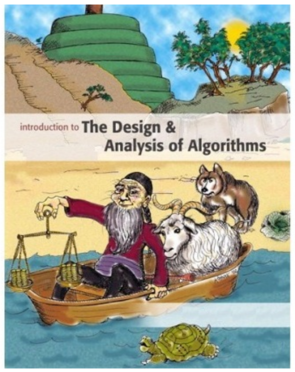
Introduction to

Algorithm Design and Analysis

[20] NP Complete Problems 2



Yu Huang

http://cs.nju.edu.cn/yuhuang
Institute of Computer Software
Nanjing University

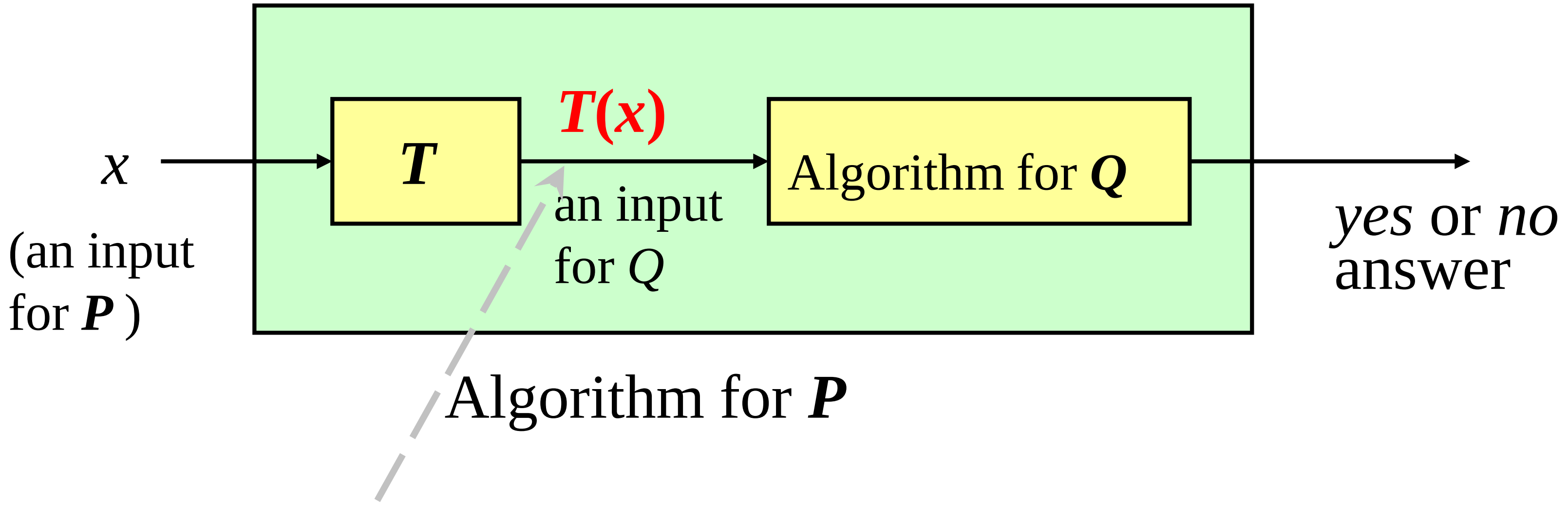
In the Last Class…

- Decision Problem
- The Class P
- The Class NP
- Reduction

In This Class

- **Reduction between problems**
- **NP-Complete Problems**
  - o Definition
  - o Proof of NP-Completeness
- **Other advanced topics**
  - o Advanced algorithms
  - o Advanced computation models

Reduction



The correct answer for ***P*** on x is *yes* **if and only if** the correct answer for **Q** on *T*(*x*) is *yes*.

NP-complete Problems

- **A problem *Q* is NP-hard if every problem *P* in NP is reducible to *Q*, that is $P \le_P Q$.**

  (which means that *Q* is at least as hard as any problem in **NP**)

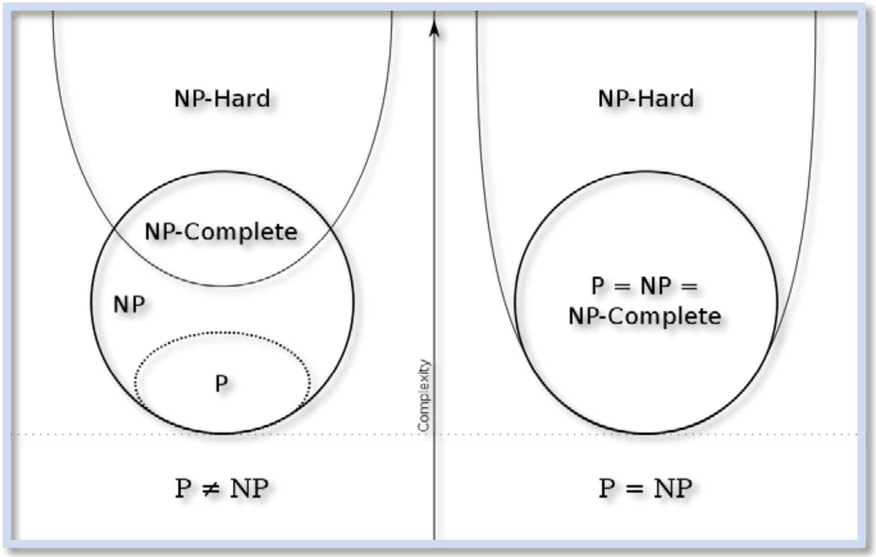
- **A problem *Q* is NP-complete if it is in NP and is NP-hard.**

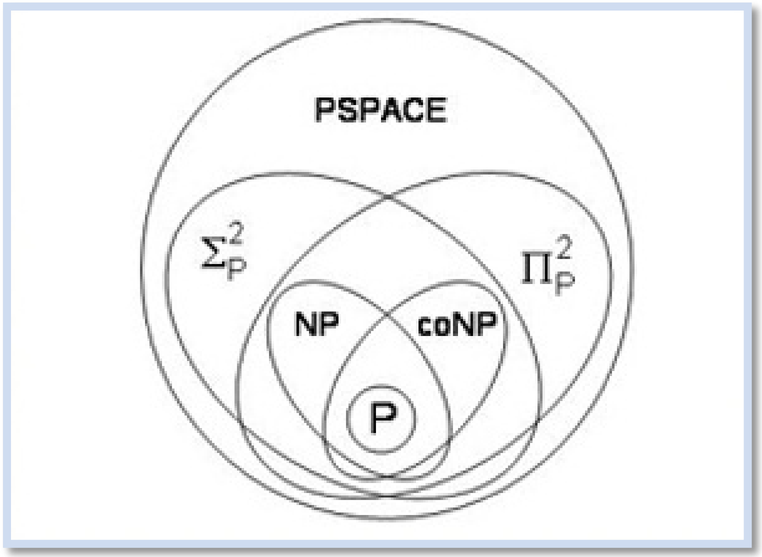
  (which means that *Q* is at most as hard as to be solved by a polynomially bounded nondeterministic algorithm)

P and NP - Revisited

- **Intuition implies that NP is a much larger set than P.**
  - No one problem in **NP** has been proved not in **P**.
- **If any NP -completed problem is in P , then NP =P .**
  - Which means that every problems in **NP** can be reducible to a problem in **P** !

P and NP - Revisited



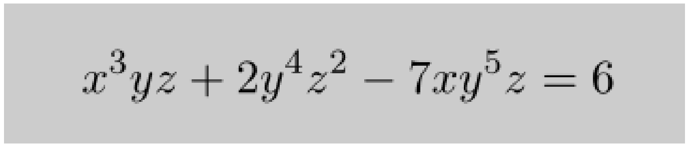


Example of an NP-hard Problem

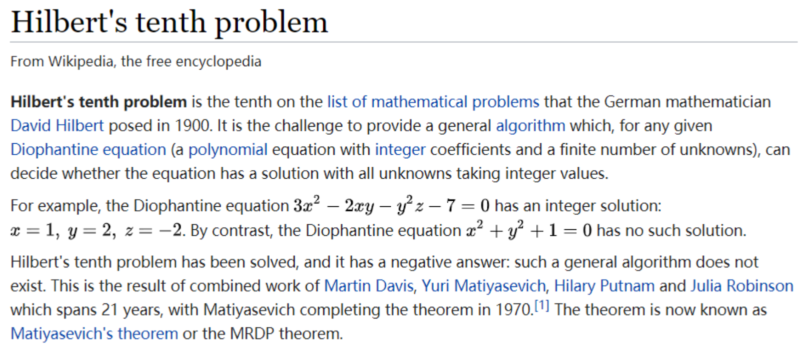
- **Halt problem: Given an arbitrary deterministic algorithm *A* and an input *I*, does *A* with input *I* ever terminate?**
  - A well-known **undecidable** problem, of course not in NP.
  - Satisfiability problem is reducible to it.
    - Construct an algorithm *A* whose input is a propositional formula *X*. If *X* has *n* variables then *A* tries out all 2^n possible truth assignments and verifies if *X* is satisfiable. If it is satisfiable then *A* stops. Otherwise, *A* enters an infinite loop.
    - So, *A* halts on *X* iff. *X* is satisfiable.

More Undecidable Problems

- Arithmetical SAT



$$x^3yz + 2y^4z^2 - 7xy^5z = 6$$



Hilbert's tenth problem

From Wikipedia, the free encyclopedia

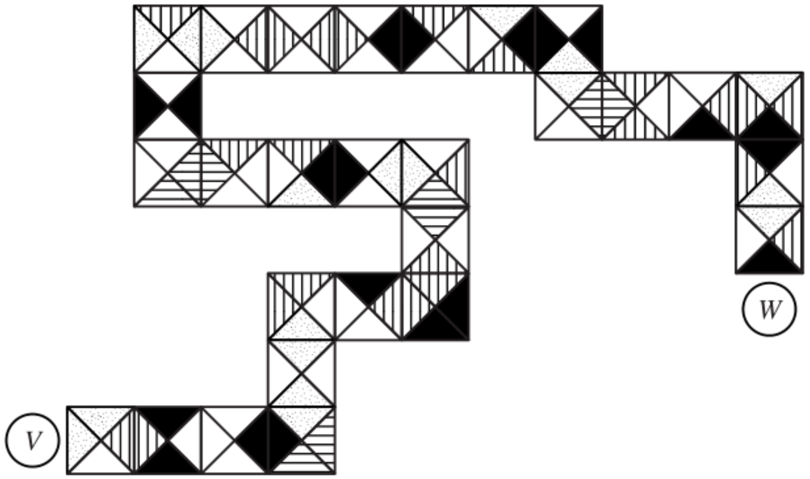
Hilbert's tenth problem is the tenth on the list of mathematical problems that the German mathematician David Hilbert posed in 1900. It is the challenge to provide a general algorithm which, for any given Diophantine equation (a polynomial equation with integer coefficients and a finite number of unknowns), can decide whether the equation has a solution with all unknowns taking integer values.

For example, the Diophantine equation $3x^2 - 2xy - y^2z - 7 = 0$ has an integer solution: $x = 1,\ y = 2,\ z = -2$. By contrast, the Diophantine equation $x^2 + y^2 + 1 = 0$ has no such solution.

Hilbert's tenth problem has been solved, and it has a negative answer: such a general algorithm does not exist. This is the result of combined work of Martin Davis, Yuri Matiyasevich, Hilary Putnam and Julia Robinson which spans 21 years, with Matiyasevich completing the theorem in 1970.[1] The theorem is now known as Matiyasevich's theorem or the MRDP theorem.

https://book.douban.com/subject/3425827/

- The *tiling* problem



https://book.douban.com/subject/3199995/

Procedure for NP-Completeness

- **Knowledge: P is NPC**
- **Task: to prove that Q is NPC**
- **Approach: to reduce P to Q**
  - For any $R \in$ **NP** , $R \le_P P$
  - Show $P \le_P Q$
  - Then $R \le_P Q$, by transitivity of reductions
  - Done. Q is NP-complete (given that Q has been proven in NP)

First Known NPC Problem

- **Cook's theorem:**
  - The SAT problem is NP-complete.
- **Reduction as tool for proving NP-completeness**
  - Since CNF-SAT is known to be NP-hard, then all the problems, to which CNF-SAT is reducible, are also NP-hard. So, the formidable task of proving NP-complete is transformed into relatively easy task of proving of being in **NP**.

Proof of Cook's Theorem

COOK, S. 1971.

The complexity of theorem-proving procedures.

In

Conference Record of

3rd Annual ACM Symposium on Theory of Computing.

ACM New York, pp. 151–158.

Stephen Arthur Cook: b.1939 in Buffalo, NY. Ph.D of Harvard. Professor of Toronto Univ. 1982 Turing Award winner. The Turing Award lecture: "An Overview of Computational Complexity", CACM, June 1983, pp.400-8

Satisfiability Problem

- **CNF**
  - A literal is a Boolean variable or a negated Boolean variable, as x or $\bar{x}$
  - A clause is several literals connected with '∨'s, as $x_1 \vee \overline{x_2}$
  - A CNF formula is several clause connected with ∧ s
- **CNF-SAT problem**
  - Is a given CNF formula satisfiable, i.e. taking the value TRUE on some assignments for all x_i.
- **A special case: 3-SAT**
  - 3-SAT: each clause can contain at most 3 literals

Proof by Reduction

- **The CNF-SAT problem is NP-complete.**
- **Prove problem Q is NP-complete, given a problem P known to be NP-complete**
  - For all R∈**NP**, $R \le_P P$;
  - **Show $P \le_P Q$**;
  - By transitivity of reduction, for all R∈**NP**, $R \le_P Q$;
  - So, Q is **NP**-hard;
  - If Q is in **NP** as well, then Q is **NP**-complete.

Proving by Reduction

- The six basic NPC problems

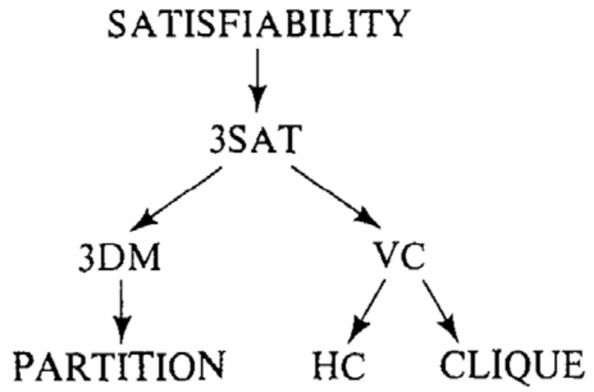

$$\begin{tikzcd}
 & \text{SATISFIABILITY} \arrow[d] & & \\
 & \text{3SAT} \arrow[dl] \arrow[dr] & & \\
\text{3DM} \arrow[d] & & \text{VC} \arrow[dl] \arrow[dr] & \\
\text{PARTITION} & \text{HC} & & \text{CLIQUE}
\end{tikzcd}$$

Figure 3.1 Diagram of the sequence of transformations used to prove that the six basic problems are NP-complete.

CLIQUE is in NP

```
void nondeteClique(graph G; int n, k)
    S = genCertif();                              → in O(n)
    if (S is a clique of size k) Output "accept";  } → in O(k^2)
    else Output "reject";                          }
    return;
```

So, we have an algorithm for the maximal clique problem with the complexity of $O(n+k^2)=\mathbf{O(n^2)}$

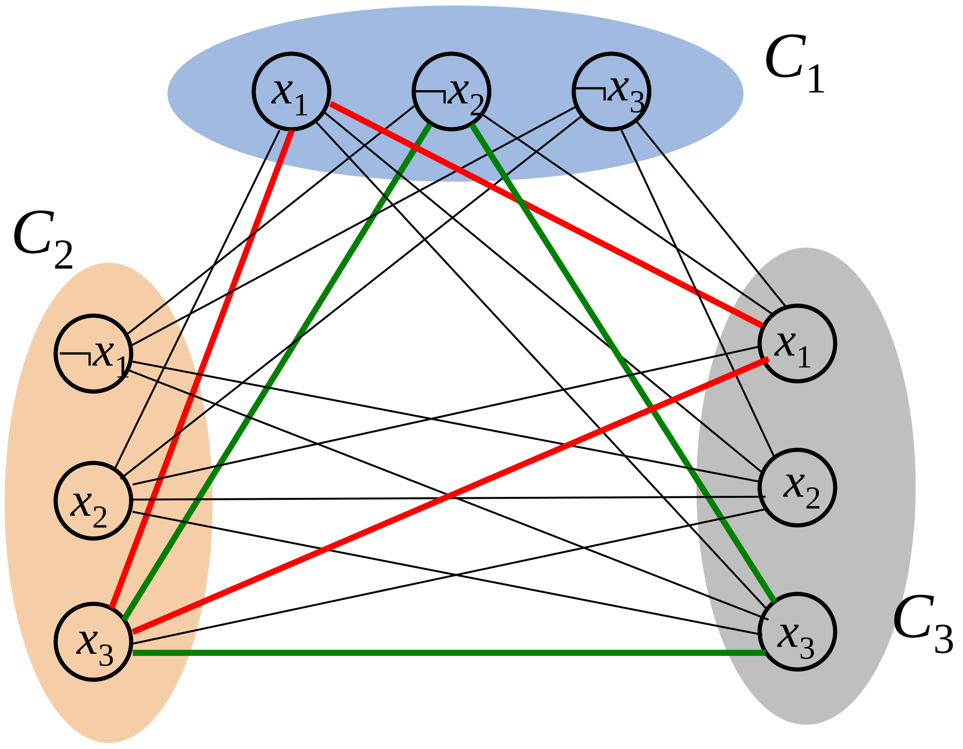
CNF-SAT to Clique

- **Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ be a formula in CNF-3 with k clauses. For $r = 1,2,\ldots,k$, each clause $C_r = (l_1^r \vee l_2^r \vee l_3^r)$, l_i^r is x_i or $\neg x_i$, any of the variables in the formula.**
- **A graph can be constructed as follows. For each C_r, create a triple of vertices v_1^r, v_2^r and v_3^r, and create edges between v_i^r and v_j^s if and only if:**
    - they are in different triples, i.e. $r \neq s$, and
    - they do not correspond to the literals negating each other

(Note: there is no edges within one triple)

3-CNF Graph

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$



Two of satisfying assignments:

x_1=1/0, x_2=0; x_3=1, or

x_1=1, x_2=1/0, x_3=1

For corresponding clique, pick one "true" literal from each triple

Clique Problem is NP-Complete

- **ϕ, with *k* clauses, is satisfiable iff. The graph *G* has a clique of size *k*.**
- **Proof: $\Rightarrow$**
    - Suppose that ϕ has a satisfying assignment.
    - Then **there is at least one "true" literal in each clause**. Picking such a literal from each clause, their corresponding vertices in *G* can be proved to be a clique, since any two of them are in different triples and cannot be complements to each other(they are both true).

Known NP-Complete Problems

- **Garey & Johnson:** ***Computer and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979**
  - About 300 problems, grouped in 12 categories:

1. Graph Theory 2. Network Design 3. Set and Partition
4. Storing and Retrieving 5. Sorting and Scheduling
6. Mathematical Planning 7. Algebra and Number Theory
8. Games and Puzzles 9. Logic
10. Automata and Theory of Languages
11. Optimization of Programs 12. Miscellaneous

Advanced Topics

- **Solving hard problems**
  - Approximation algorithms
  - Randomized algorithms
- **Solving more complex problems**
  - Online algorithms
  - External memory models
  - Distributed computation models

Approximation

- **Make modifications on the problem**
  - o Restrictions on the input
  - o Change the criteria for the output
  - o Find new abstractions for a practical situation
- **Find approximate solution**
  - o Approximation algorithm
  - o Bound of the errors

Bin Packing Problem

- **Suppose we have**
  - An unlimited number of bins each of capacity one, and *n* objects with sizes $s_1, s_2, \ldots, s_n$ where $0<s_i\leq 1$ (s_i are rational numbers)
- ***Optimization problem***
  - Determine the smallest number of bins into which the objects can be packets (and find an optimal packing) .
- **Bin packing is a NPC problem**

Feasible Solution

- **Set of feasible solutions**
  - For any given input $I=\{s_1,s_2,\dots,s_n\}$, the feasible solution set, $FS(I)$ is the set of all **valid packing** using any number of bins.
  - In other words, that is the set of all partitions of I into disjoint subsets $T_1,T_2,\dots,T_p$, for some p, such that the **total of the s_i in any subset is at most 1**.

Optimal Solution

- **In the bin packing problem, the optimization parameter is the number of bins used.**
  - For any given input I and a feasible solution x, ***val***(I,x) is the value of the optimization parameter.
  - For a given input I, the optimum value, ***opt***$(I)=\min\{val(I,x) \mid x \in FS(I)\}$
- **An optimal solution for *I* is a feasible solution which achieves the optimum value.**

Approximate Algorithm

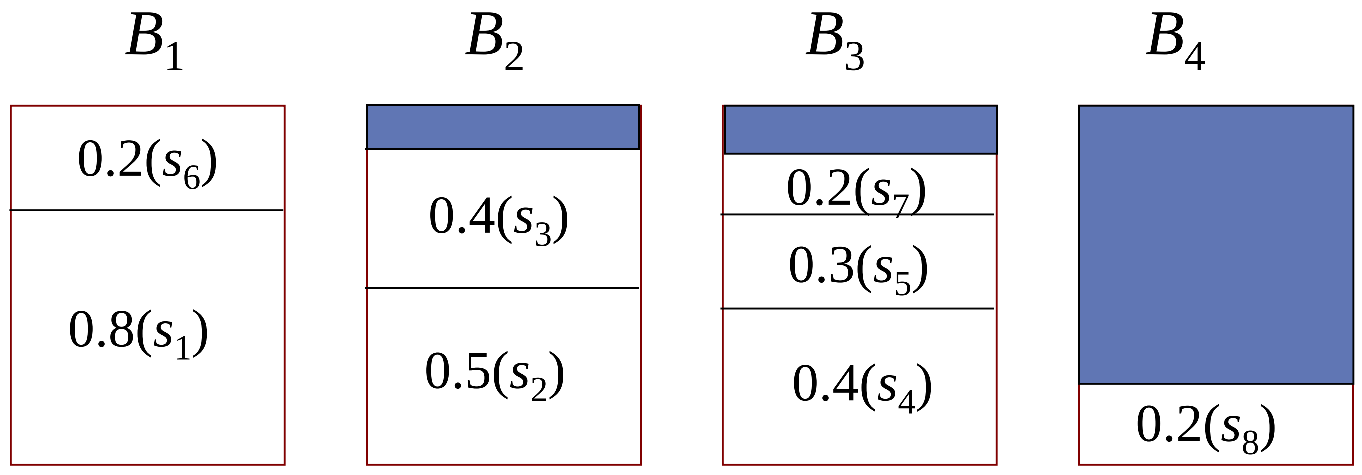
- **An approximation algorithm A for a problem**
  - Polynomial-time algorithm that, when given input I, output an element of FS(I).
- **Quality of an approximation algorithm.**

$$r_{\mathrm{A}}(I) = \frac{val(I, \mathrm{A}(I))}{opt(I)} \text{ or } r_{\mathrm{A}}(I) = \frac{opt(I)}{val(I, \mathrm{A}(I))}$$

  - RA(m) = max $\{r_A(I) \mid$ I such that opt(I)=m$\}$
- **Bounded RA(m)**
  - For an approximation algorithm, we hope the value of RA(m) is bounded by small constants.

First Fit Decreasing - FFD

- The strategy: packing the largest as possible
- Example: S=(0.8, 0.5, 0.4, 0.4, 0.3, 0.2, 0.2, 0.2)



This is ***NOT*** an optimal solution!

The Procedure

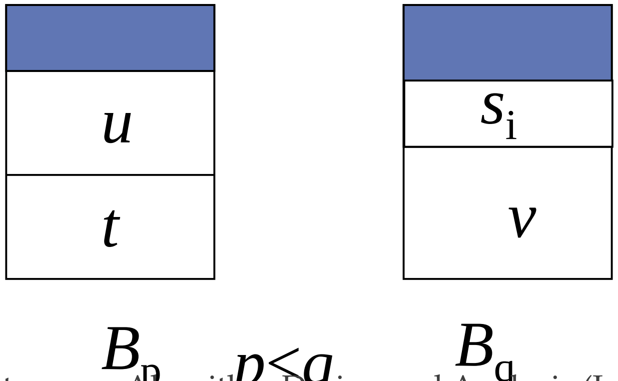
```
binpackFFD(S, n, bin) //bin is filled and output, object i is packed in bin[i]
    float[] used=new float[n+1]; //used[j] is the occupied space in bin j
    int i,j;
    <initialize all used entries to 0.0>
    <sort S into nonincreasing order> // in S after sorted

    for (i=1; i≤n; i++)
        for (j=1; j≤n; j++)
            if (used[j]+S[i]≤1.0)
                bin[i]=j;
                used[j]+=S[i];
                break;
```

Small Objects in Extra Bins

- **Problem formulation**
  - Let $S=\{s_1, s_2, \ldots, s_n\}$ be an input, **in nonincreasing order**
  - Let $opt(S)$ be the minimum number of bins for S.
- **All of the objects placed by FFD in the extra bins have size at most 1/3.**
- **Let i be the index of the first object placed by FFD in bin $opt(S)+1$.**
  - What we have to do for the proof is: $s_i \leq 1/3$.

What about a s_i Larger than 1/3?

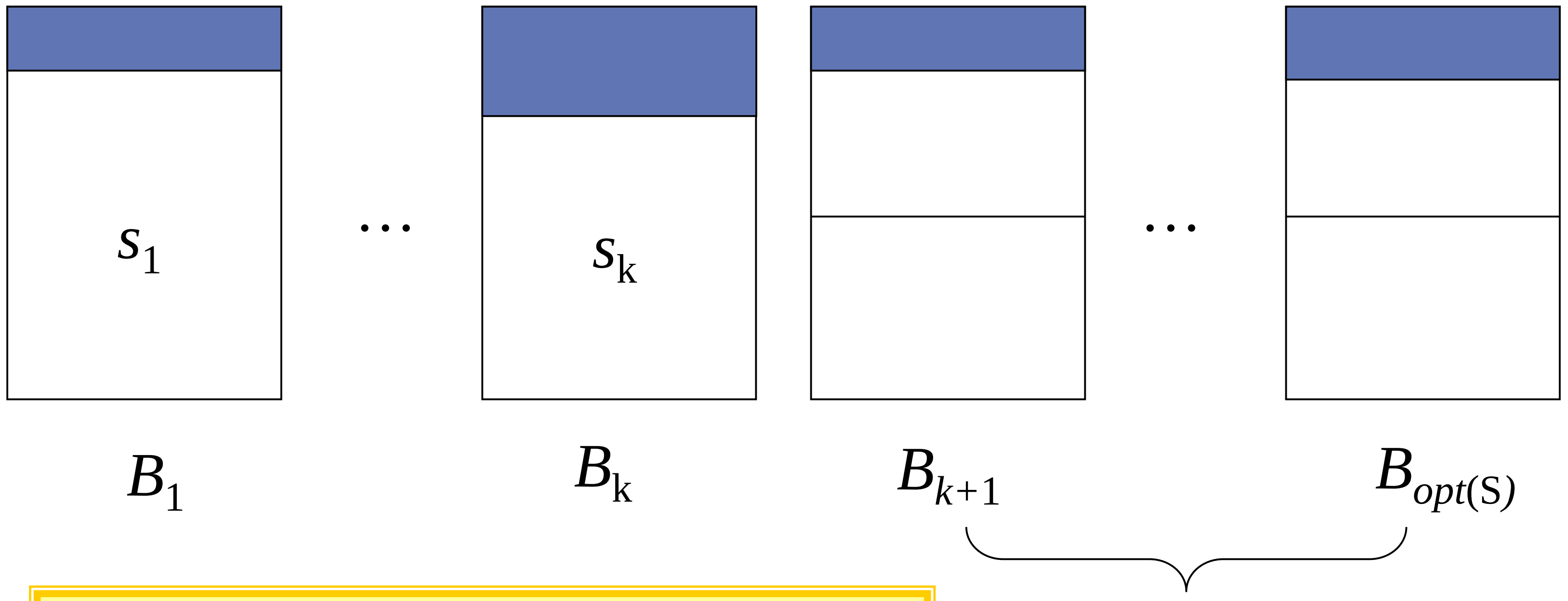
- **[S is sorted] The $s_1, s_2, \ldots, s_{i-1}$ are all larger than 1/3.**
- **So, bin B_j for $j=1,\ldots,opt(S)$ contain at most 2 objects each.**
- **Then, for some $k \geq 0$, the first k bins contain one object each and the remaining $opt(S)-k$ bins contain two each.**
  - Proof: no situation (that is, some bin containing 2 objects has a smaller index than some bin containing only one object) as the following is possible



Then: we must have:

$t>v$, $u>s_i$, so $v+s_i<1$, no extra bin is needed!

Considering S_i



containing 2 objects each

So, in any optimal solution, there will be *k* bins that do not contain any of the objects *k*+1,…, *i*.

Contradiction at Last!

- Any optimal solution use only *opt*(*S*) bins.
- However, there are *k* bins that do not contain any of the objects *k*+1, …, *i*-1, *i*. *k*+1,…, *i*-1 must occupy *opt*(*S*)-*k* bins, with each bin containing 2.
- Since all objects down through to s_i are larger than 1/3, s_i can not fit in any of the *opt*(*S*)-*k* bins.
- So, extra bin needed, and contradiction.

Objected in Extra Bins Bounded

- **For any input $S=\{s_1, s_2,\ldots,s_n\}$, the number of objects placed by FFD in extra bins is at most $opt(S)$-1.**

Since all the objects fit in $opt(S)$, $\sum_{i=1}^{n} s_i \le opt(S)$.

Assuming that FFD puts $opt(S)$ objects in extra bins, and their sizes are: $t_1, t_2, \ldots, t_{opt(S)}$.

Let b_j be the final contents of bin B_j for $1 \le j \le opt(S)$.

Note $b_j + t_j > 1$, otherwise t_j should be put in B_j. So:

$$\sum_{i=1}^{n} s_i \ge \sum_{j=1}^{opt(S)} b_j + \sum_{j=1}^{opt(S)} t_j = \sum_{j=1}^{opt(S)} (b_i + t_i)$$

$> opt(S)$; Contradiction!

A Good Approximation

- **Using FFD, the number of bins used is at most about 1/3 more than optimal value.**

$$R_{FFD}(m) \le \frac{4}{3} + \frac{1}{3m}$$

FFD puts at most *m*-1 objects in extra bins, and the size of the *m*-1 object are at most 1/3 each, so, FFD uses at most $\lceil (m\text{-}1)/3 \rceil$ extra bins.

$$r_{FFD}(S) \le \frac{m + \left\lceil \frac{m-1}{3} \right\rceil}{m} \le 1 + \frac{m+1}{3m} \le \frac{4}{3} + \frac{1}{3m}$$

Average Performance is Much Better

- **Empirical Studies on large inputs.**
  - The number of extra bins are estimated by the amount of empty space in the packings produced by the algorithm.
  - It has been shown that for n objects with sizes uniformly distributed between zero and one, the expected amount of empty space in packings by FFD is approximately $0.3\sqrt{n}$.

Randomized Algorithm

https://book.douban.com/subject/1856070/

- **Mote Carlo**
  - Always finish in time
  - The answer may be incorrect
- **Las Vegas**
  - Always return the correct answer
  - The running time varies a lot

Online Algorithm

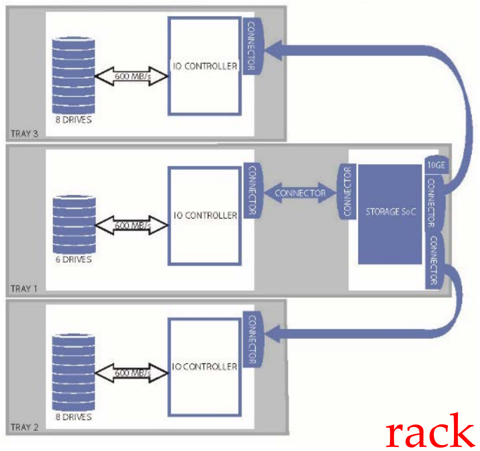
- **The main difference**
  - Offline algorithm: you can obtain all your input in advance
  - Online algorithm: you must cope with unpredictable inputs
- **How to analyze an online algorithm**
  - Competitive analysis: the performance of an online algorithm is compared to that of an optimal offline algorithm

Distributed Data

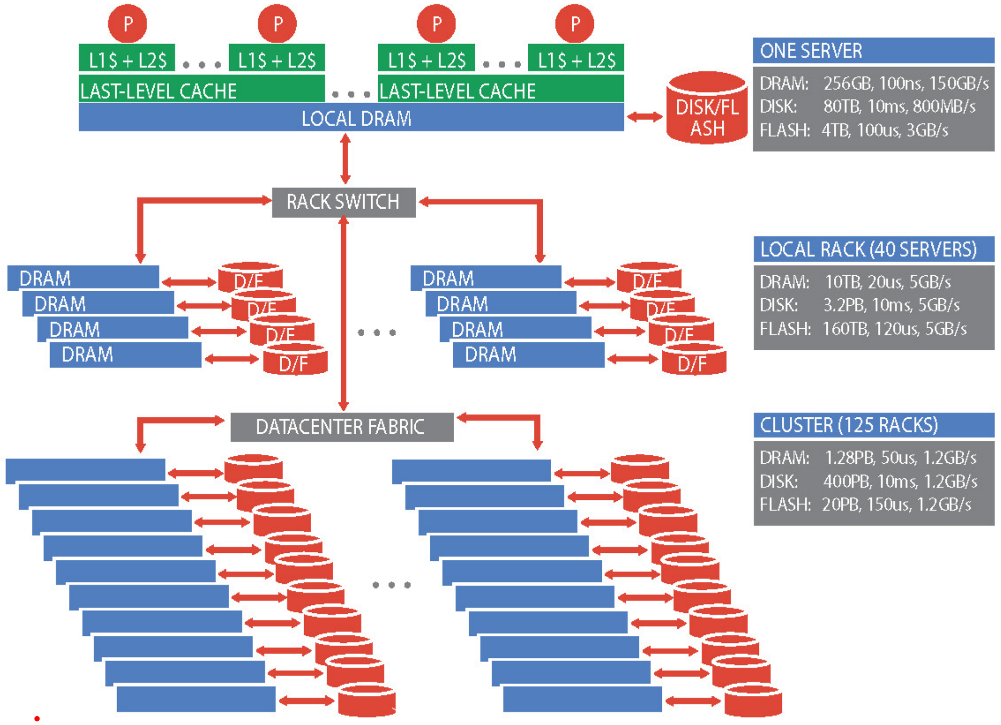
- External memory model

https://book.douban.com/subject/4665320/

tray



rack

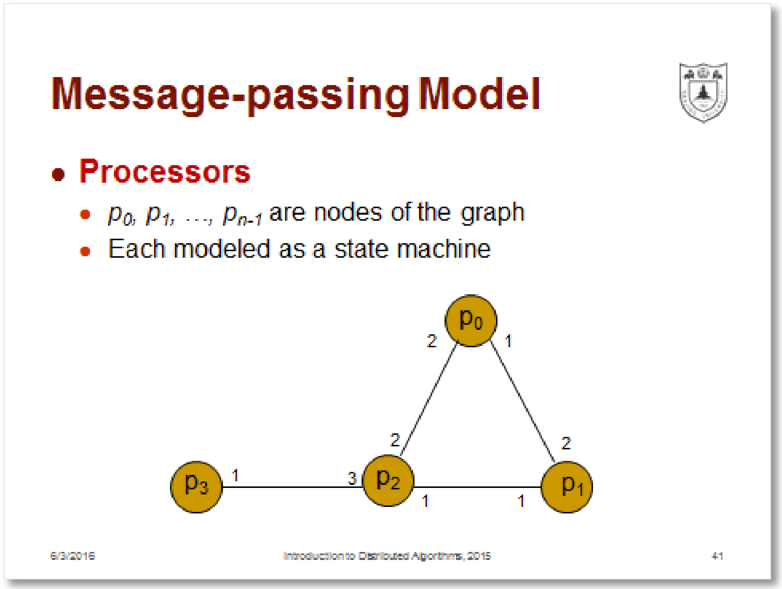


fabric

Distributed Algorithms

- Model of distributed computation

https://book.douban.com/subject/4190386/

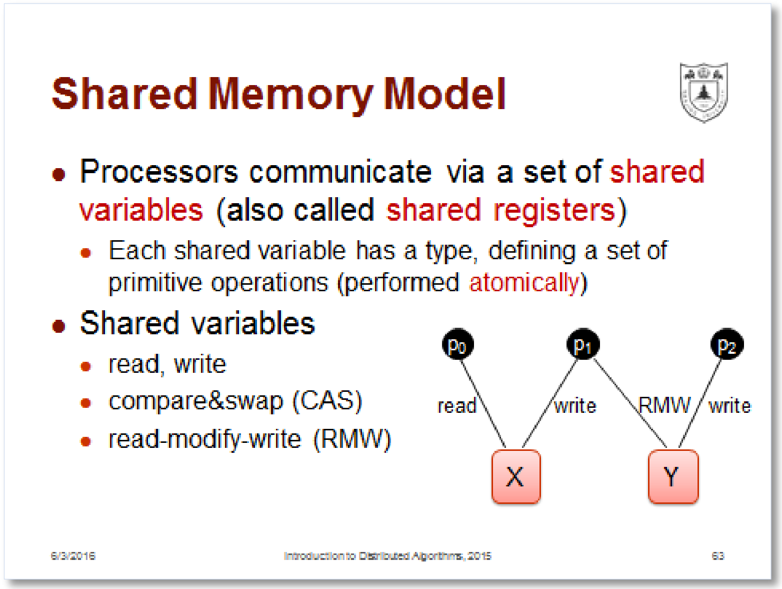


Message-passing Model

- **Processors**
  - p_0, p_1, …, p_{n-1} are nodes of the graph
  - Each modeled as a state machine

p_0
p_1
p_2
p_3
2
1
2
2
1
3
1
1

6/3/2016 Introduction to Distributed Algorithms, 2015 41



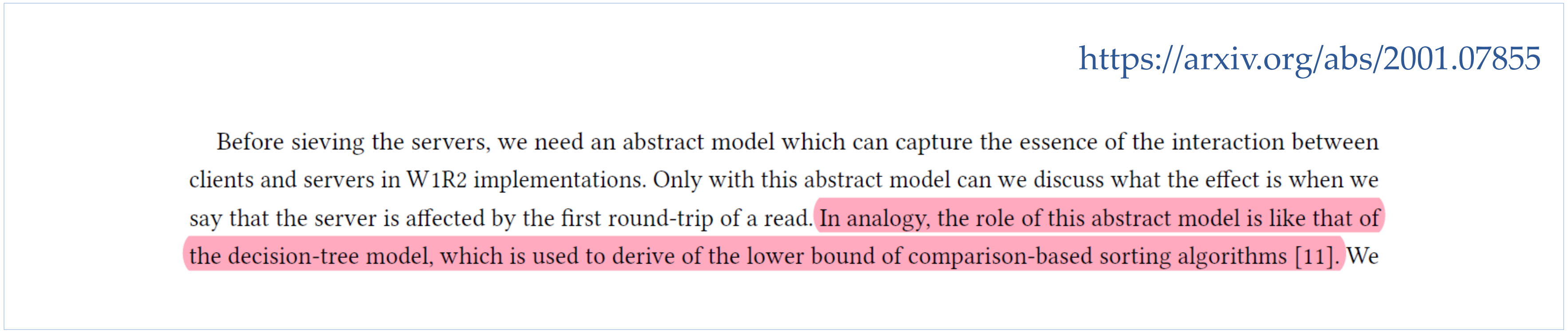
Shared Memory Model

- Processors communicate via a set of shared variables (also called shared registers)
  - Each shared variable has a type, defining a set of primitive operations (performed atomically)
- Shared variables
  - read, write
  - compare&swap (CAS)
  - read-modify-write (RMW)

p_0
p_1
p_2
read
write
RMW
write
X
Y

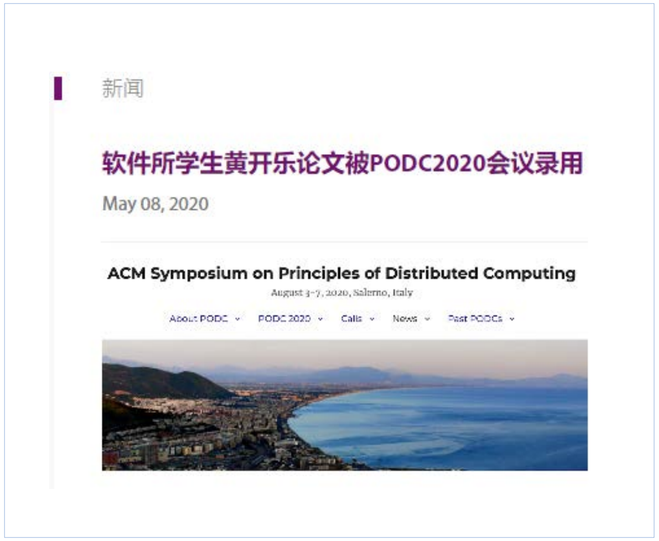
6/3/2016 Introduction to Distributed Algorithms, 2015 63

Distributed Algorithms



https://arxiv.org/abs/2001.07855

Before sieving the servers, we need an abstract model which can capture the essence of the interaction between clients and servers in W1R2 implementations. Only with this abstract model can we discuss what the effect is when we say that the server is affected by the first round-trip of a read. In analogy, the role of this abstract model is like that of the decision-tree model, which is used to derive of the lower bound of comparison-based sorting algorithms [11]. We



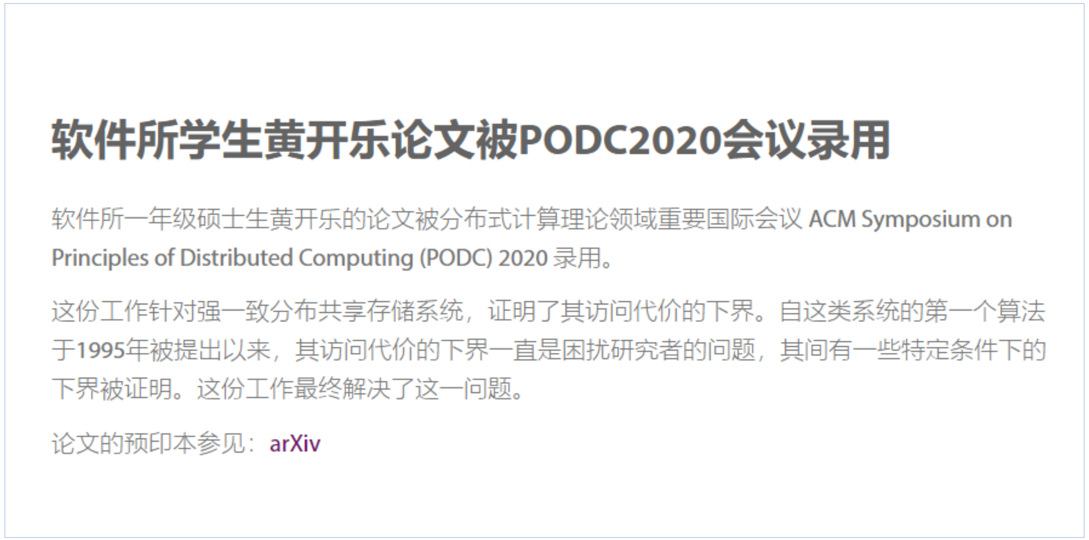
新闻

软件所学生黄开乐论文被PODC2020会议录用

May 08, 2020

ACM Symposium on Principles of Distributed Computing

August 3-7, 2020, Salerno, Italy



软件所学生黄开乐论文被PODC2020会议录用

软件所一年级硕士生黄开乐的论文被分布式计算理论领域重要国际会议 ACM Symposium on Principles of Distributed Computing (PODC) 2020 录用。

这份工作针对强一致分布共享存储系统，证明了其访问代价的下界。自这类系统的第一个算法于1995年被提出以来，其访问代价的下界一直是困扰研究者的问题，其间有一些特定条件下的下界被证明。这份工作最终解决了这一问题。

论文的预印本参见：arXiv

Thank you!

Q & A

Yu Huang
http://cs.nju.edu.cn/yuhuang