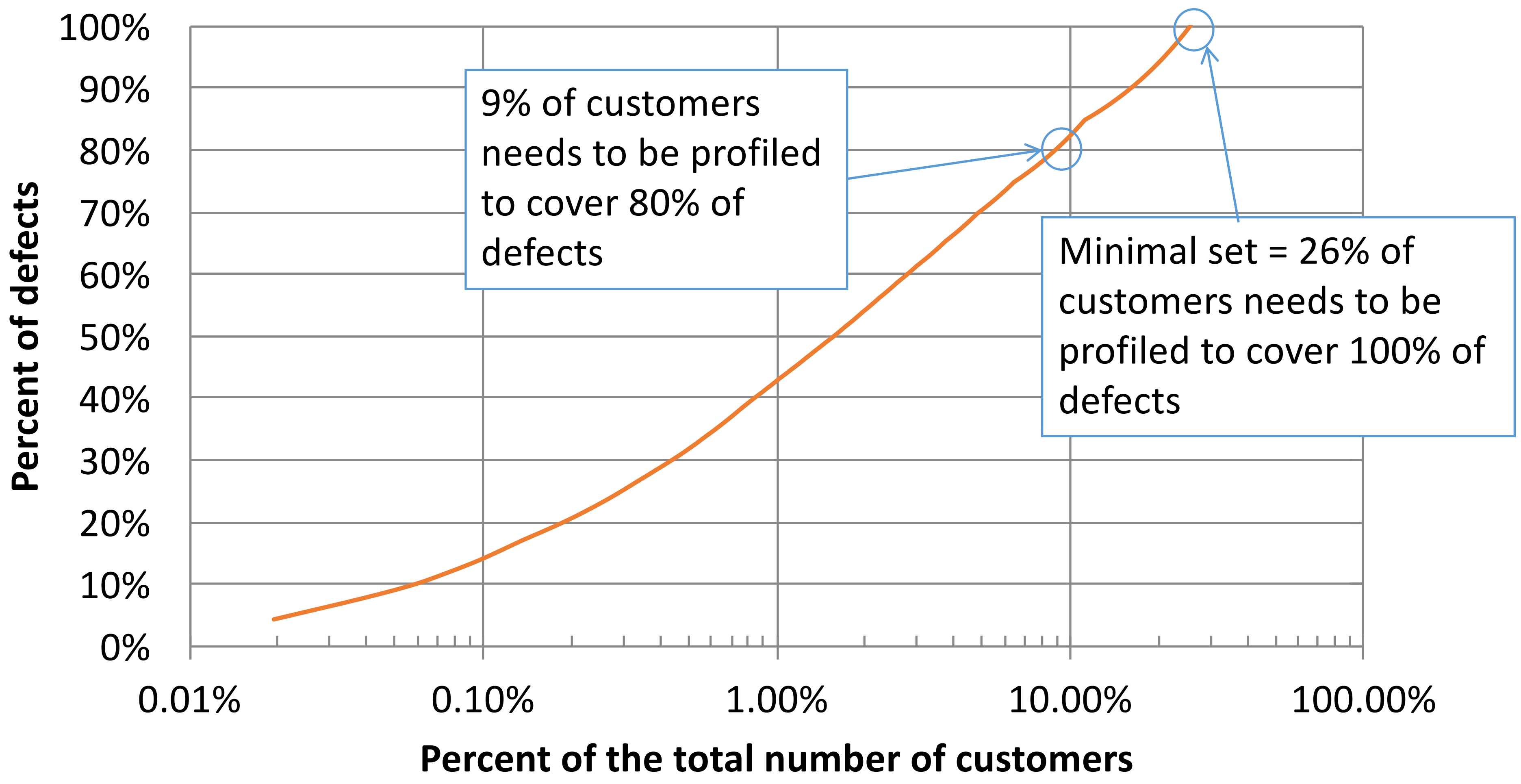
Defects	**Customers**		
	c_2	c_3	c_4
d_1	×		
d_2	×	×	
d_3		×	
d_4		×	×
d_5			×

Prioritization of Customers within the Minimal Set: Greedy Heuristics*

- I/O:
    - Input: set of clients and associated defects *S*
    - Output: list of customer *L*
        - Ordered from largest number of defects to smallest
- Set *L* to [ ]
- While set of customer *S* is not empty
    - Greedily select customer *C* with the largest number of non-covered defects
    - Mark customer *C* defects as covered
    - Push customer name to *L*
    - Remove *C from S*
- Return *L*

* Avoid applying this heuristic to the initial set of customers directly (skipping the BIP step) because of the sub-optimality of the heuristics

Percentage of the Total Number of Customers Needed to Cover a Certain Percentage of Defects: Example (Industrial Data [7])



Case Study Summary

- Customer profiling
  - Post-release SE process
  - Gathers information about customers and feeds it to requirement engineers and testers, enhancing internal processes
- Types of profiles
  - Operational Profiles
  - Usage Profiles
- Customer selection technique
  - Minimize customer set
  - Prioritizes a minimal set of customers

Summary

- Types of problems
- Computability
- Complexity Classes
- Examples of approximations for NP-complete problems

References

1. Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2009) Introduction to Algorithms. 3rd ed. McGraw-Hill Higher Education
2. Ellis Horowitz and Sartaj Sahni. (1974). Computing Partitions with Applications to the Knapsack Problem. J. ACM 21, 2, 277-292. DOI=http://dx.doi.org/10.1145/321812.321823
3. Cook, Stephen (1971). "The complexity of theorem proving procedures". Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158.
4. Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems". In Raymond E. Miller and James W. Thatcher (editors). Complexity of Computer Computations. New York: Plenum. pp. 85–103. ISBN 0-306-30707-3.
5. Bernstein, Ethan; Vazirani, Umesh (1997). "Quantum Complexity Theory". SIAM Journal on Computing. 26 (5): 1411. doi:10.1137/S0097539796300921.
6. Vazirani, Vijay V. (2001), Approximation Algorithms, Springer-Verlag, ISBN 3-540-65367-8
7. A. Miranskyy, E. Cialini, and D. Godwin. (2009). Selection of customers for operational and usage profiling. In Proc. of the 2nd Int. Workshop on Testing Database Systems (DBTest '09). Article 7, 6 pages.