Algorithm Design XIII

NP Problem I

Guoqiang Li
School of Software

Shanghai Jiao Tong University

Efficient Problems, Difficult Problems

Efficient Algorithms

We have developed algorithms for

Efficient Algorithms

We have developed algorithms for

- Finding shortest paths in graphs,
- Minimum spanning trees in graphs,
- Matchings in bipartite graphs,
- Maximum increasing subsequences,
- Maximum flows in networks,
- ......

Efficient Algorithms

We have developed algorithms for

- Finding shortest paths in graphs,
- Minimum spanning trees in graphs,
- Matchings in bipartite graphs,
- Maximum increasing subsequences,
- Maximum flows in networks,
- ......

All these algorithms are efficient, since their time requirement grows as a polynomial function (such as n, n^2, or n^3) of the size of the input.

注意理论与实际的区别：理论上认为只要是多项式时间就是高效的算法。

Exponential Search Space

In these problems we are searching for a solution (path, tree, matching) from among an exponential population of possibilities.

Exponential Search Space

In these problems we are searching for a solution (path, tree, matching) from among an exponential population of possibilities.

All these problems could in principle be solved in exponential time by checking through all candidate solutions, one by one.

Exponential Search Space

In these problems we are searching for a solution (path, tree, matching) from among an exponential population of possibilities.

All these problems could in principle be solved in exponential time by checking through all candidate solutions, one by one.

An algorithm with running time 2^n, or worse, is useless in practice.

Exponential Search Space

In these problems we are searching for a solution (path, tree, matching) from among an exponential population of possibilities.

All these problems could in principle be solved in exponential time by checking through all candidate solutions, one by one.

An algorithm with running time 2^n, or worse, is useless in practice.

The efficient algorithms is to find clever ways to bypass exhaustive search, using clues from the input to narrow down the search space.

Exponential Search Space

In these problems we are searching for a solution (path, tree, matching) from among an exponential population of possibilities.

All these problems could in principle be solved in exponential time by checking through all candidate solutions, one by one.

An algorithm with running time 2^n, or worse, is useless in practice.

The efficient algorithms is to find clever ways to bypass exhaustive search, using clues from the input to narrow down the search space.

Are there "search problems" in which seeking a solution among an exponential chaos, and the fastest algorithms for them are exponential?

Minimum Spanning Trees

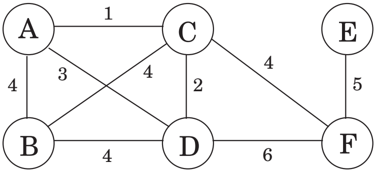
(最小生成树)

Build a Network

Suppose you are asked to network a collection of computers by linking selected pairs of them.

This translates into a graph problem in which

- nodes are computers,
- undirected edges are potential links, each with a maintenance cost.



A General Kruskal's Algorithm

$X = \{\ \}$;
repeat until $|X| = |V| - 1$;
    pick a set $S \subset V$ for which X has no edges between S and $V - S$;
    let $e \in E$ be the minimum-weight edge between S and $V - S$;
    $X = X \cup \{e\}$;

A Little Change of the MST

Shanghai Jiao Tong University

WHAT IF THE TREE IS NOT ALLOWED TO BRANCH?

MST如果不允许分支<==>旅行商问题(即移除从终点j到起点1的那条边之后的TSP路径)

SATISFIABILITY PROBLEM

(可满足性问题)

Satisfiability

The instances of SATISFIABILITY or SAT:

$$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})$$

a Boolean formula in conjunctive normal form (CNF).

Satisfiability

The instances of SATISFIABILITY or SAT:

$$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})$$

a Boolean formula in conjunctive normal form (CNF).

It is a collection of clauses (the parentheses),

- each consisting of the disjunction of several literals;
- a literal is either a Boolean variable or the negation of one.

The instances of SATISFIABILITY or SAT:

$$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})$$

a Boolean formula in conjunctive normal form (CNF).

It is a collection of clauses (the parentheses),

- each consisting of the disjunction of several literals;
- a literal is either a Boolean variable or the negation of one.

A satisfying truth assignment is an assignment of `false` or `true` to each variable so that every clause contains a literal of `true`.

The instances of SATISFIABILITY or SAT:

$$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})$$

a Boolean formula in conjunctive normal form (CNF). （合取范式）

It is a collection of clauses (the parentheses), （clause:子句）

- each consisting of the disjunction of several literals;
- a literal is either a Boolean variable or the negation of one.

A satisfying truth assignment is an assignment of `false` or `true` to each variable so that every clause contains a literal of `true`.

Given a Boolean formula in conjunctive normal form, either find a satisfying truth assignment or else report that none exists.

2-SAT

Given a set of clauses, where each clause is the disjunction of two literals, looking for an assignment so that all clauses are satisfied.

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_1 \vee \overline{x}_2) \wedge (x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_4)$$

2-SAT

Given a set of clauses, where each clause is the disjunction of two literals, looking for an assignment so that all clauses are satisfied.

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_1 \vee \overline{x}_2) \wedge (x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_4)$$

Given an instance I of 2-SAT with n variables and m clauses, construct a directed graph $G_I = (V, E)$ as follows.

2-SAT

Given a set of clauses, where each clause is the disjunction of two literals, looking for an assignment so that all clauses are satisfied.

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_1 \vee \overline{x}_2) \wedge (x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_4)$$

Given an instance I of 2-SAT with n variables and m clauses, construct a directed graph $G_I = (V, E)$ as follows.

- G_I has $2n$ nodes, one for each variable and its negation.
- G_I has $2m$ edges: for each clause $(\alpha \vee \beta)$ of I, G_I has an edge from the negation of α to β, and one from the negation of β to α.

```
A U B <==> (!A->B) intersection (!B->A)
```

2-SAT

Show that if G_I has a strongly connected component containing both x and $\overline{x}$ for some variable x, then I has no satisfying assignment.

2-SAT

Show that if G_I has a strongly connected component containing both x and $\overline{x}$ for some variable x, then I has no satisfying assignment.

If none of G_I's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable.

2-SAT

Show that if G_I has a strongly connected component containing both x and $\overline{x}$ for some variable x, then I has no satisfying assignment.

If none of G_I's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable.

Conclude that there is a linear-time algorithm for solving 2-SAT.

SAT==>有向图==>强联通分量==>检查每一个连通分量是否冲突==>O(n+m)
recall:homework

A Little Change of the 2-SAT

3-SAT, SAT?

3-SAT没有多项式时间的解法!

Search Problems

The instances of SATISFIABILITY or SAT:

$$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})$$

a Boolean formula in conjunctive normal form (CNF).

It is a collection of clauses (the parentheses),

- each consisting of the disjunction of several literals;
- a literal is either a Boolean variable or the negation of one.

A satisfying truth assignment is an assignment of `false` or `true` to each variable so that every clause contains a literal of `true`.

Given a Boolean formula in conjunctive normal form, either find a satisfying truth assignment or else report that none exists.

Search Problems

SAT is a typical search problem.

Search Problems

SAT is a typical search problem.

We are given an `instance` I

Search Problems

SAT is a typical search problem.

We are given an `instance` I

- Some input data specifying the problem
- A Boolean formula in conjunctive normal form

Search Problems

SAT is a typical search problem.

We are given an `instance` I

- Some input data specifying the problem
- A Boolean formula in conjunctive normal form

we are asked to find a `solution` S

Search Problems

SAT is a typical search problem.

We are given an `instance` I

- Some input data specifying the problem
- A Boolean formula in conjunctive normal form

we are asked to find a `solution` S

- An object that meets a particular specification
- An assignment that satisfies each clause

Search Problems

SAT is a typical search problem.

We are given an `instance` I

- Some input data specifying the problem
- A Boolean formula in conjunctive normal form

we are asked to find a `solution` S

- An object that meets a particular specification
- An assignment that satisfies each clause

If no such solution exists, we must say so.

Search Problems

A search problem must have the property that any proposed solution S to an instance I can be quickly checked for correctness.

Search Problems

SHANGHAI JIAO TONG UNIVERSITY

A search problem must have the property that any proposed solution S to an instance I can be quickly checked for correctness.

S must be concise, with length polynomially bounded by that of I.

Search Problems

A search problem must have the property that any proposed solution S to an instance I can be quickly checked for correctness.

S must be concise, with length polynomially bounded by that of I.

- This is true for SAT, where S is an assignment to the variables.

Search Problems

A search problem must have the property that any proposed solution S to an instance I can be quickly checked for correctness.

S must be concise, with length polynomially bounded by that of I.

- This is true for SAT, where S is an assignment to the variables.

There is a polynomial-time algorithm that takes as input I and S and decides whether or not S is a solution of I.

Search Problems

SHANGHAI JIAO TONG UNIVERSITY

一个search problem可能是很难解决的，但是一定能够是很好验证的。

A search problem must have the property that any proposed solution S to an instance I can be quickly checked for correctness.

S must be concise, with length polynomially bounded by that of I.

- This is true for SAT, where S is an assignment to the variables.

There is a polynomial-time algorithm that takes as input I and S and decides whether or not S is a solution of I.

- For SAT, it is easy to check whether the assignment specified by S satisfies every clause in I.

Search Problems

A search problem is specified by an algorithm C that takes two inputs, an instance I and a proposed solution S, and runs in time polynomial in $|I|$.

Search Problems

A search problem is specified by an algorithm C that takes two inputs, an instance I and a proposed solution S, and runs in time polynomial in $|I|$.

We say S is a solution to I if and only if $C(I, S) = \texttt{true}$.

Satisfiability Revisit

Researchers over the past 80 years have tried to find efficient ways to solve the SAT, but without success.

Satisfiability Revisit

Researchers over the past 80 years have tried to find efficient ways to solve the SAT, but without success.

The fastest algorithms we have are still exponential on their worst-case inputs.

Satisfiability Revisit

Researchers over the past 80 years have tried to find efficient ways to solve the SAT, but without success.

The fastest algorithms we have are still exponential on their worst-case inputs.

There are two natural variants of SAT with good algorithms.

Satisfiability Revisit

Researchers over the past 80 years have tried to find efficient ways to solve the SAT, but without success.

The fastest algorithms we have are still exponential on their worst-case inputs.

There are two natural variants of SAT with good algorithms.

- 2-SAT can be solved in linear time.
- All clauses contain at most one positive literal, say Horn formula（霍尔范式）, can be found by the greedy algorithm.

Traveling Salesman Problem

(TSP Revisiting)

The Traveling Salesman Problem

The Traveling Salesman Problem

In the traveling salesman problem(TSP) we are given n vertices and all $n(n-1)/2$ distances between them, and a budget b.

The Traveling Salesman Problem

In the traveling salesman problem(TSP) we are given n vertices and all $n(n-1)/2$ distances between them, and a budget b.

To find a cycle that passes through every vertex exactly once, of total cost b or less - or to report that no such cycle.

The Traveling Salesman Problem

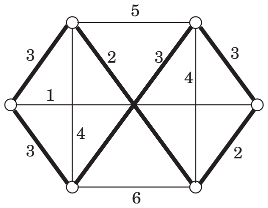
In the traveling salesman problem(TSP) we are given n vertices and all $n(n-1)/2$ distances between them, and a budget b.

(TSP问题中输入是一个完全图)

To find a cycle that passes through every vertex exactly once, of total cost b or less - or to report that no such cycle.

A permutation $\tau(1), \ldots, \tau(n)$ of the vertices such that when they are toured in this order, the total distance covered is at most b:

$$d_{\tau(1),\tau(2)} + d_{\tau(2),\tau(3)} + \ldots + d_{\tau(n),\tau(1)} \leq b$$



The Traveling Salesman Problem

The Traveling Salesman Problem

We have defined the TSP as a search problem: given an instance, find a tour within the budget (or report that none exists).

The Traveling Salesman Problem

SHANGHAI JIAO TONG UNIVERSITY

We have defined the TSP as a search problem: given an instance, find a tour within the budget (or report that none exists).

But why are we expressing the TSP in this way, when in reality it is an optimization problem, in which the shortest possible tour is sought?

在动态规划部分讲过的TSP问题是一个**优化问题**，因为他是只知道图数据要去找最优路径；
而这部分提到的TSP问题是一个搜索问题，因为他是已经提供了一个instance，即上限为b的一个TSP问题，而要去找一个解去看满不满足这个约束。
优化问题不好验证，更加适用于近似算法。
搜索问题容易验证，更加适用于理论分析。

Search VS. Optimization

Turning an optimization problem into a search problem does not change its difficulty, because the two versions reduce to one another.

Search VS. Optimization

Turning an optimization problem into a search problem does not change its difficulty, because the two versions reduce to one another.

Any algorithm that solves the optimization also solves the search problem:

Search VS. Optimization

Turning an optimization problem into a search problem does not change its difficulty, because the two versions reduce to one another.

Any algorithm that solves the optimization also solves the search problem:

- find the optimum tour and if it is within budget, return it; if not, there is no solution.

Search VS. Optimization

Turning an optimization problem into a search problem does not change its difficulty, because the two versions reduce to one another.

Any algorithm that solves the optimization also solves the search problem:

- find the optimum tour and if it is within budget, return it; if not, there is no solution.

Conversely, an algorithm for the search problem can also be used to solve the optimization problem:

Search VS. Optimization

Turning an optimization problem into a search problem does not change its difficulty, because the two versions reduce to one another.

Any algorithm that solves the optimization also solves the search problem:

- find the optimum tour and if it is within budget, return it; if not, there is no solution.

Conversely, an algorithm for the search problem can also be used to solve the optimization problem:

- First suppose that we knew the cost of the optimum tour; then we could find this tour by calling the algorithm for the search problem, using the optimum cost as the budget.
- We can find the optimum cost by binary search.

优化问题转化到搜索问题：对于优化问题得到的一个结果，直接检查是否满足搜索问题的budget要求，若满足则搜索问题可解，反之则不能。
搜索问题转化到优化问题：对于优化问题的理论上界，可以确定最终的优化问题的答案应该就位于0与这个上界之间。设为A。对于A进行一次搜索问题算法，若可解，则对于二分之后的左半部分的上界进行递归的搜索算法，若不可解，则对范围的右半部分的下界进行递归的搜索算法，通过这种方式找到优化问题的上确界，就是优化问题的答案。因而有一个搜索算法，就会有一个对应的优化算法。

Search Instead of Optimization

Isn't any optimization problem also a search problem in the sense that we are searching for a solution that has the property of being optimal?

Search Instead of Optimization

Isn't any optimization problem also a search problem in the sense that we are searching for a solution that has the property of being optimal?

The solution to a search problem should be easy to recognize, or as we put it earlier, polynomial-time checkable.

Search Instead of Optimization

Isn't any optimization problem also a search problem in the sense that we are searching for a solution that has the property of being optimal?

The solution to a search problem should be easy to recognize, or as we put it earlier, polynomial-time checkable.

Given a potential solution to the TSP, it is easy to check the properties "is a tour" (just check that each vertex is visited exactly once)

Search Instead of Optimization

Isn't any optimization problem also a search problem in the sense that we are searching for a solution that has the property of being optimal?

The solution to a search problem should be easy to recognize, or as we put it earlier, polynomial-time checkable.

Given a potential solution to the TSP, it is easy to check the properties "is a tour" (just check that each vertex is visited exactly once) and "has total length $\leq b$."

Search Instead of Optimization

SHANGHAI JIAO TONG UNIVERSITY

Isn't any optimization problem also a search problem in the sense that we are searching for a solution that has the property of being optimal?

The solution to a search problem should be easy to recognize, or as we put it earlier, polynomial-time checkable.

Given a potential solution to the TSP, it is easy to check the properties "is a tour" (just check that each vertex is visited exactly once) and "has total length $\leq b$."

But how could one check the property "is optimal"?

TSP Revisit

There are no known polynomial-time algorithms for the TSP, despite much effort by researchers over nearly a century.

TSP Revisit

There are no known polynomial-time algorithms for the TSP, despite much effort by researchers over nearly a century.

There exists a faster, yet still exponential, dynamic programming algorithm.

TSP Revisit

There are no known polynomial-time algorithms for the TSP, despite much effort by researchers over nearly a century.

There exists a faster, yet still exponential, dynamic programming algorithm.

The MINIMUM SPANNING TREE (MST) problem, for which we do have efficient algorithms, provides a stark contrast here.

TSP Revisit

There are no known polynomial-time algorithms for the TSP, despite much effort by researchers over nearly a century.

There exists a faster, yet still exponential, dynamic programming algorithm.

The Minimum spanning tree (MST) problem, for which we do have efficient algorithms, provides a stark contrast here.

The TSP can be thought of as a tough cousin of the MST problem, in which the tree is not allowed to branch and is therefore a path.

TSP Revisit

There are no known polynomial-time algorithms for the TSP, despite much effort by researchers over nearly a century.

There exists a faster, yet still exponential, dynamic programming algorithm.

The Minimum spanning tree (MST) problem, for which we do have efficient algorithms, provides a stark contrast here.

The TSP can be thought of as a tough cousin of the MST problem, in which the tree is not allowed to branch and is therefore a path.

This extra restriction on the structure of the tree results in a much harder problem.

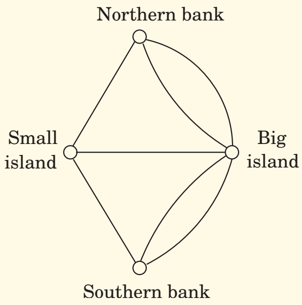
(不允许有分支)

Euler and Rudrata

Euler Path

Euler path:

Given a graph, find a path that contains each edge exactly once



Euler Path

Euler Path

SHANGHAI JIAO TONG UNIVERSITY

The answer is yes if and only if

Euler Path

The answer is yes if and only if

1. the graph is connected and
2. every vertex, with the possible exception of two vertices (the start and final vertices of the walk), has even degree.

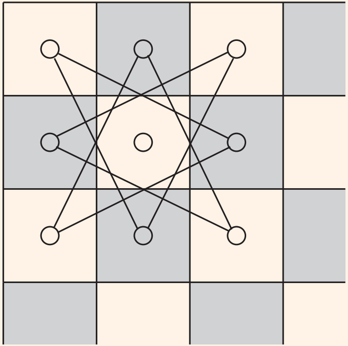
Euler Path

The answer is yes if and only if

1. the graph is connected and
2. every vertex, with the possible exception of two vertices (the start and final vertices of the walk), has even degree.

There is a polynomial time algorithm for EULER PATH.

Rudrata Cycle



Rudrata Cycle

Rudrata Cycle:

Rudrata Cycle

RUDRATA CYCLE:

Given a graph, find a cycle that visits each vertex exactly once.

Rudrata Cycle

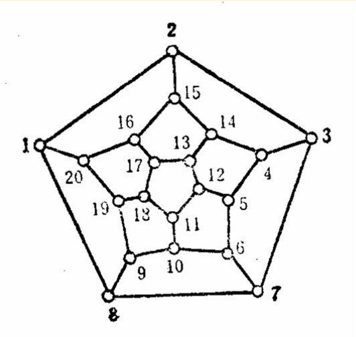
Hamilton问题与TSP问题的区别：
前者不要求完全图，后者要求输入应该是一个完全图。

SHANGHAI JIAO TONG UNIVERSITY

RUDRATA CYCLE:

Given a graph, find a cycle that visits each vertex exactly once.

In the literature this problem is known as the Hamilton cycle problem.



Cuts and Bisections

Minimum Cut

Minimum Cut

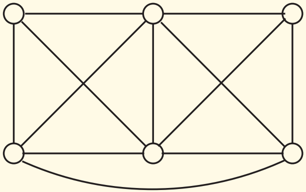
A cut is a set of edges whose removal leaves a graph disconnected.

Minimum Cut

A cut is a set of edges whose removal leaves a graph disconnected.

MINIMUM CUT: given a graph and a budget b, find a cut with at most b edges.

(a search problem)



在这个图中，答案应该是3.

Minimum Cut

Minimum Cut

This problem can be solved in polynomial time by $n-1$ max-flow computations:

Minimum Cut

This problem can be solved in polynomial time by $n-1$ max-flow computations:

- give each edge a capacity of 1,
- and find the maximum flow between some fixed node and every single other node.

Minimum Cut

This problem can be solved in polynomial time by $n-1$ max-flow computations:

- give each edge a capacity of 1,
- and find the maximum flow between some fixed node and every single other node.

The smallest such flow will correspond (via the max-flow min-cut theorem) to the smallest cut.

算法逻辑：选定一个点为基准点作为src，对于其他每个点，分别作为sink与src进行一次max-flow算法；
即一共有n个点，则需要n-1次的max-flow算法，找到这n-1个结果中的最小值就是最终结果。

Balanced Cut

Balanced Cut

In many graphs, the smallest cut leaves just a singleton vertex on one side

Balanced Cut

In many graphs, the smallest cut leaves just a singleton vertex on one side - it consists of all edges adjacent to this vertex.

Balanced Cut

In many graphs, the smallest cut leaves just a singleton vertex on one side - it consists of all edges adjacent to this vertex.

Far more interesting are small cuts that partition the vertices of the graph into nearly equal-sized sets.

Balanced Cut

In many graphs, the smallest cut leaves just a singleton vertex on one side - it consists of all edges adjacent to this vertex.

Far more interesting are small cuts that partition the vertices of the graph into nearly equal-sized sets.

BALANCED CUT:

Balanced Cut

In many graphs, the smallest cut leaves just a singleton vertex on one side - it consists of all edges adjacent to this vertex.

Far more interesting are small cuts that partition the vertices of the graph into nearly equal-sized sets.

BALANCED CUT: Given a graph with n vertices and a budget b, partition the vertices into two sets S and T such that $|S|, |T| \geq n/3$ and such that there are at most b edges between S and T.

Integer Linear Programming

Linear Programming

In a LINEAR PROGRAMMING problem we are given a set of variables, and to assign real values to them so as to

Linear Programming

Shanghai Jiao Tong University

In a LINEAR PROGRAMMING problem we are given a set of variables, and to assign real values to them so as to

1. satisfy a set of linear equations and/or linear inequalities involving these variables, and
2. maximize or minimize a given linear objective function.

Linear Programming

$$
\begin{aligned}
\max x_1 + 6x_2 + 13x_3 \\
x_1 \leq 200 \\
x_2 \leq 300 \\
x_1 + x_2 + x_3 \leq 400 \\
x_2 + 3x_3 \leq 600 \\
x_1, x_2, x_3 \geq 0
\end{aligned}
$$

Integer Linear Programming

INTEGER LINEAR PROGRAMMING (ILP):

Integer Linear Programming

Shanghai Jiao Tong University

INTEGER LINEAR PROGRAMMING (ILP): We are given a set of linear inequalities $A\mathbf{x} \leq b$, where

INTEGER LINEAR PROGRAMMING (ILP): We are given a set of linear inequalities $A\mathbf{x} \leq b$, where

- A is an $m \times n$ matrix and
- b is an m-vector;
- an objective function specified by an n-vector c;
- a goal g.

Integer Linear Programming

SHANGHAI JIAO TONG UNIVERSITY

（整数线性规划）

INTEGER LINEAR PROGRAMMING (ILP): We are given a set of linear inequalities $A\mathbf{x} \leq b$, where

- A is an $m \times n$ matrix and
- b is an m-vector;
- an objective function specified by an n-vector c;
- a goal g.

We want to find a nonnegative integer n-vector x such that $A\mathbf{x} \leq b$ and $c \cdot \mathbf{x} \geq g$.

Integer Linear Programming

$$
\begin{aligned}
\max 2x_1 + 5x_2 \\
2x_1 - x_2 \leq 4 \\
x_1 + 2x_2 \leq 9 \\
-x_1 + x_2 \leq 3 \\
x_1, x_2 \geq 0
\end{aligned}
$$

$$
\begin{aligned}
2x_1 + 5x_2 \leq g \\
2x_1 - x_2 \leq 4 \\
x_1 + 2x_2 \leq 9 \\
-x_1 + x_2 \leq 3 \\
x_1, x_2 \geq 0
\end{aligned}
$$

Integer Linear Programming

$$\begin{aligned} \max 2x_1 + 5x_2 \\ 2x_1 - x_2 \leq 4 \\ x_1 + 2x_2 \leq 9 \\ -x_1 + x_2 \leq 3 \\ x_1, x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} 2x_1 + 5x_2 \leq g \\ 2x_1 - x_2 \leq 4 \\ x_1 + 2x_2 \leq 9 \\ -x_1 + x_2 \leq 3 \\ x_1, x_2 \geq 0 \end{aligned}$$

But there is a redundancy here:

- the last constraint $c \cdot \mathbf{x} \geq g$ is itself a linear inequality and
- can be absorbed into $A\mathbf{x} \leq b$.

对于一个整数线性规划问题，在其值域约束只能取到{0,1}的情况下，实际上就等价于一个SAT问题。但是一般无解。

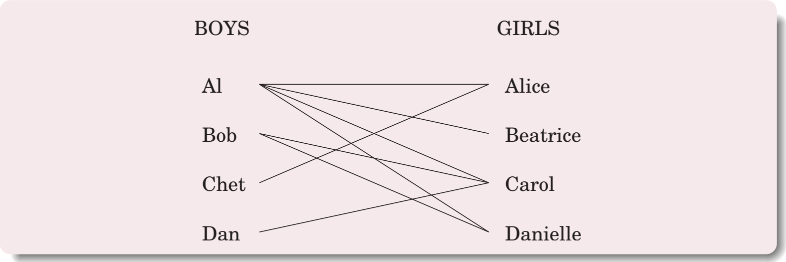
Integer Linear Programming

So, we define ILP to be following search problem:

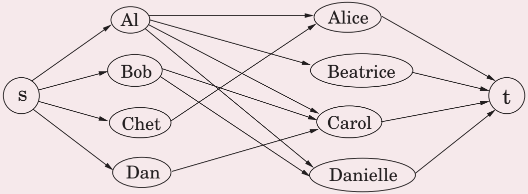
Given A and b, find a nonnegative integer vector $\mathbf{x}$ satisfying the inequalities $A\mathbf{x} \leq b$.

3D-MATCHING

Bipartite Matching



Bipartite Matching



Three-Dimensional Matching

3D MATCHING:

Three-Dimensional Matching

3D MATCHING: There are n boys, n girls, and n pets.

Three-Dimensional Matching

3D MATCHING: There are n boys, n girls, and n pets. The compatibilities are specified by a set of triples,

Three-Dimensional Matching

3D MATCHING: There are n boys, n girls, and n pets. The compatibilities are specified by a set of triples, each containing a boy, a girl, and a pet.

Three-Dimensional Matching

3D MATCHING: There are n boys, n girls, and n pets. The compatibilities are specified by a set of triples, each containing a boy, a girl, and a pet.

A triple (b, g, p) means that boy b, girl g, and pet p get along well together.

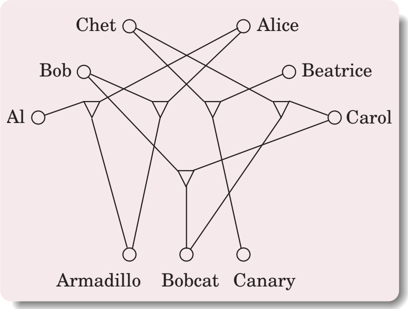
Three-Dimensional Matching

Shanghai Jiao Tong University

3D MATCHING: There are n boys, n girls, and n pets. The compatibilities are specified by a set of triples, each containing a boy, a girl, and a pet.

A triple (b, g, p) means that boy b, girl g, and pet p get along well together.

To find n disjoint triples and thereby create n harmonious households.



在右图中，由于每个人都只能在一个组中，
所以约束条件实际上也是一个SAT问题，但是
同样不一定可解。

INDEPENDENT SET, VERTEX COVER, AND CLIQUE

Independent Set, Vertex Cover, and Clique

INDEPENDENT SET: Given a graph and an integer g, find g vertices, no two of which have an edge between them.

INDEPENDENT SET: Given a graph and an integer g, find g vertices, no two of which have an edge between them.

VERTEX COVER: Given a graph and an integer b, find b vertices cover (touch) every edge.

Independent Set, Vertex Cover, and Clique

SHANGHAI JIAO TONG UNIVERSITY

独立集

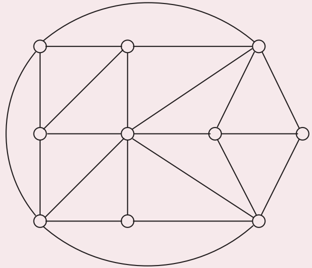
INDEPENDENT SET: Given a graph and an integer g, find g vertices, no two of which have an edge between them.

顶点覆盖

VERTEX COVER: Given a graph and an integer b, find b vertices cover (touch) every edge.

团：即两两之间有边的顶点集合

CLIQUE: Given a graph and an integer g, find g vertices such that all possible edges between them are present.



独立集：指的是将图中所有顶点划分成不可相交的子集，并且每个子集中包含的点之间都没有边。
顶点覆盖：指的是一个对于图中任意一条边都至少有一个顶点在某个点集中，这样的点集称之为**顶点覆盖**。
因为独立集中每个子集中点都没有边，所以边只能位于某两个点集之间，设一个图中的所有点构成集合V，将其划分为两个独立集S，V-S，则两个集合都是一个顶点覆盖。
所以**最大独立集问题等价于最小顶点覆盖问题**，即当独立集S取到最大值的时候，剩余的点集V-S就是一个最小顶点覆盖。

Longest Path

Longest Path

Longest Path

SHANGHAI JIAO TONG UNIVERSITY

LONGEST PATH: Given a graph G with nonnegative edge weights and two distinguished vertices s and t, along with a goal g.

Longest Path

LONGEST PATH: Given a graph G with nonnegative edge weights and two distinguished vertices s and t, along with a goal g.

To find a path from s to t with total weight at least g.

Longest Path

LONGEST PATH: Given a graph G with nonnegative edge weights and two distinguished vertices s and t, along with a goal g.

To find a path from s to t with total weight at least g.

To avoid trivial solutions we require that the path be simple, containing no repeated vertices.

Knapsack

Knapsack

Knapsack

Shanghai Jiao Tong University

KNAPSACK: Given integer weights $w_1, \ldots, w_n$ and integer values $v_1, \ldots, v_n$ for n items.

Knapsack

KNAPSACK: Given integer weights $w_1, \ldots, w_n$ and integer values $v_1, \ldots, v_n$ for n items. We are also given a weight capacity W and a goal g.

Knapsack

KNAPSACK: Given integer weights $w_1, \ldots, w_n$ and integer values $v_1, \ldots, v_n$ for n items. We are also given a weight capacity W and a goal g.

Seek a set of items whose total weight is at most W and whose total value is at least g.

Knapsack

KNAPSACK: Given integer weights $w_1, \ldots, w_n$ and integer values $v_1, \ldots, v_n$ for n items. We are also given a weight capacity W and a goal g.

Seek a set of items whose total weight is at most W and whose total value is at least g.

The problem is solvable in time $O(nW)$ by dynamic programming.

Knapsack

Is there a polynomial algorithm for KNAPSACK? Nobody knows of one.

Knapsack

Is there a polynomial algorithm for KNAPSACK? Nobody knows of one.

A variant of the KNAPSACK problem is that the unary integers.

Knapsack

Is there a polynomial algorithm for KNAPSACK? Nobody knows of one.

A variant of the KNAPSACK problem is that the unary integers.

- by writing $IIIIIIIIIIII$ for 12.
- It defines a legitimate problem, which we could call UNARY KNAPSACK.
- It has a polynomial algorithm.

Knapsack

Is there a polynomial algorithm for KNAPSACK? Nobody knows of one.

A variant of the KNAPSACK problem is that the unary integers.

- by writing $IIIIIIIIIIII$ for 12.
- It defines a legitimate problem, which we could call UNARY KNAPSACK.
- It has a polynomial algorithm.

A different variation:

- Suppose now that each item's value is equal to its weight, the goal g is the same as the capacity W.
- This special case is tantamount to finding a subset of a given set of integers that adds up to exactly W.

Knapsack

SHANGHAI JIAO TONG UNIVERSITY

Is there a polynomial algorithm for KNAPSACK? Nobody knows of one.

A variant of the KNAPSACK problem is that the unary integers.

- by writing $IIIIIIIIIIII$ for 12.
- It defines a legitimate problem, which we could call UNARY KNAPSACK.
- It has a polynomial algorithm.

A different variation:

- Suppose now that each item's value is equal to its weight, the goal g is the same as the capacity W.
- This special case is tantamount to finding a subset of a given set of integers that adds up to exactly W.
- Q: Could it be polynomial?

Subset Sum

Subset Sum

SUBSET SUM: Find a subset of a given set of integers that adds up to exactly W.