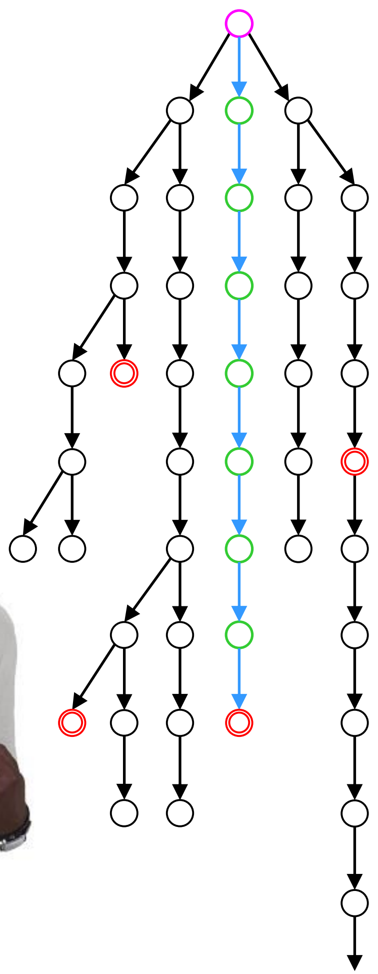
NP Completeness

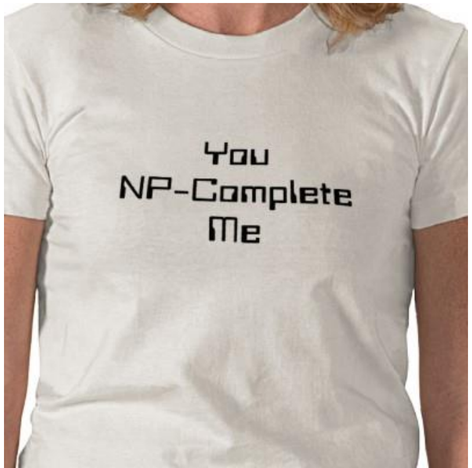
- Tractability
- Polynomial time
- Computation vs. verification
- Power of non-determinism
- Encodings
- Transformations & reducibilities
- P vs. NP
- “Completeness”

Stephen Cook

Leonid Levin

Richard Karp





NP Completeness Benefits

1. Saves time & effort of trying to solve intractable problems efficiently;
2. Saves money by not separately working to efficiently solve different problems;
3. Helps systematically build on & leverage the work (or lack of progress) of others;
4. Transformations can be used to solve new problems by reducing them to known ones;
5. Illuminates the structure & complexity of seemingly unrelated problems;

NP Completeness Benefits

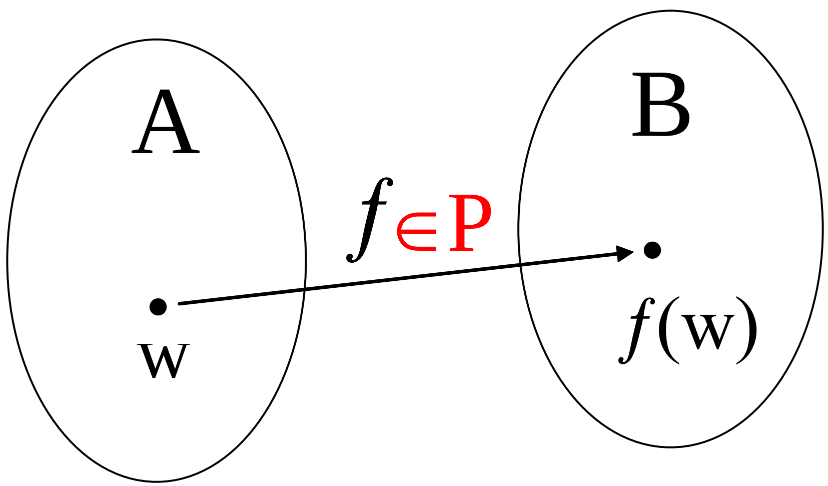
6. Informs as to when we should use approximate solutions vs. exact ones;
7. Helps understand the ubiquitous concept of *parallelism* (via non-determinism);
8. Enabled vast, deep, and general studies of other "completeness" theories;
9. Helps explain why verifying proofs seems to be easier than constructing them;
10. Illuminates the fundamental nature of algorithms and computation;

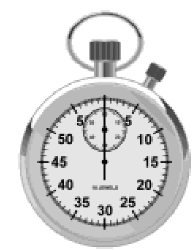
NP Completeness Benefits

11. Gave rise to new and novel mathematical approaches, proofs, and analyses;
12. Helps us to more easily reason about and manipulate large classes of problems;
13. Robustly decouples / abstracts complexity from underlying computational models;
14. Gives disciplined techniques for identifying "hardest" problems / languages;
15. Forged new unifications between computer science, mathematics, and logic;
16. NP-Completeness is interesting and fun!

Reducibilities Reloaded

Def: A language A is polynomial-time reducible to a language B if $\exists$ polynomial-time computable function f:$\Sigma^* \rightarrow \Sigma^*$ where $w \in A \Leftrightarrow f(w) \in B \;\forall w$





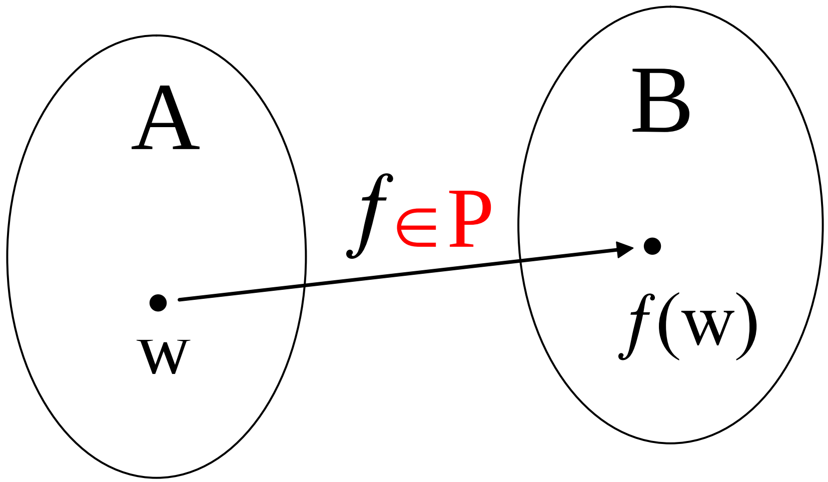
Note: f is a polynomial-time "reduction" of A to B

Denotation: $A \leq_P B$

Intuitively, A is "no harder" than B (modulo P)

Reducibilities Reloaded

Def: A language A is polynomial-time reducible to a language B if $\exists$ polynomial-time computable function $f:\Sigma^*\rightarrow\Sigma^*$ where $w\in A \Leftrightarrow f(w)\in B\ \forall w$



Note: be very careful about the reduction direction!

Theorem: If $A \le_P B$ and B is decidable within polynomial time then A is decidable within polynomial time.

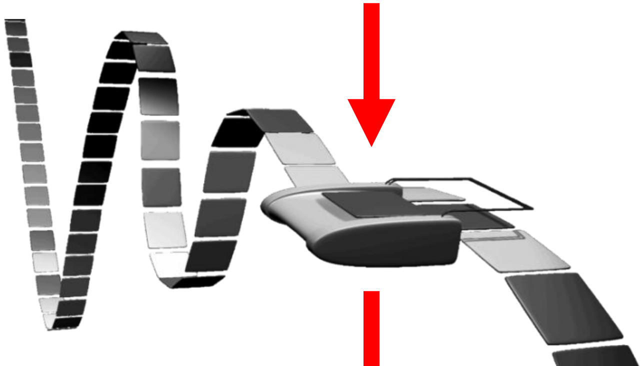
Theorem: If $A \le_P B$ and A is not decidable within polynomial time then B is not decidable within polynomial time.

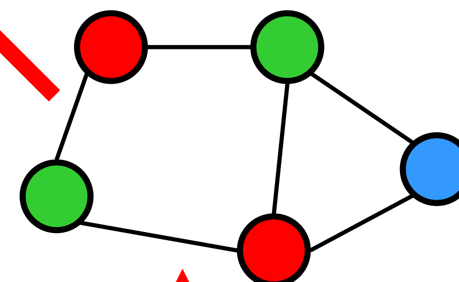
Problem Transformations

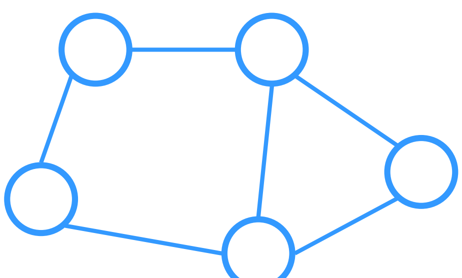
Idea: To solve a problem, efficiently transform to another problem, and then use a solver for the other problem:

Satisfiability
(x+y)(x'+y')

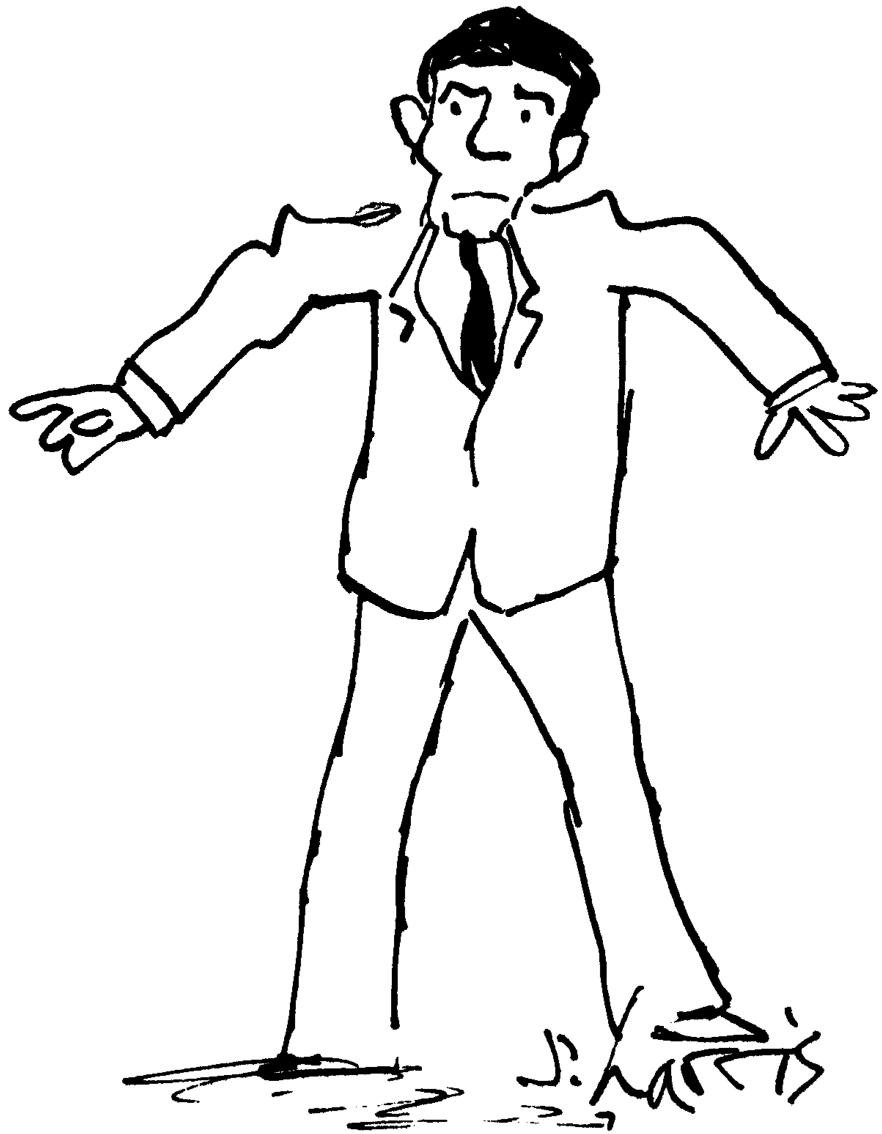
SAT solution
x=1, y=0







Colorability



AS LUCANUS, A GIANT BUG, AWOKE ONE MORNING FROM UNEASY DREAMS, HE FOUND HIMSELF TRANSFORMED INTO FRANZ KAFKA.

NP Hardness & Completeness

Def: A problem L' is NP-hard if:

(1) Every L in NP reduces to L' in polynomial time.

Def: A problem L' is NP-complete if:

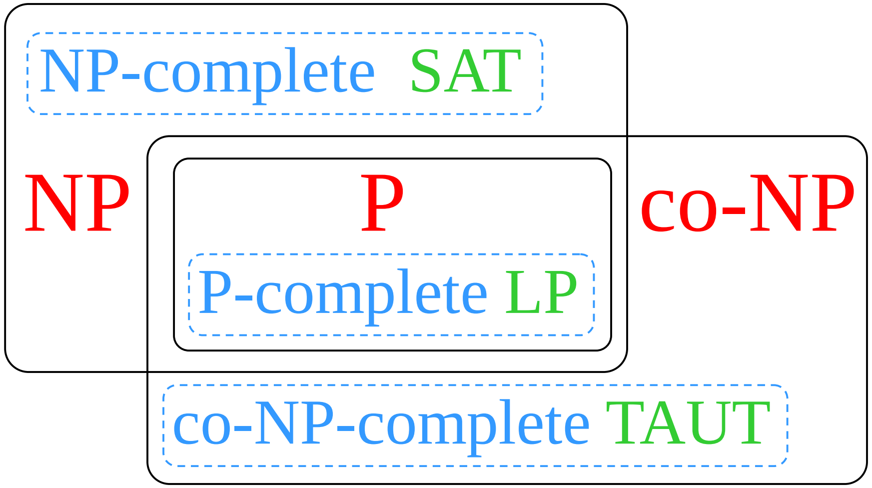
(1) L is NP-hard; and (2) L is in NP.

One NPC problem is in P $\Rightarrow$ P=NP

Open: is P=NP ?

Open: is NP=co-NP ?

Theorem: P = co-P



Boolean Satisfiability Problem (SAT)

Def: CNF (Conjunctive Normal Form) formula
is in a product-of-sums format.

Ex: $(x_1+x_4+x_5+x_7+x'_8)(x'_1+x_3+x'_4+x'_5)$

Def: A formula is satisfiable if it can be made true
by some assignment of all of its variables.

Problem (SAT): given an n-variable Boolean
formula (in CNF), is it satisfiable?

Ex: (x+y)(x'+z') is satisfiable (e.g., let x=1 & Z=0)
(x+z)(x')(z') is not satisfiable (why?)

The Cook/Levin Theorem


Stephen Cook


Leonid Levin

Theorem [Cook/Levin, 1971]: SAT is NP-complete.

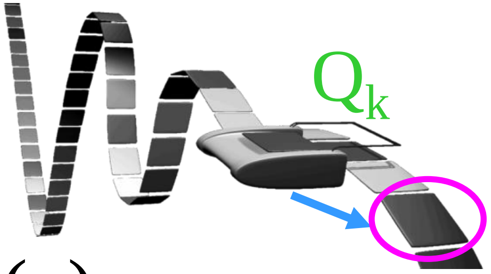
Proof idea: given a non-deterministic polynomial time TM M and input w, construct a CNF formula that is satisfiable iff M accepts w.

Create boolean variables:

q[i,k] $\Rightarrow$ at step i, M is in state k

h[i,k] $\Rightarrow$ at step i, M's RW head scans tape cell k

s[i,j,k] $\Rightarrow$ at step i, M's tape cell j contains symbol S_k



M halts in polynomial time p(n)

$\Rightarrow$ total # of variables is polynomial in p(n)

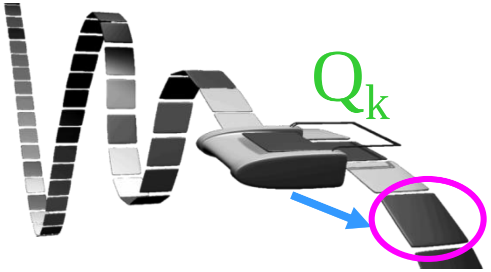
The Cook/Levin Theorem


Stephen Cook


Leonid Levin

Add clauses to the formula to enforce necessary restrictions on how M operates / runs:

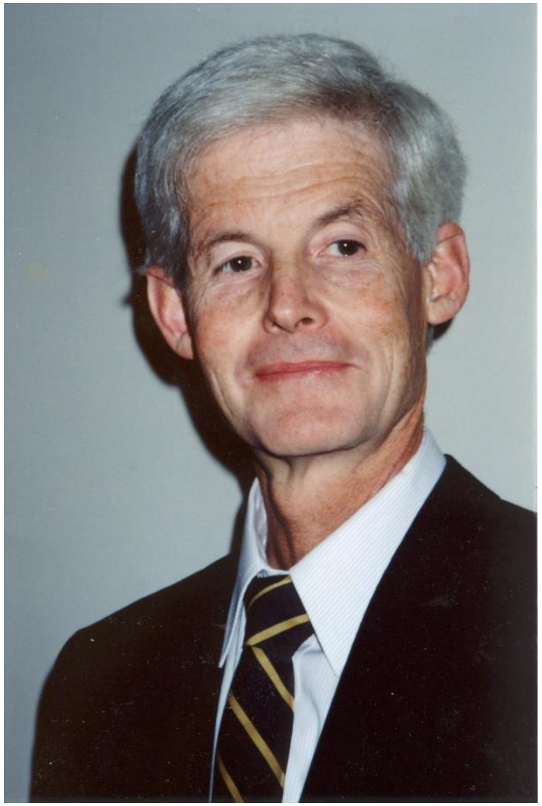
- At each time i:
  - M is in <u>exactly</u> 1 state
  - r/w head scans <u>exactly</u> 1 cell
  - All cells contain <u>exactly</u> 1 symbol
- At time 0 $\Rightarrow$ M is in its initial state
- At time P(n) $\Rightarrow$ M is in a final state
- Transitions from step i to i+1 all obey M's transition function



Resulting formula is satisfiable iff M accepts w!

Historical Note

The Cook/Levin theorem was independently proved by Stephen Cook and Leonid Levin



- Denied tenure at Berkeley (1970)
- Invented NP completeness (1971)
- Won Turing Award (1982)

- Student of Andrei Kolmogorov
- Seminal paper obscured by Russian, style, and Cold War

"Guess and Verify" Approach

Note: SAT $\in$ NP.

Idea: Nondeterministically "guess" each Boolean variable value, and then verify the guessed solution.

$\Rightarrow$ polynomial-time nondeterministic algorithm $\in$ NP

This "guess & verify" approach is general.

Idea: "Guessing" is usually trivially fast ($\in$ NP)

$\Rightarrow$ NP can be characterized by the "verify" property:

NP $\equiv$ set of problems for which proposed solutions can be quickly verified

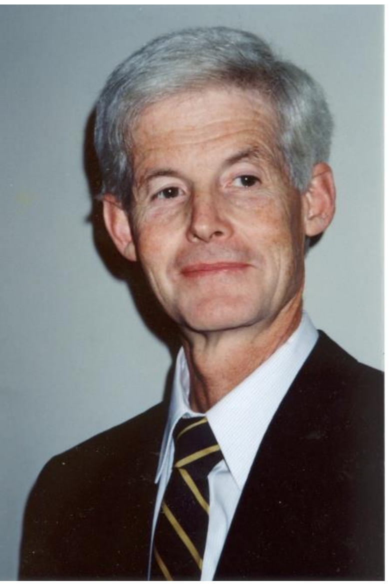
$\equiv$ set of languages for which string membership can be quickly tested.

Appears in
Proceedings Third Annual
ACM Symposium
Theory of Computing
May, 1971

The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto



Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a set of strings means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

1. Tautologies and Polynomial Re-Reducibility.

certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles in [1].

A query machine is a multitape Turing machine with a distinguished tape called the query tape, and three distinguished states called the query state, yes state, and no state, respectively. If M is a query machine and T is a set of strings, then a T-computation of M is a computation of M in which initially M is in the initial state and has an input string w on its input tape, and each time M assumes the query state there is a string u on the query tape, and the next state M assumes is the yes state if $u \in T$ and the no state if $u \notin T$. We think of an "oracle", which knows T, placing M in the

ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том IX 1973 Вып. 3

Entire paper!

КРАТКИЕ СООБЩЕНИЯ

УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем тождества элементов групп, гомеоморфности многообразий, разрешимости диофантовых уравнений и других). Тем самым был снят вопрос о нахождении практического способа их решения. Однако существование алгоритмов для решения других задач не снимает для них аналогичного вопроса из-за фантастически большого объема работы, предписываемого этими алгоритмами. Такова ситуация с так называемыми переборными задачами: минимизации булевых функций, поиска доказательств ограниченной длины, выяснения изоморфности графов и другими. Все эти задачи решаются тривиальными алгоритмами, состоящими в переборе всех возможностей. Однако эти алгоритмы требуют экспоненциального времени работы и у математиков сложилось убеждение, что более простые алгоритмы для них невозможны. Был получен ряд серьезных аргументов в пользу его справедливости (см.[1, 2]), однако доказать это утверждение не удалось никому. (Например, до сих пор не доказано, что для нахождения математических доказательств нужно больше времени, чем для их проверки.)

Однако если предположить, что вообще существует какая-нибудь (хотя бы искусственно построенная) массовая задача переборного типа, неразрешимая простыми (в смысле объема вычислений) алгоритмами, то можно показать, что этим же свойством обладают и многие «классические» переборные задачи (в том числе задача минимизации, задача поиска доказательств и др.). В этом и состоят основные результаты статьи.

Функции $f(n)$ и $g(n)$ будем называть сравнимыми, если при некотором k

$$f(n) \leqslant (g(n)+2)^k \quad \text{и} \quad g(n) \leqslant (f(n)+2)^k.$$

Аналогично будем понимать термин «меньше или сравнимо».

Определение. Задачей переборного типа (или просто переборной задачей) будем называть задачу вида «по данному x найти какое-нибудь y длины, сравнимой с длиной x, такое, что выполняется $A(x, y)$», где $A(x, y)$ – какое-нибудь свойство, проверяемое алгоритмом, время работы которого сравнимо с длиной x. (Под алгоритмом здесь можно понимать, например, алгоритмы Колмогорова — Успенского или машины Тьюринга, или нормальные алгоритмы; x, y – двоичные слова). Квазипереборной задачей будем называть задачу выяснения, существует ли такое y.

Мы рассмотрим шесть задач этих типов. Рассматриваемые в них объекты кодируются естественным образом в виде двоичных слов. При этом выбор естественной кодировки не существен, так как все они дают сравнимые длины кодов.

Задача 1. Заданы списком конечное множество и покрытие его 500-элементными подмножествами. Найти подпокрытие заданной мощности (соответственно выяснить существует ли оно).

Задача 2. Таблично задана частичная булева функция. Найти заданного размера дизъюнктивную нормальную форму, реализующую эту функцию в области определения (соответственно выяснить существует ли она).

Задача 3. Выяснить, выводима или опровержима данная формула исчисления высказываний. (Или, что то же самое, равна ли константе данная булева формула.)

Задача 4. Даны два графа. Найти гомоморфизм одного на другой (выяснить его существование).

Задача 5. Даны два графа. Найти изоморфизм одного в другой (на его часть).

Задача 6. Рассматриваются матрицы из целых чисел от 1 до 100 и некоторое условие о том, какие числа в них могут соседствовать по вертикали и какие по горизонтали. Заданы числа на границе и требуется продолжить их на всю матрицу с соблюдением условия.

Пусть $f(n)$ — монотонная функция.

Теорема 1. *Если вообще существует какая-нибудь массовая задача переборного (квазипереборного) типа, неразрешимая за время, меньшее $f(n)$ при длине аргумента, сравнимой с n, то этим же свойством обладают задачи 1—6.*

Идея доказательства состоит в том, что задачи 1—6 являются «универсальными задачами перебора».

Определение. Пусть $A(x, y)$ и $B(x, y)$ определяют соответственно переборные задачи A и B. Мы говорим, что задача A сводится к B, если есть три алгоритма $r(x)$, $p(y)$ и $s(y)$, работающие за время, сравнимое с длиной аргумента, такие, что $A(x, p(y)) \equiv B(r(x), y)$ и $A(x, y) \equiv B(r(x), s(y))$ (т. е. по A – задаче x легко построить эквивалентную B задачу $r(x)$). Задача, к которой сводится любая задача перебора, называется «универсальной».

Таким образом, суть доказательства теоремы 1 состоит в следующей лемме.

Лемма 1. *Задачи 1—6 являются универсальными переборными задачами.*

Описанный метод, по-видимому, позволяет легко получить результаты типа теоремы 1 и леммы 1 для большинства интересных переборных задач. Однако остается проблема доказать условие, имеющееся в этой теореме. В этом направлении давно уже делаются многочисленные попытки и получен ряд интересных результатов (см., например, [3, 4]). Впрочем, универсальность различных массовых задач перебора можно устанавливать и без решения этой проблемы. В системе алгоритмов Колмогорова – Успенского может быть доказана также следующая

Теорема 2. *Для произвольной массовой переборной задачи $A(x, y)$ существует алгоритм, решающий ее за время, оптимальное с точностью до умножения на константу и прибавления величины, сравнимой с длиной x.*

Автор выражает искреннюю благодарность А. Н. Колмогорову, Б. А. Трахтенброту, Я. М. Бардзиню, Ю. И. Альбртону и М. И. Дегтярю за ценное обсуждение.

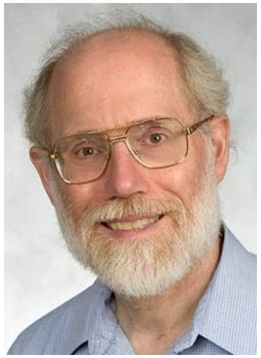
ЛИТЕРАТУРА

1. Яблонский С. В. Об алгоритмических трудностях синтеза минимальных контактных схем. Сб. «Проблемы кибернетики», 2. М., Физматгиз, 1959, 75—121.
2. Журавлев Ю. И. Теоретико-множественные методы в алгебре логики. Сб. «Проблемы кибернетики», 8. М., Физматгиз, 1962, 5—44.
3. Трахтенброт Б. А. Оптимальные вычисления и частотное явление Яблонского. Семинар. Новосибирск, «Наука», СО, 1965, 4, 5, 79—93.
4. Дегтярь М. И. О невозможности элиминации полного перебора при вычислении функций относительно их графиков. Докл. АН СССР, 1969, 189, 4, 748—751.

Поступила в редакцию
7 июня 1972 г.

An NP-Complete Encyclopedia

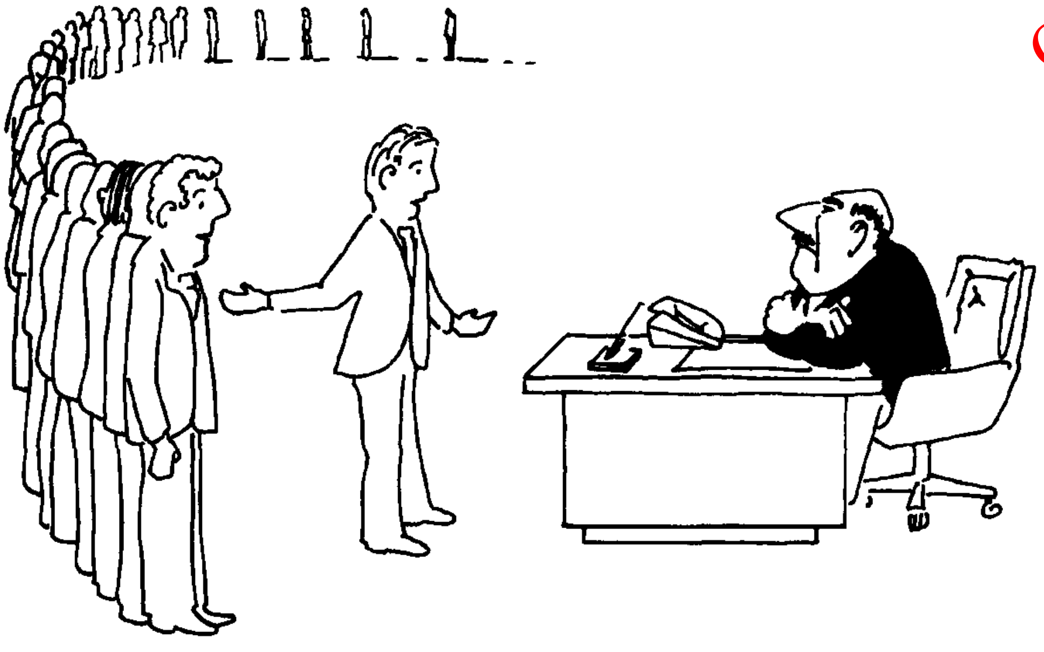

Michael Garey

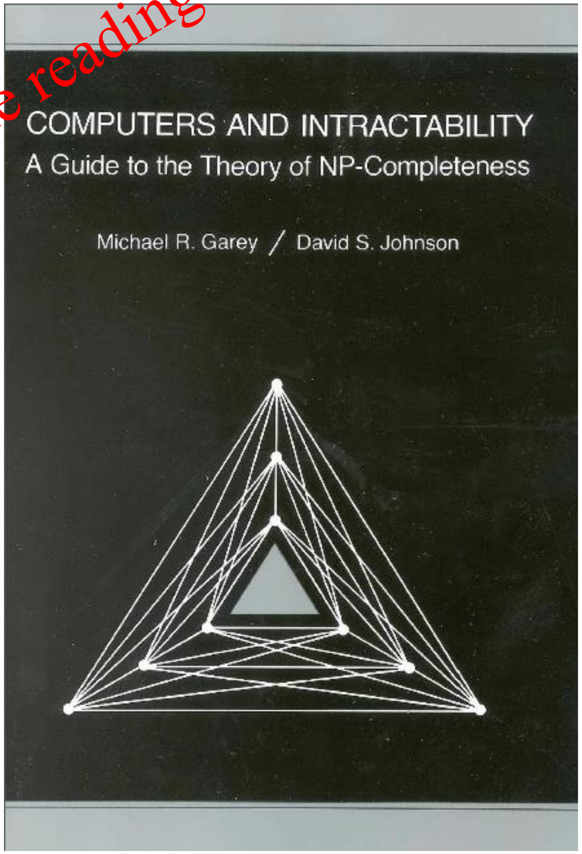

David Johnson

Classic book: Garey & Johnson, 1979

- Definitive guide to NP-completeness
- Lists hundreds of NP-complete problems
- Gives reduction types and refs

On the reading list!


"I can't find an efficient algorithm, but neither can all these famous people."

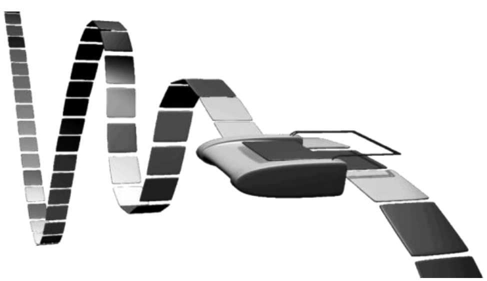


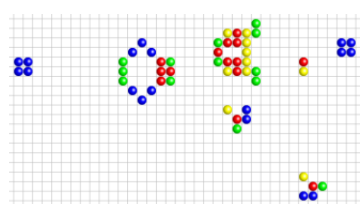
Robustness of P and NP

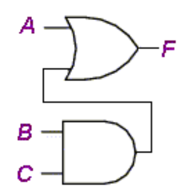
Compositions of polynomials yields polynomials

Computation models' efficiencies are all polynomially related (i.e., can efficiently simulate one another).

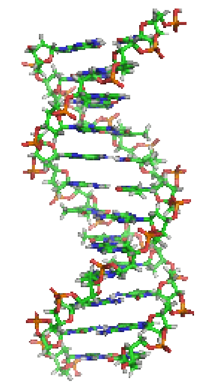
Defs of P and NP is computation model-independent!

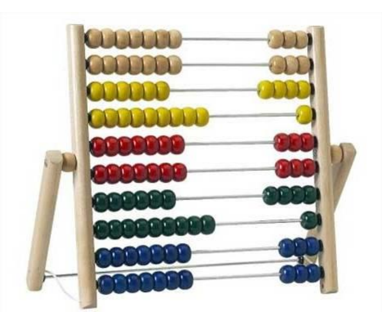


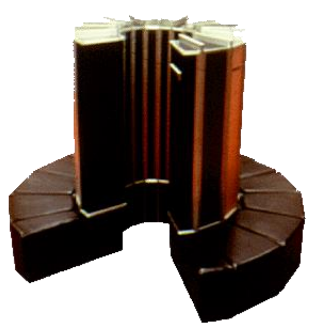




μ

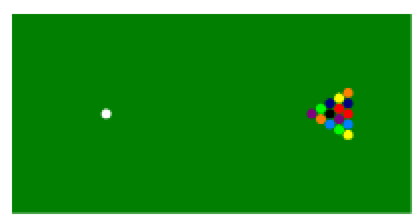


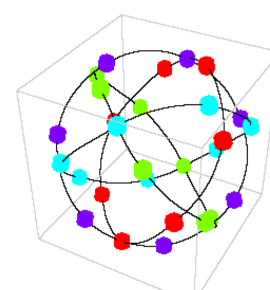




λ

$x^3 + y^3 + z^3 = 33$





Clay Mathematics Institute

Dedicated to increasing and disseminating mathematical knowledge

P vs NP Problem

Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971.

- The Millennium Problems
- Official Problem Description — Stephen Cook
- Lecture by Vijaya Ramachandran at University of Texas (video)
- Minesweeper

Clay Mathematics Institute

Dedicated to increasing and disseminating mathematical knowledge

HOME | ABOUT CMI | PROGRAMS | NEWS & EVENTS | AWARDS | SCHOLARS | PUBLICATIONS

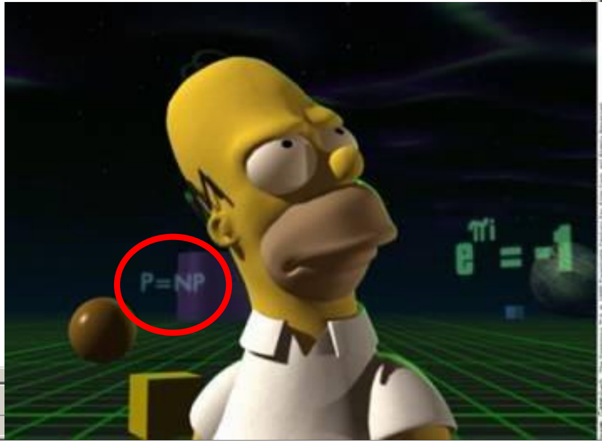
Millennium Problems

In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven *Prize Problems*. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a $7 million prize fund for the solution to these problems, with $1 million allocated to each. During the Millennium Meeting held on May 24, 2000 at the Collège de France, Timothy Gowers presented a lecture entitled *The Importance of Mathematics*, aimed for the general public, while John Tate and Michael Atiyah spoke on the problems. The CMI invited specialists to formulate each problem.

One hundred years earlier, on August 8, 1900, David Hilbert delivered his famous lecture about open mathematical problems at the second International Congress of Mathematicians in Paris. This influenced our decision to announce the millennium problems as the central theme of a Paris meeting.

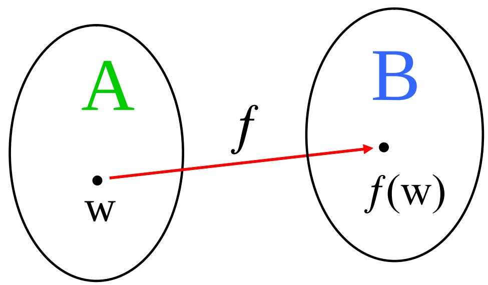
The rules for the award of the prize have the endorsement of the CMI Scientific Advisory Board and the approval of the Directors. The members of these boards have the responsibility to preserve the nature, the integrity, and the spirit of this prize.

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs NP ??
- ~~Poincaré Conjecture~~ Perelman 2006
- Riemann Hypothesis
- Yang-Mills Theory
- Rules
- Millennium Meeting Videos



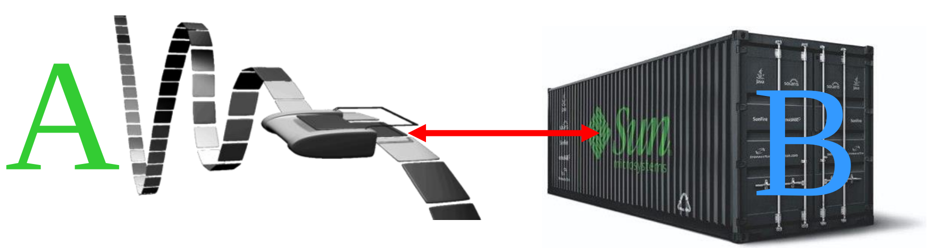
Reduction Types

Many-one reduction: converts an instance of one problem to a single instance of another problem.



$A \leq_M B$

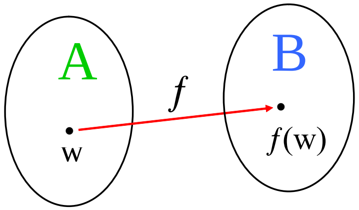
Turing reduction: solves a problem A by multiple calls to an "oracle" for problem B.



$A \leq_T B$

Polynomial-Time Reduction Types

Polynomial-time many-one reduction: transforms in polynomial time an instance of problem A to an instance of problem B.

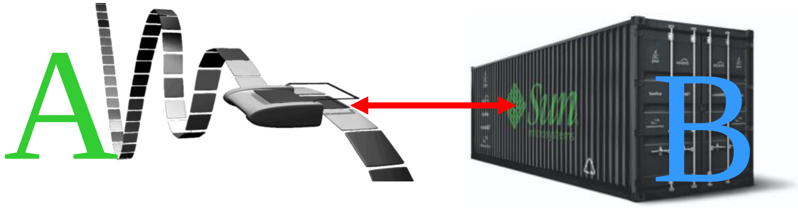


⇒ "Karp" reduction (transformation)

Richard Karp

Polynomial-time Turing reduction: solves problem A by polynomially-many calls to "oracle" for B.

⇒ "Cook" reduction



Stephen Cook

Open: do polynomial-time-bounded many-one and Turing reductions yield the same complexity classes?

(NP, co-NP, NP-complete, co-NP-complete, etc.)

Boolean 3-Satisfiability (3-SAT)

Def: 3-CNF: each sum term has exactly 3 literals.

Ex: $(x_1+x_5+x_7)(x_3+x'_4+x'_5)$

Def: 3-SAT: given an n-variable boolean formula (in CNF), is it satisfiable?

Theorem: 3-SAT is NP-complete.

Proof: convert each long clause of the given formula into an equivalent set of 3-CNF clauses:

Ex: $(x+y+z+u+v+w)$

$\Rightarrow (x+y+a)(a'+z+b)(b'+u+c)(c'+v+w)$

Resulting formula is satisfiable iff original formula is.

1-SAT and 2-SAT

Idea: Determine the "boundary of intractability" by varying / trivializing some of the parameters.

Q: Is 1-SAT NP-complete?

A: No (look for a variable & its negation)

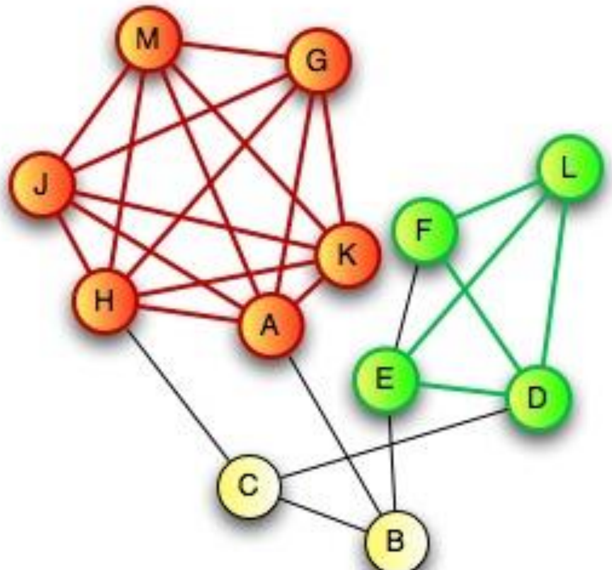
Q: Is 2-SAT NP-complete?

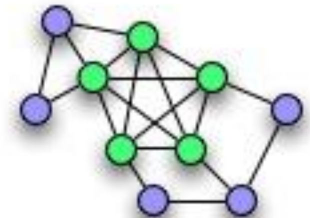
A: No (cycles in the implication graph)

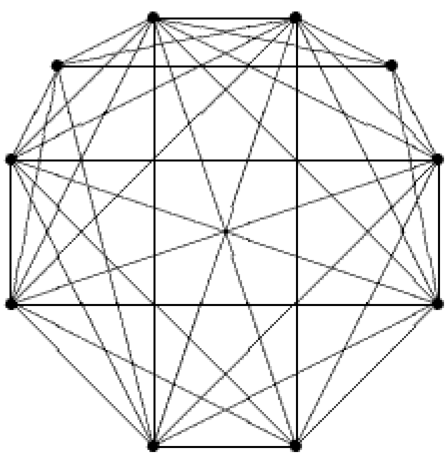
Classic NP Complete Problems

Richard Karp

Clique: given a graph and integer k, is there a subgraph that is a complete graph of size k?

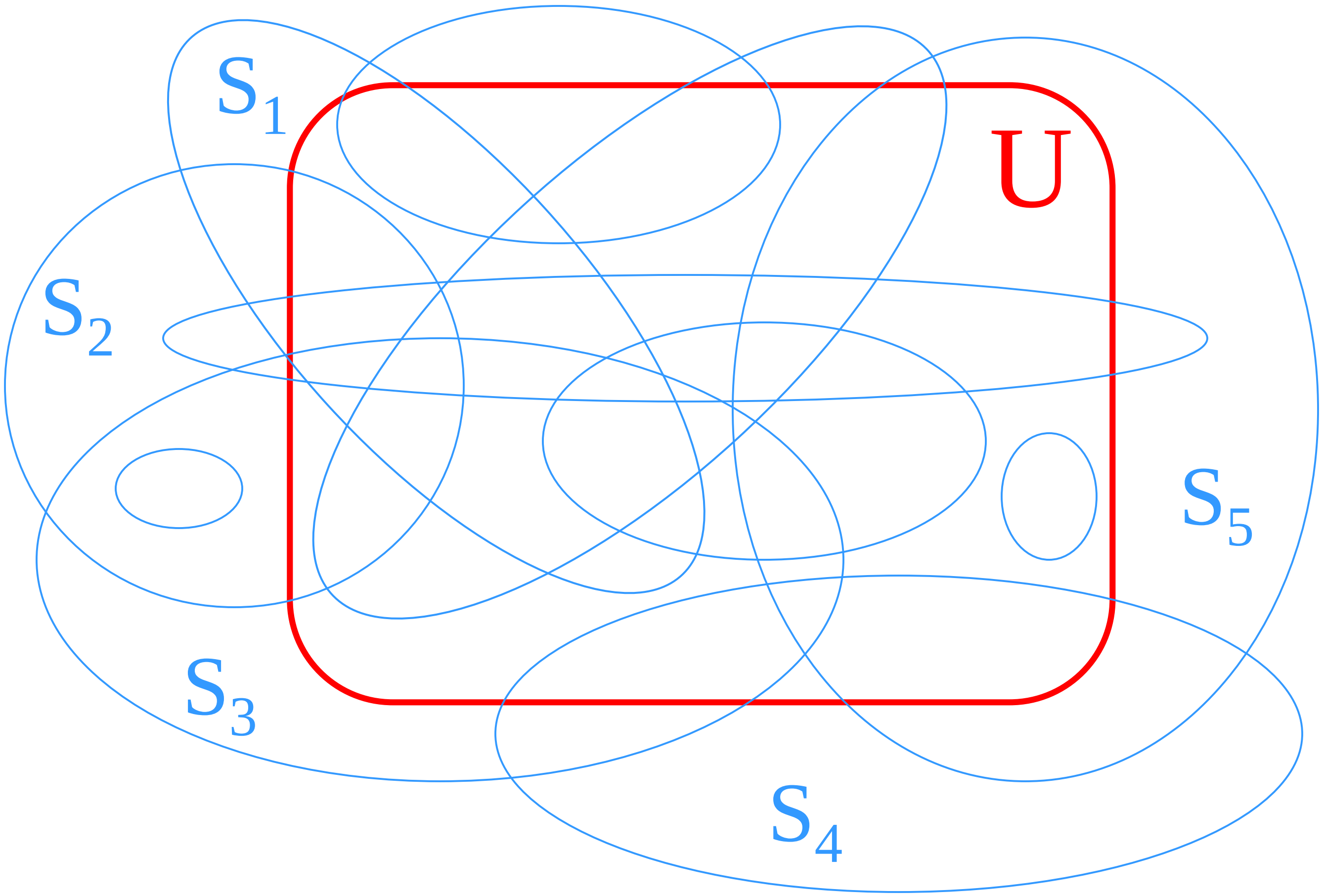


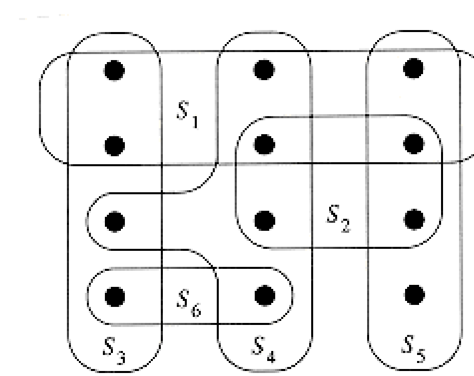




Classic NP Complete Problems

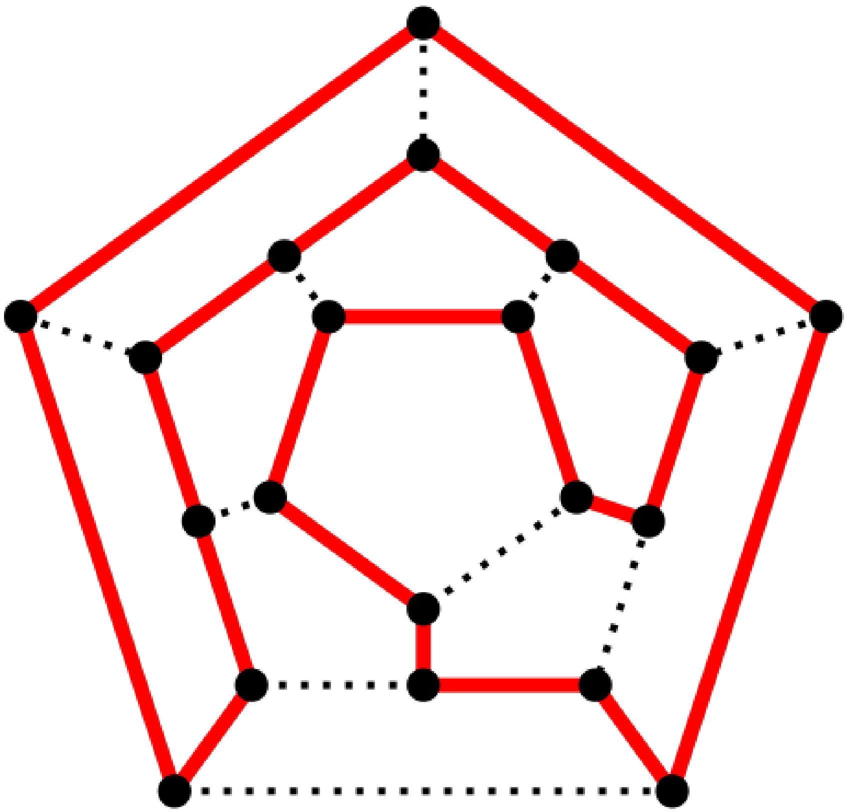
Set Cover: given a universe U, a collection of subsets S_i and an integer k, can k of these subsets cover U?

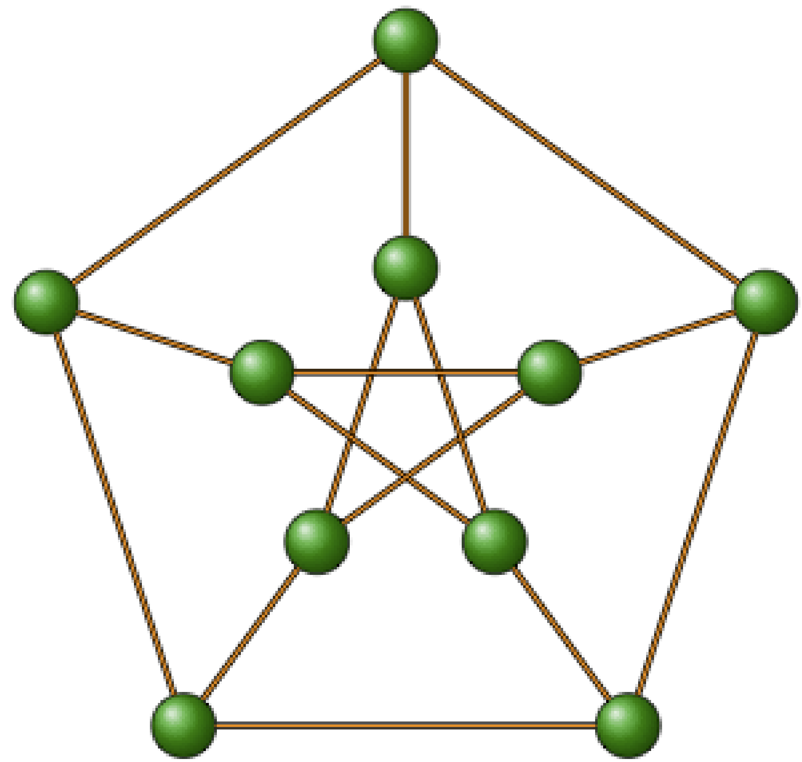


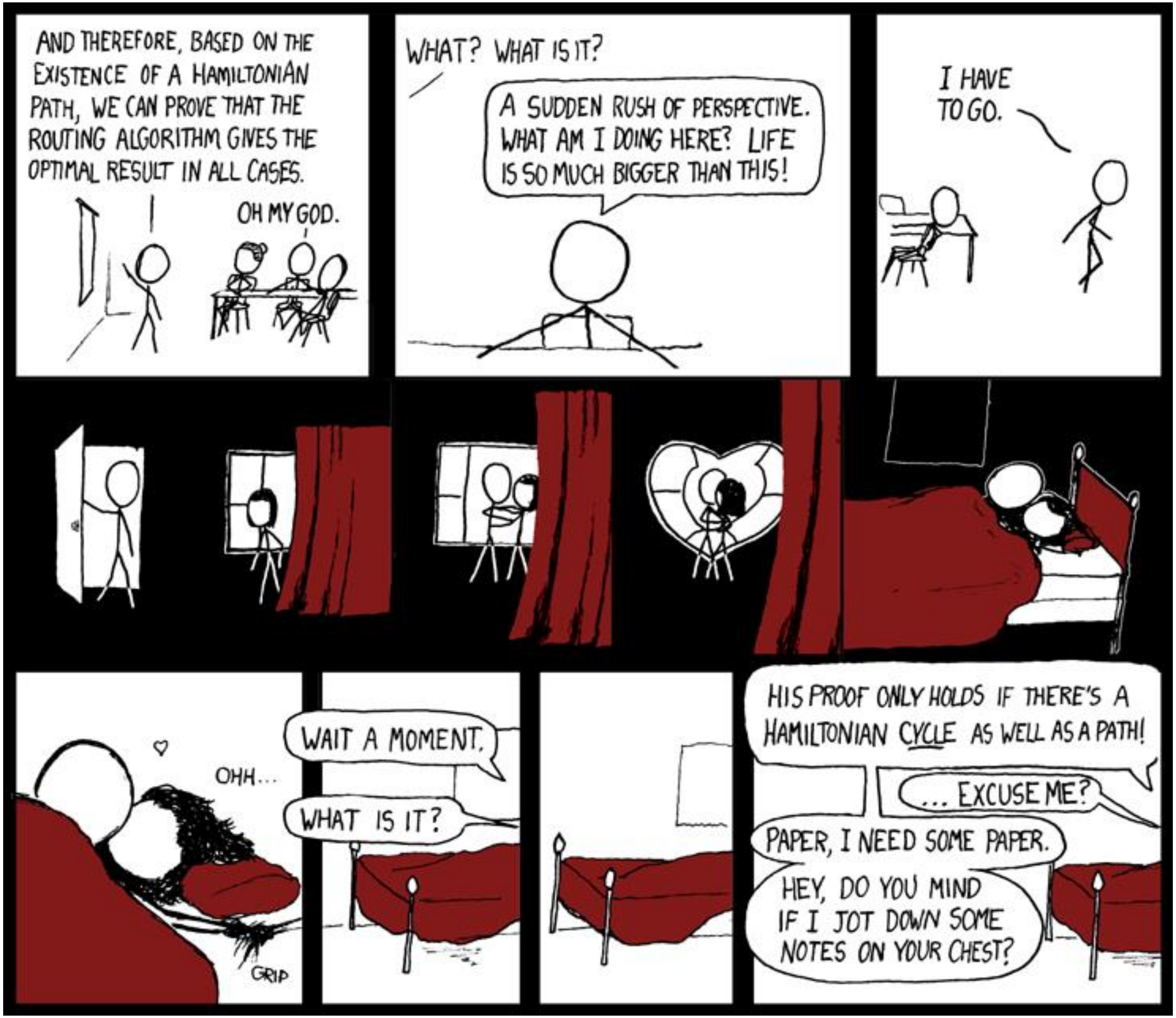


Classic NP Complete Problems

Hamiltonian cycle: Given an undirected graph, is there a closed path that visits every vertex exactly once?

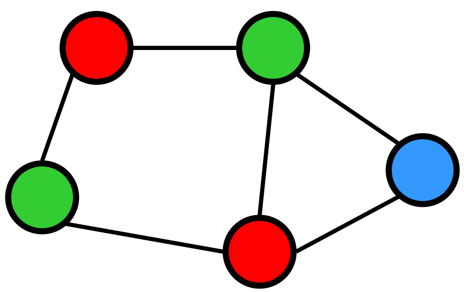


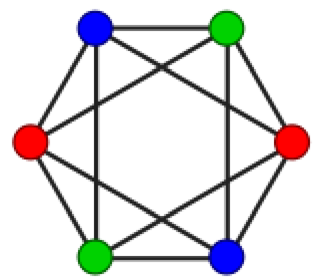


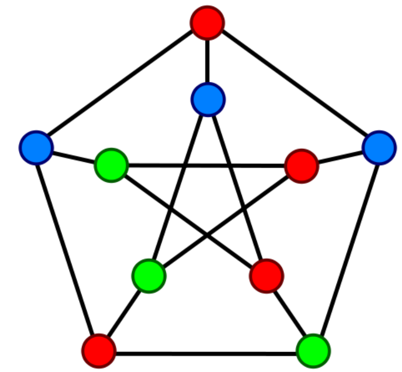

AND THEREFORE, BASED ON THE EXISTENCE OF A HAMILTONIAN PATH, WE CAN PROVE THAT THE ROUTING ALGORITHM GIVES THE OPTIMAL RESULT IN ALL CASES.
OH MY GOD.
WHAT? WHAT IS IT?
A SUDDEN RUSH OF PERSPECTIVE. WHAT AM I DOING HERE? LIFE IS SO MUCH BIGGER THAN THIS!
I HAVE TO GO.
OHH...
GRIP
WAIT A MOMENT.
WHAT IS IT?
HIS PROOF ONLY HOLDS IF THERE'S A HAMILTONIAN CYCLE AS WELL AS A PATH!
... EXCUSE ME?
PAPER, I NEED SOME PAPER.
HEY, DO YOU MIND IF I JOT DOWN SOME NOTES ON YOUR CHEST?

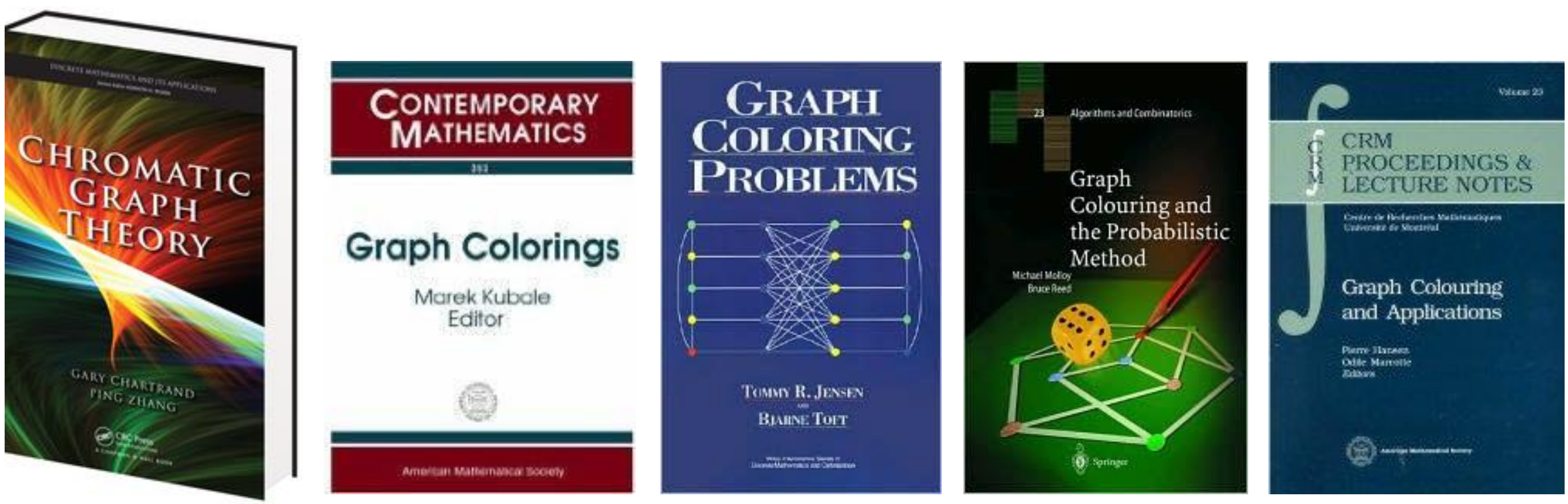
Classic NP Complete Problems

Graph coloring: given an integer k and a graph, is it k-colorable? (adjacent nodes get different colors)



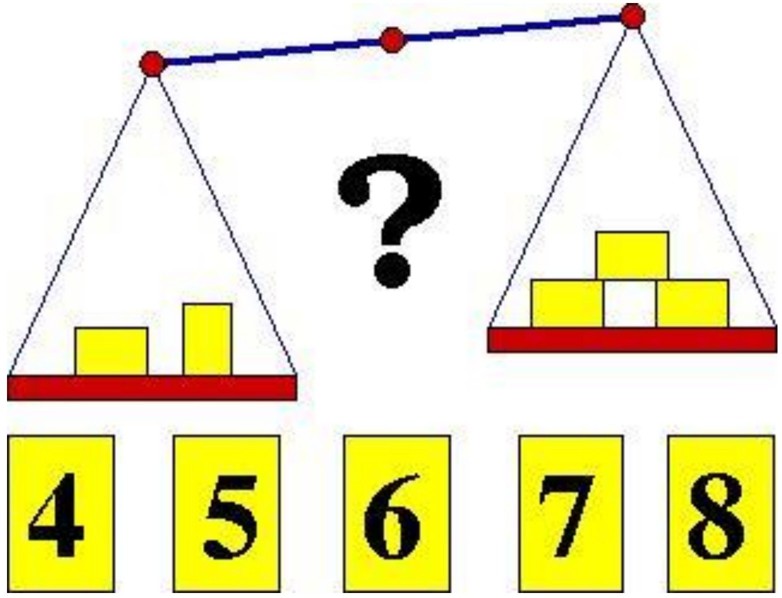






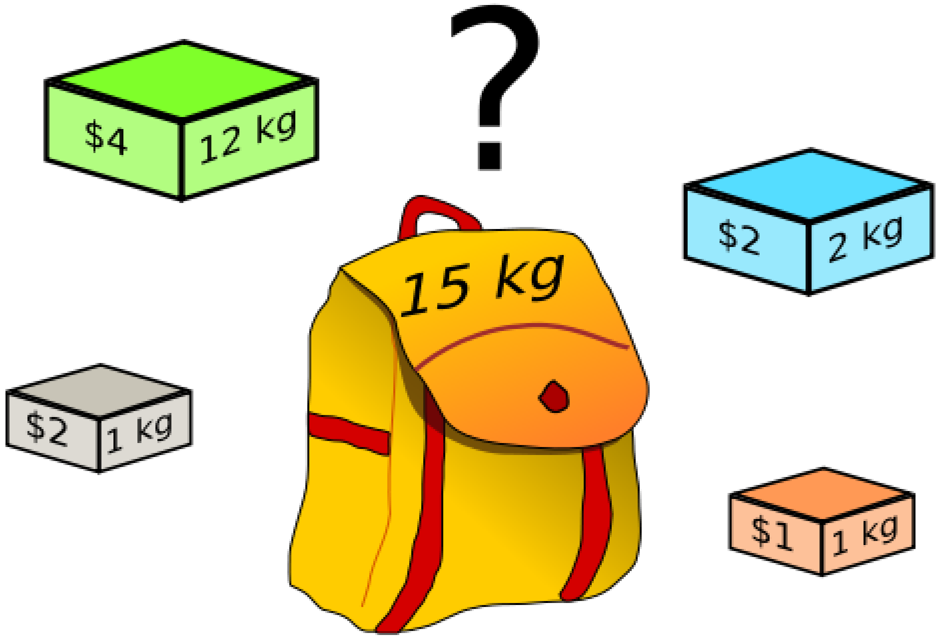
Classic NP Complete Problems

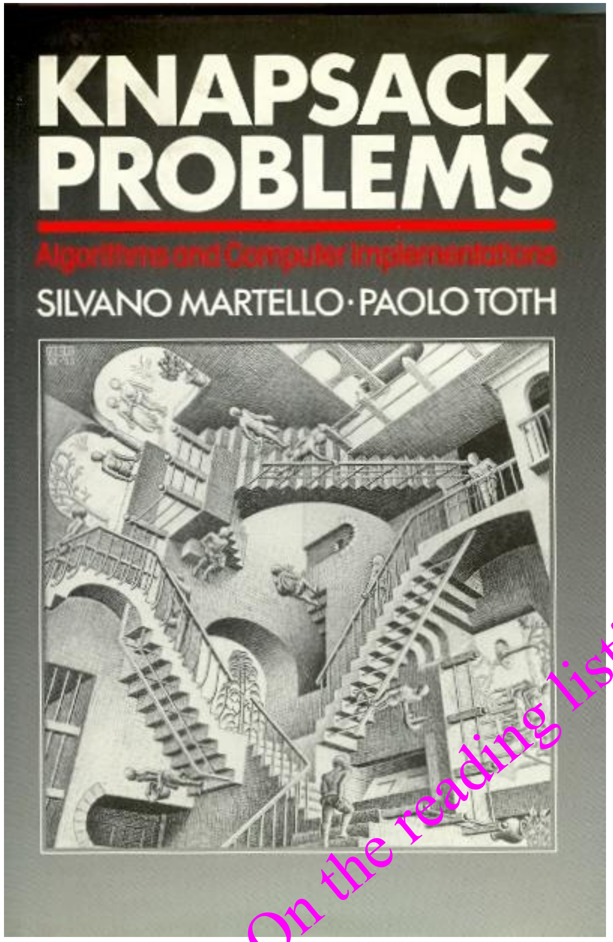
Partition: Given a set of integers, is there a way to partition is into two subsets each with the same sum?



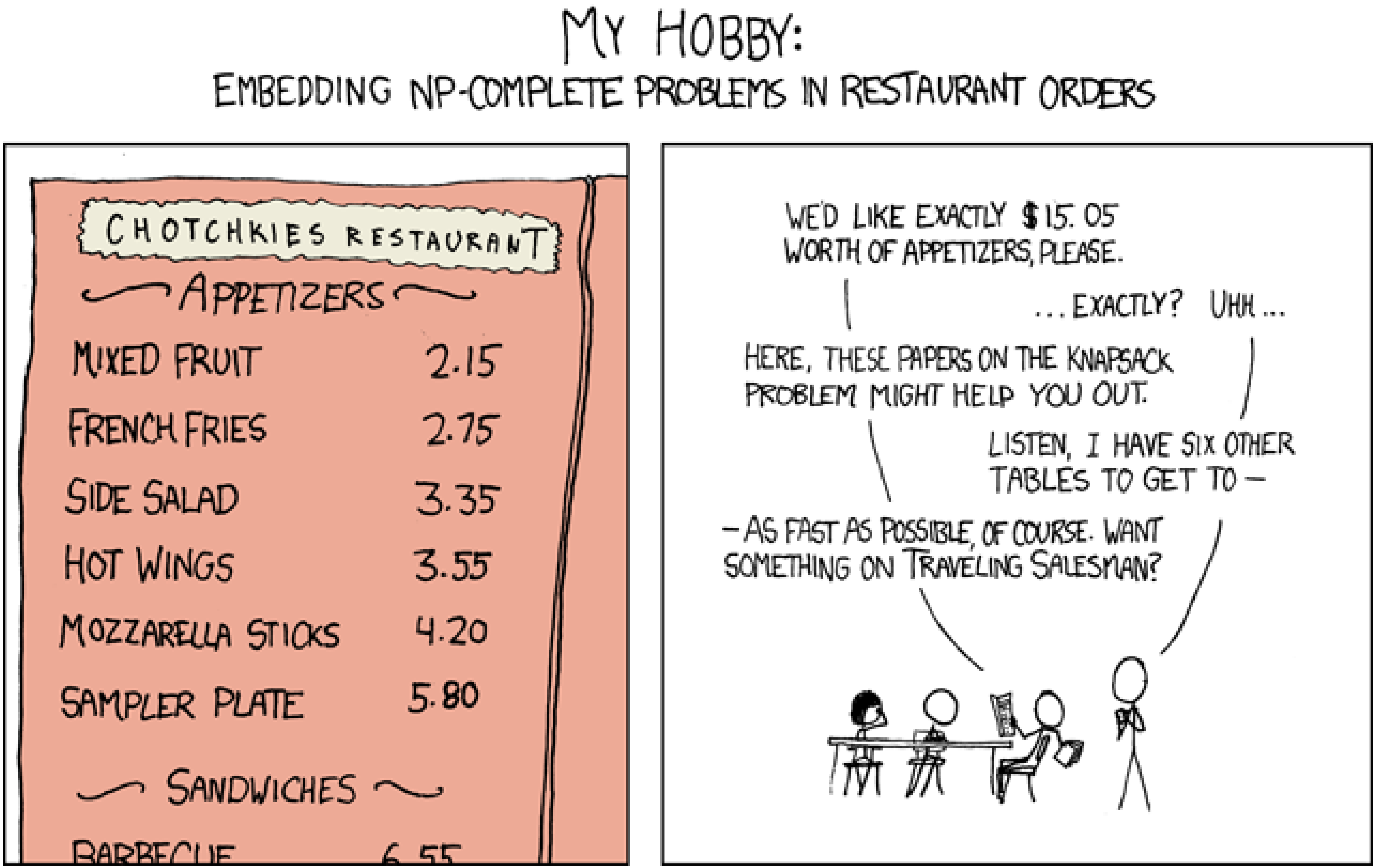
Classic NP Complete Problems

Knapsack: maximize the total value of a set of items without exceeding an overall weight constraint.



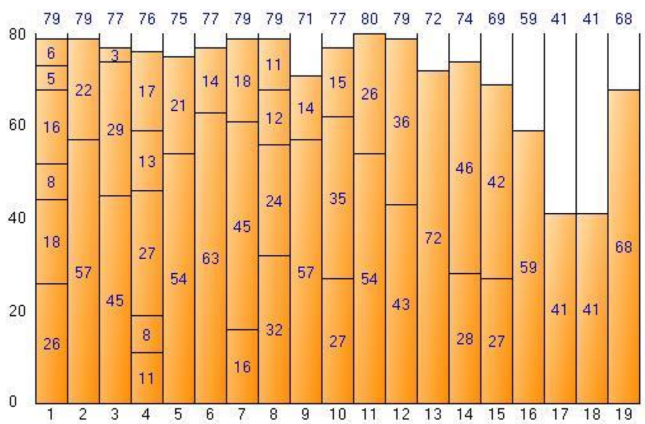


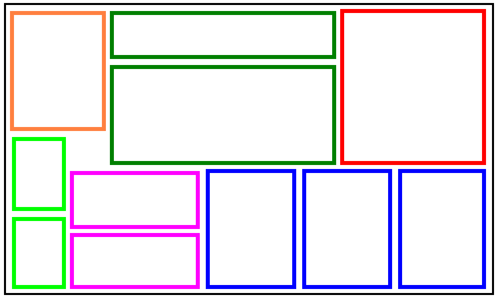
On the reading list!

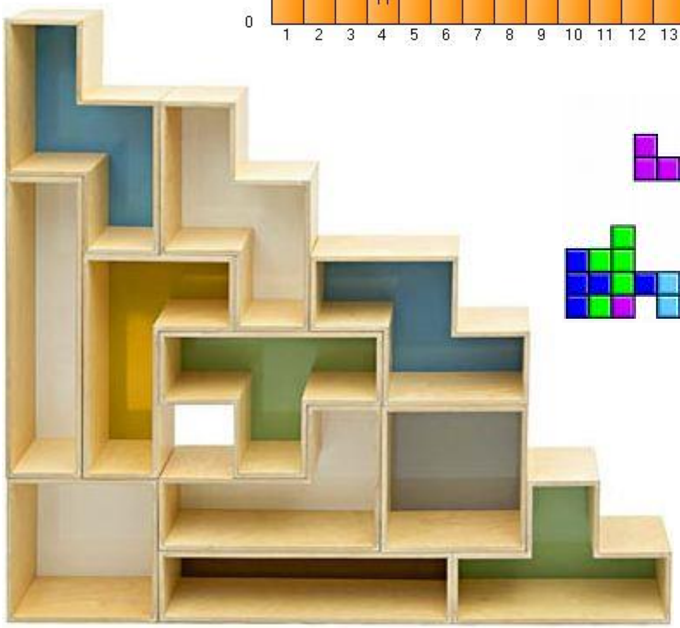

MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO –
– AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

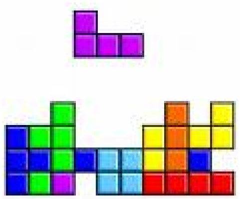
NP Complete Problems

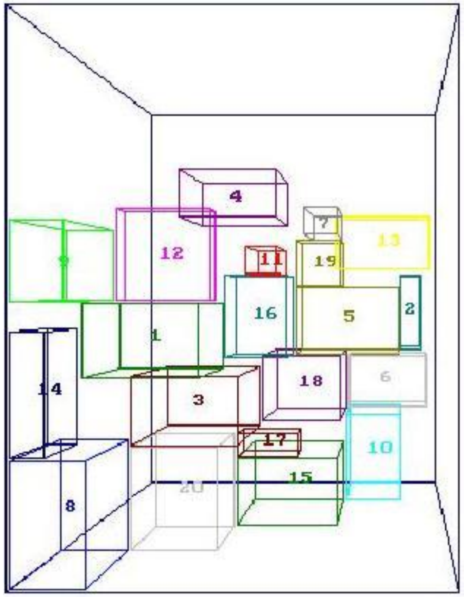
Bin packing: minimize the number of same-size bins necessary to hold a set of items of various sizes.





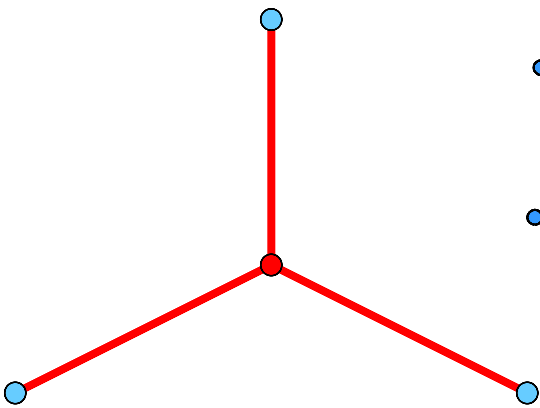


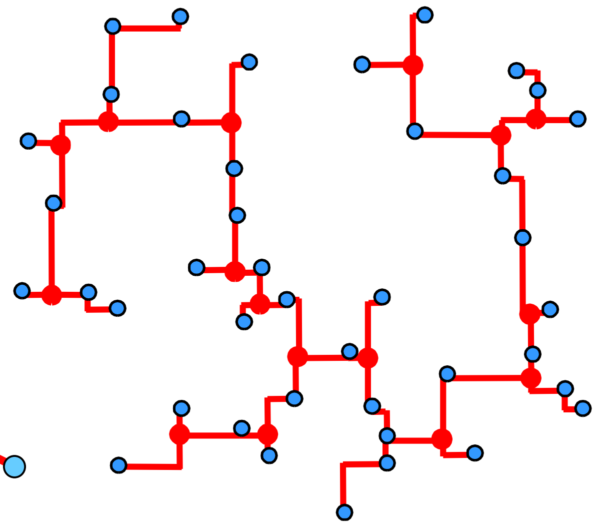


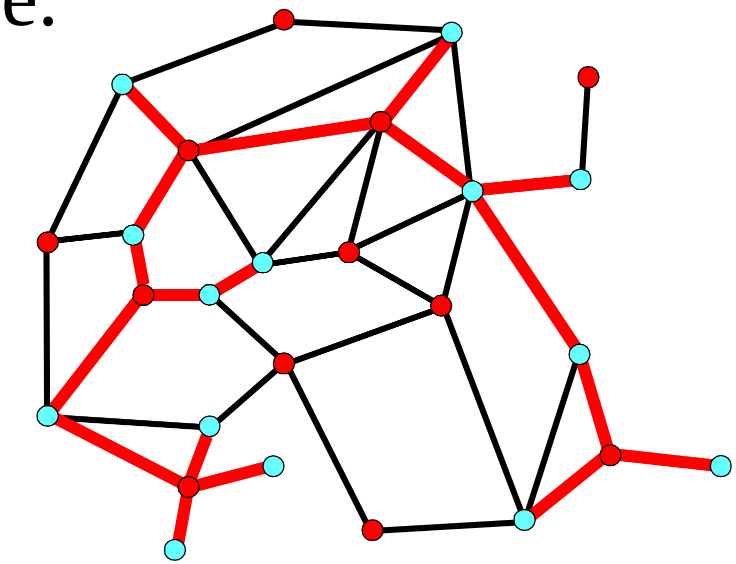


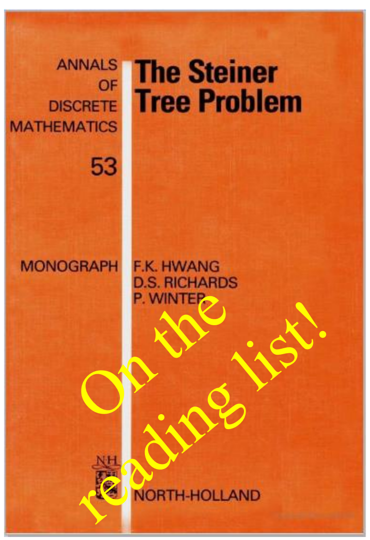
Other Classic NP Complete Problems

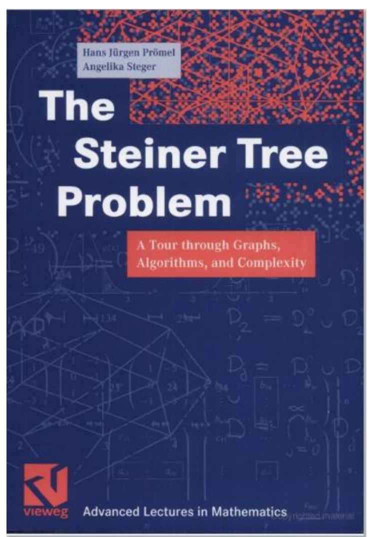
Steiner Tree: span a given node subset in a weighted graph using a minimum-cost tree.

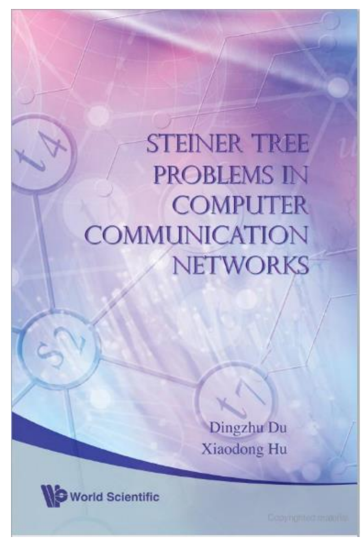


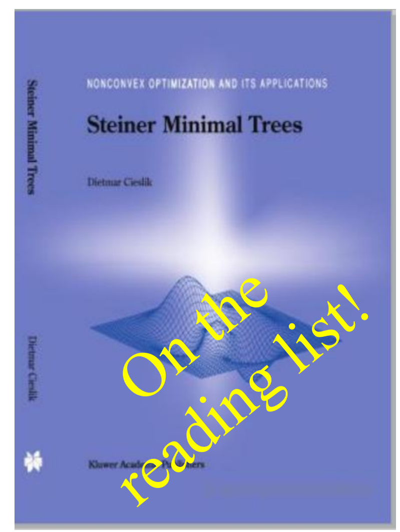


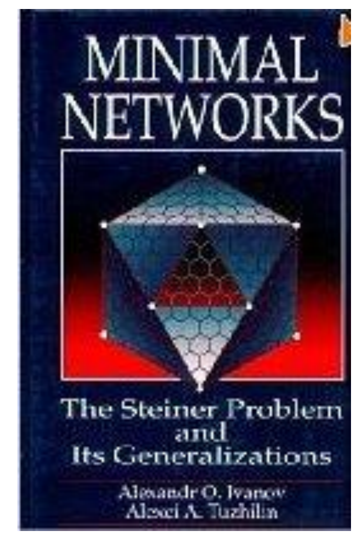






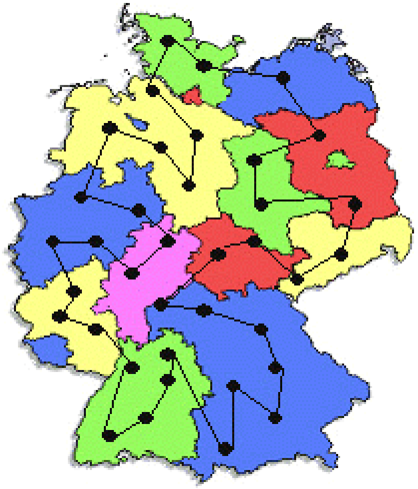


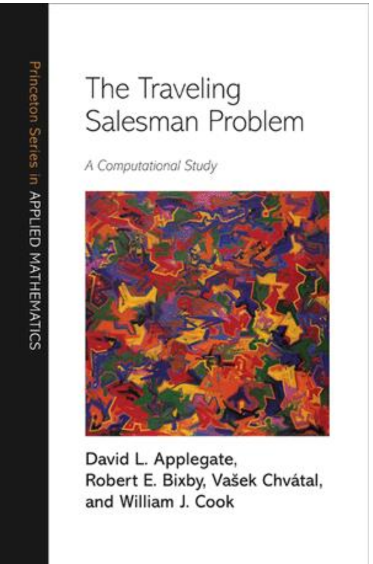


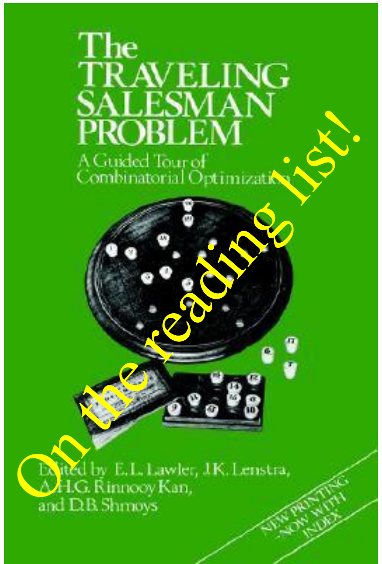


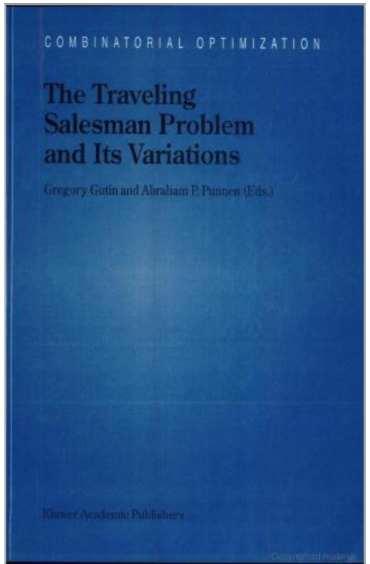
Other Classic NP Complete Problems

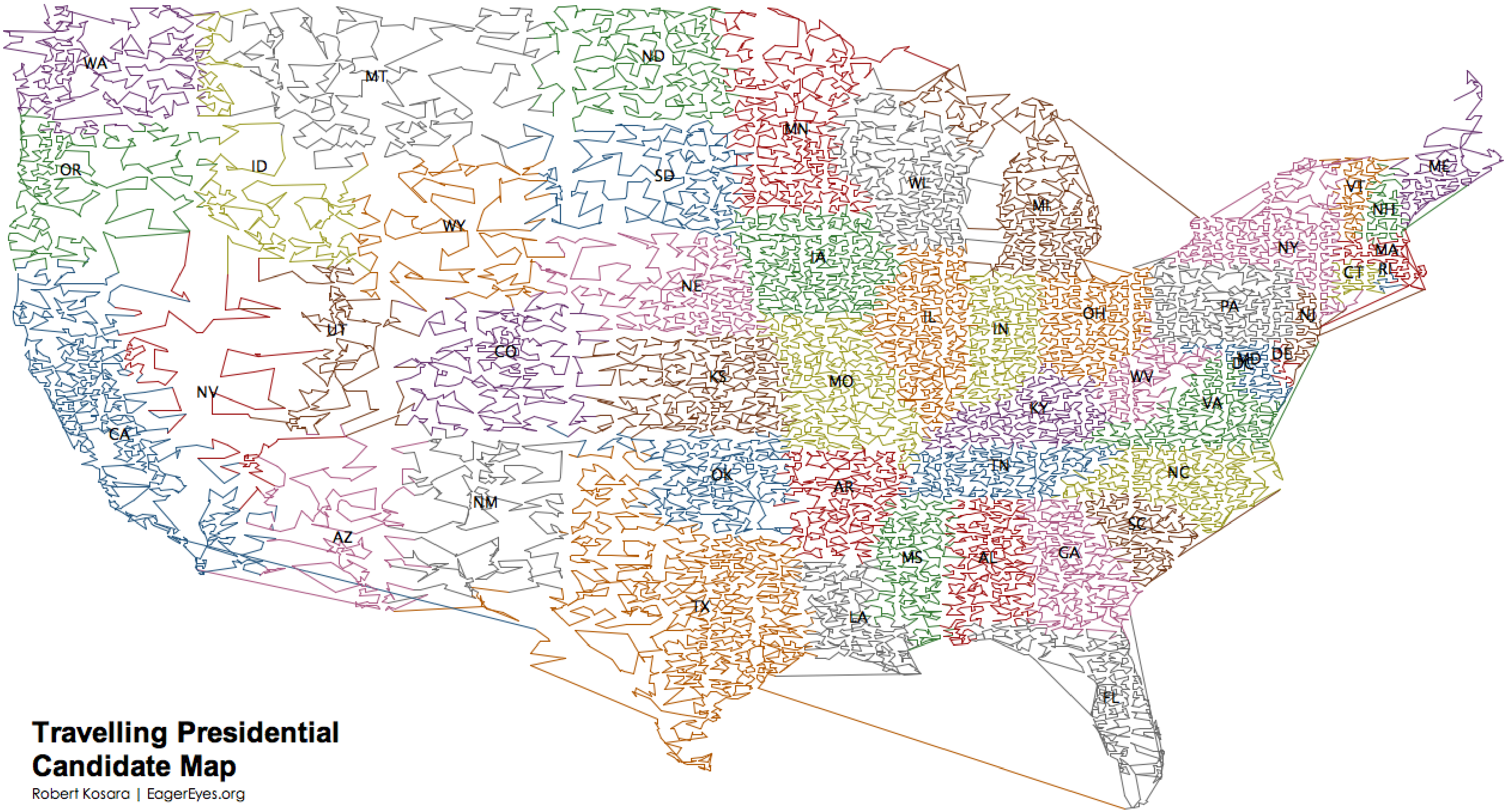
Traveling salesperson: given a set of points, find the shortest tour that visits every point exactly once.





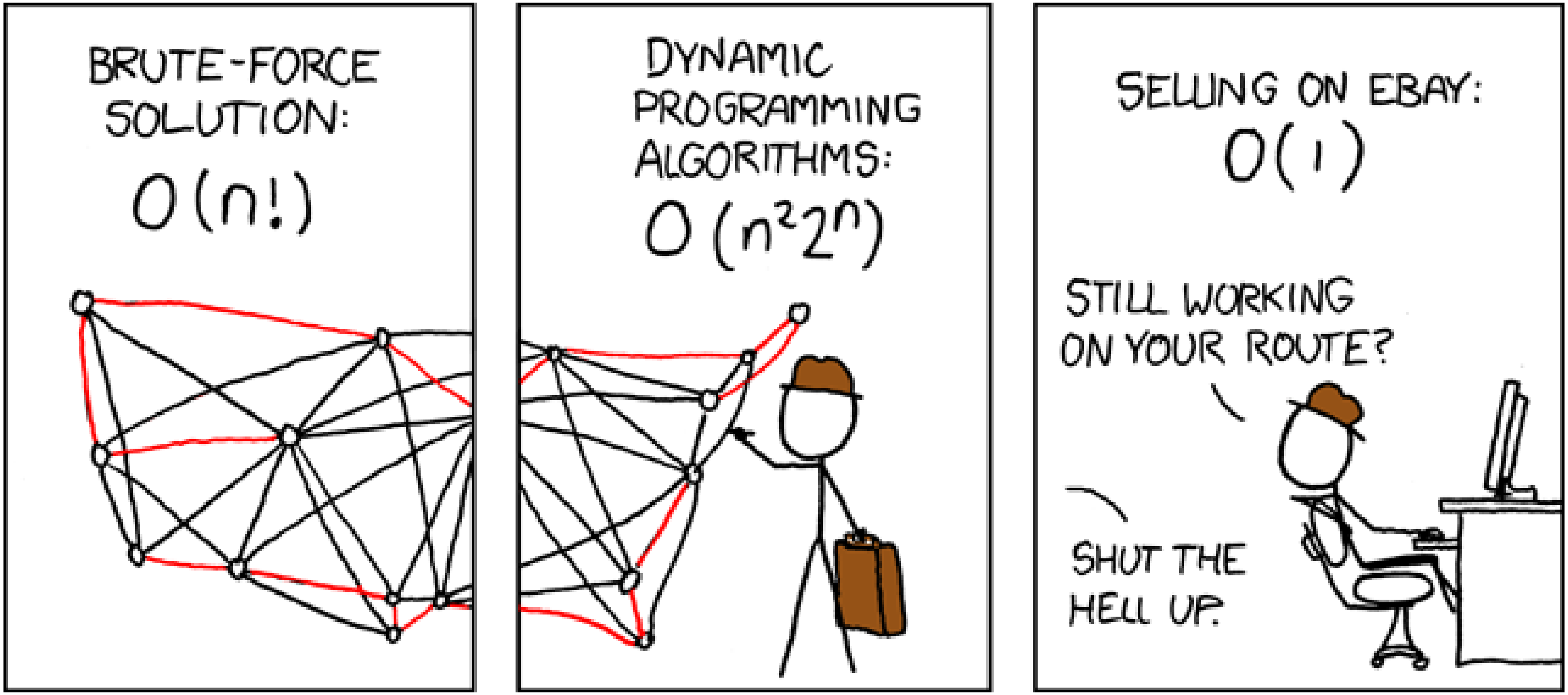


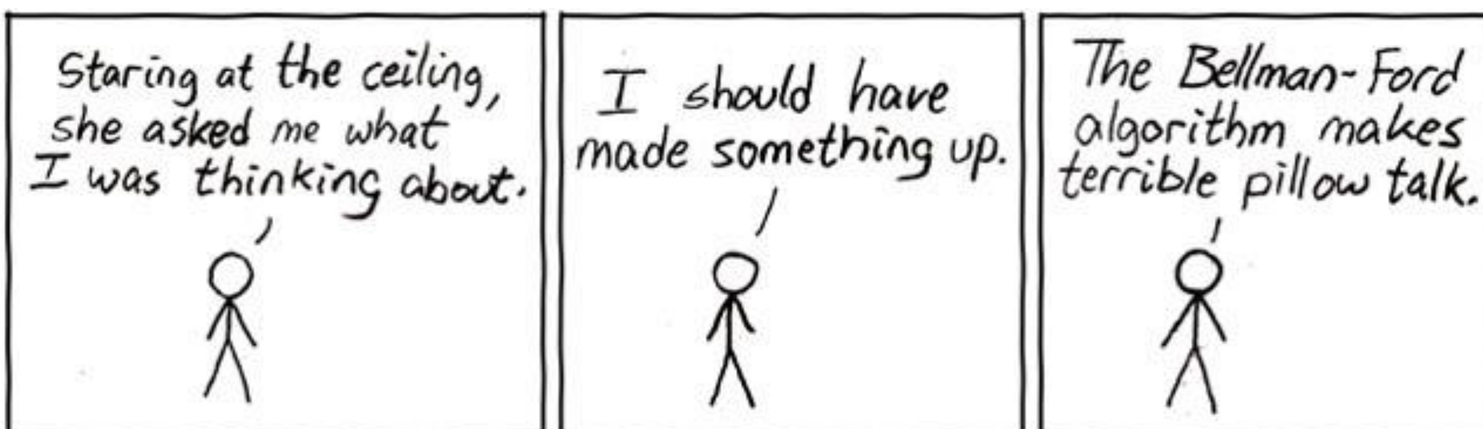




Travelling Presidential Candidate Map

Robert Kosara | EagerEyes.org


BRUTE-FORCE SOLUTION:
O(n!)
DYNAMIC PROGRAMMING ALGORITHMS:
O(n²2ⁿ)
SELLING ON EBAY:
O(1)
STILL WORKING ON YOUR ROUTE?
SHUT THE HELL UP.

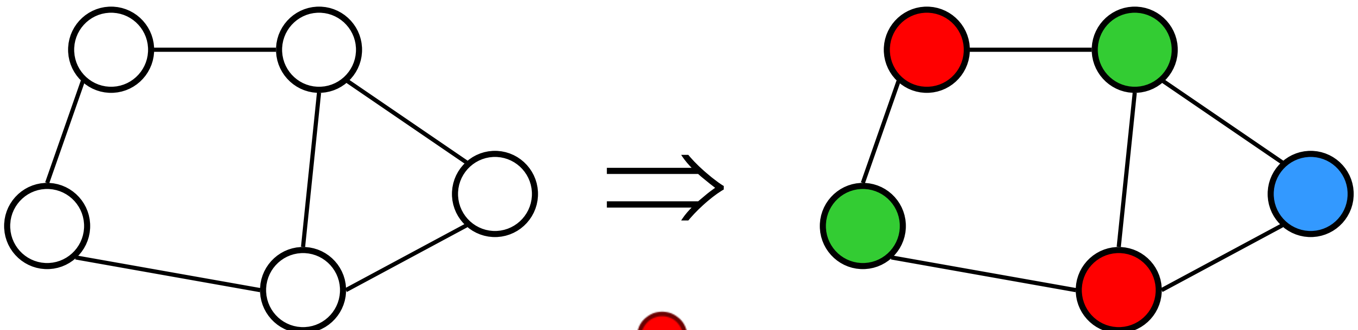

Staring at the ceiling, she asked me what I was thinking about.
I should have made something up.
The Bellman-Ford algorithm makes terrible pillow talk.

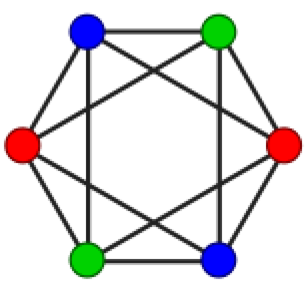
Graph Colorability

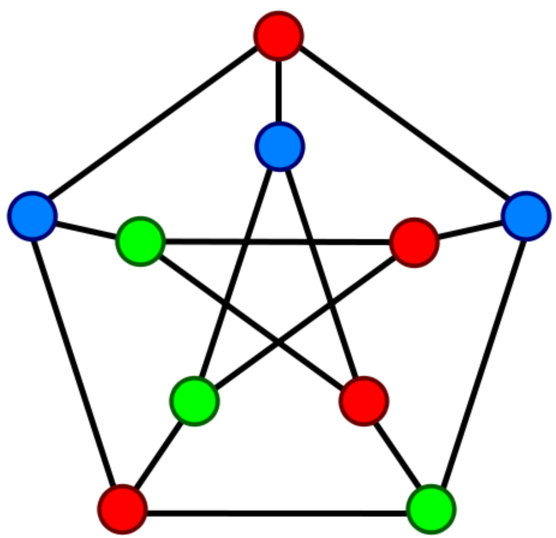
Problem: given a graph G and an integer k,

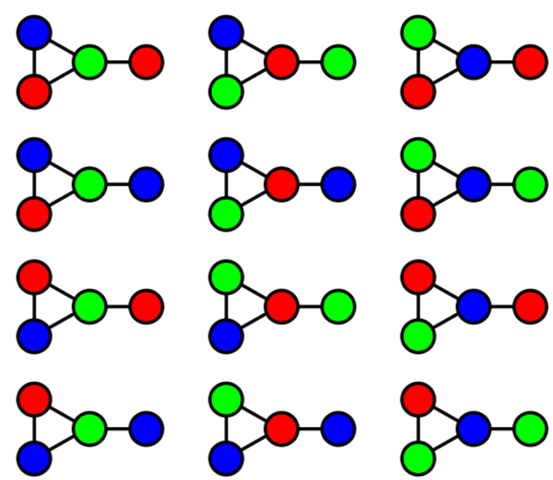
is G k-colorable?

Note: adjacent nodes must have different colors









from "Complexity of Computer Computations", pp. 85–103, 1972.

REDUCIBILITY AMONG COMBINATORIAL PROBLEMS†

Richard M. Karp

University of California at Berkeley

On the reading list!

Abstract: A large class of computational problems involve the determination of properties of graphs, digraphs, integers, arrays of integers, finite families of finite sets, boolean formulas and elements of other countable domains. Through simple encodings from such domains into the set of words over a finite alphabet these problems can be converted into language recognition problems, and we can inquire into their computational complexity. It is reasonable to consider such a problem satisfactorily solved when an algorithm for its solution is found which terminates within a number of steps bounded by a polynomial in the length of the input. We show that a large number of classic unsolved problems of covering, matching, packing, routing, assignment and sequencing are equivalent, in the sense that either each of them possesses a polynomial-bounded algorithm or none of them does.

1. INTRODUCTION

All the general methods presently known for computing the chromatic number of a graph, deciding whether a graph has a Hamilton circuit, or solving a system of linear inequalities in which the variables are constrained to be 0 or 1, require a combinatorial search for which the worst case time requirement grows exponentially with the length of the input. In this paper we give theorems which strongly suggest, but do not imply, that these problems, as well as many others, will remain intractable perpetually.

†This research was partially supported by National Science Foundation Grant GJ-474.

We are specifically interested in the existence of algorithms that are guaranteed to terminate in a number of steps bounded by a polynomial in the length of the input. We exhibit a class of well-known combinatorial problems, including those mentioned above, which are equivalent, in the sense that a polynomial-bounded algorithm for any one of them would effectively yield a polynomial-bounded algorithm for all. We also show that, if these problems do possess polynomial-bounded algorithms then all the problems in an unexpectedly wide class (roughly speaking, the class of problems solvable by polynomial-depth backtrack search) possess polynomial-bounded algorithms.

The following is a brief summary of the contents of the paper. For the sake of definiteness our technical development is carried out in terms of the recognition of languages by one-tape Turing machines, but any of a wide variety of other abstract models of computation would yield the same theory. Let Σ^* be the set of all finite strings of 0's and 1's. A subset of Σ^* is called a _language_. Let $\mathcal{P}$ be the class of languages recognizable in polynomial time by one-tape deterministic Turing machines, and let $\mathcal{NP}$ be the class of languages recognizable in polynomial time by one-tape nondeterministic Turing machines. Let Π be the class of functions from Σ^* into Σ^* computable in polynomial time by one-tape Turing machines. Let L and M be languages. We say that $L \propto M$ (L _is reducible to_ M) if there is a function $f \in \Pi$ such that $f(x) \in M \Leftrightarrow x \in L$. If $M \in \mathcal{P}$ and $L \propto M$ then $L \in \mathcal{P}$. We call L and M equivalent if $L \propto M$ and $M \propto L$. Call L _(polynomial) complete_ if $L \in \mathcal{NP}$ and every language in $\mathcal{NP}$ is reducible to L. Either all complete languages are in $\mathcal{P}$, or none of them are. The former alternative holds if and only if $\mathcal{P} = \mathcal{NP}$.

The main contribution of this paper is the demonstration that a large number of classic difficult computational problems, arising in fields such as mathematical programming, graph theory, combinatorics, computational logic and switching theory, are complete (and hence equivalent) when expressed in a natural way as language recognition problems.

This paper was stimulated by the work of Stephen Cook (1971), and rests on an important theorem which appears in his paper. The author also wishes to acknowledge the substantial contributions of Eugene Lawler and Robert Tarjan.

2. THE CLASS $\mathcal{P}$

There is a large class of important computational problems which involve the determination of properties of graphs, digraphs, integers, finite families of finite sets, boolean formulas and elements of other countable domains. It is a reasonable working hypothesis, championed originally by Jack Edmonds (1965) in connection with problems in graph theory and integer programming, and by now widely accepted, that such a problem can be regarded as tractable if and only if there is an algorithm for its solution whose running time is bounded by a polynomial in the size of the input. In this section we introduce and begin to investigate the class of problems solvable in polynomial time.

We begin by giving an extremely general definition of "deterministic algorithm", computing a function from a countable domain D into a countable range R.

For any finite alphabet A, let A^* be the set of finite strings of elements of A; for $x \in A^*$, let $\lg(x)$ denote the length of x.

A deterministic algorithm A is specified by:

a countable set D (the domain)
a countable set R (the range)
a finite alphabet Δ such that $\Delta^* \wedge R = \phi$
an encoding function $E: D \to \Delta^*$
a transition function $\tau: \Delta^* \to \Delta^* \cup R$.

The computation of A on input $x \in D$ is the unique sequence $y_1, y_2, \ldots$ such that $y_1 = E(x)$, $y_{i+1} = \tau(y_i)$ for all i and, if the sequence is finite and ends with y_k, then $y_k \in R$. Any string occurring as an element of a computation is called an instantaneous description. If the computation of A on input x is finite and of length $t(x)$, then $t(x)$ is the running time of A on input x. A is terminating if all its computations are finite. A terminating algorithm A computes the function $f_A: D \to R$ such that $f_A(x)$ is the last element of the computation of A on x.

If $R = \{\text{ACCEPT},\text{REJECT}\}$ then A is called a recognition algorithm. A recognition algorithm in which $D = \Sigma^*$ is called a string recognition algorithm. If A is a string recognition algorithm then the language recognized by A is $\{x \in \Sigma^* \mid f_A(x) = \text{ACCEPT}\}$. If $D = R = \Sigma^*$ then A is called a string mapping algorithm. A terminating algorithm A with domain $D = \Sigma^*$ operates in polynomial time if there is a polynomial $p(\cdot)$ such that, for every $x \in \Sigma^*$, $t(x) \le p(\lg(x))$.

To discuss algorithms in any practical context we must specialize the concept of deterministic algorithm. Various well known classes of string recognition algorithms (Markov algorithms, one-tape Turing machines, multitape and multihead Turing machines, random access machines, etc.) are delineated by restricting the functions E and τ to be of certain very simple types. These definitions are standard [Hopcroft & Ullman (1969)] and will not be repeated here. It is by now commonplace to observe that many such classes are equivalent in their capability to recognize languages; for each such class of algorithms, the class of languages recognized is the class of recursive languages. This invariance under changes in definition is part of the evidence that recursiveness is the correct technical formulation of the concept of decidability.

The class of languages recognizable by string recognition algorithms which operate in polynomial time is also invariant under a wide range of changes in the class of algorithms. For example, any language recognizable in time $p(\cdot)$ by a multihead or multitape Turing machine is recognizable in time $p^2(\cdot)$ by a one-tape Turing machine. Thus the class of languages recognizable in polynomial time by one-tape Turing machines is the same as the class recognizable by the ostensibly more powerful multihead or multitape Turing machines. Similar remarks apply to random access machines.

Definition 1. $\mathcal{P}$ is the class of languages recognizable by one-tape Turing machines which operate in polynomial time.

Definition 2. Π is the class of functions from Σ^* into Σ^* defined by one-tape Turing machines which operate in polynomial time.

The reader will not go wrong by identifying $\mathcal{P}$ with the class of languages recognizable by digital computers (with unbounded backup storage) which operate in polynomial time and Π with the class of string mappings performed in polynomial time by such computers.

Remark. If $f: \Sigma^* \to \Sigma^*$ is in Π then there is a polynomial $p(\cdot)$ such that $\lg(f(x)) \le p(\lg(x))$.

We next introduce a concept of reducibility which is of central importance in this paper.

Definition 3. Let L and M be languages. Then $L \propto M$ (L is reducible to M) if there is a function $f \in \Pi$ such that $f(x) \in M \Leftrightarrow x \in L$.

Lemma 1. If $L \propto M$ and $M \in \mathcal{P}$ then $L \in \mathcal{P}$.

Proof. The following is a polynomial-time bounded algorithm to decide if $x \in L$: compute $f(x)$; then test in polynomial time whether $f(x) \in M$.

We will be interested in the difficulty of recognizing subsets of countable domains other than Σ^*. Given such a domain D, there is usually a natural one-one encoding $e: D \to \Sigma^*$. For example we can represent a positive integer by the string of 0's and 1's comprising its binary representation, a 1-dimensional integer array as a list of integers, a matrix as a list of 1-dimensional arrays, etc.; and there are standard techniques for encoding lists into strings over a finite alphabet, and strings over an arbitrary finite alphabet as strings of 0's and 1's. Given such an encoding $e: D \to \Sigma^*$, we say that a set $T \subseteq D$ is <u>recognizable in polynomial time</u> if $e(T) \in \mathcal{P}$. Also, given sets $T \subseteq D$ and $U \subseteq D'$, and encoding functions $e: D \to \Sigma^*$ and $e': D' \to \Sigma^*$ we say $T \propto U$ if $e(T) \propto e'(U)$.

As a rule several natural encodings of a given domain are possible. For instance a graph can be represented by its adjacency matrix, by its incidence matrix, or by a list of unordered pairs of nodes, corresponding to the arcs. Given one of these representations, there remain a number of arbitrary decisions as to format and punctuation. Fortunately, it is almost always obvious that any two "reasonable" encodings e_o and e_1 of a given problem are equivalent; i.e., $e_o(S) \in \mathcal{P} \Leftrightarrow e_1(S) \in \mathcal{P}$. One important exception concerns the representation of positive integers; we stipulate that a positive integer is encoded in a binary, rather than unary, representation. In view of the invariance of recognizability in polynomial time and reducibility under reasonable encodings, we discuss problems in terms of their original domains, without specifying an encoding into Σ^*.

We complete this section by listing a sampling of problems which are solvable in polynomial time. In the next section we examine a number of close relatives of these problems which are not known to be solvable in polynomial time. Appendix 1 establishes our notation.

Each problem is specified by giving (under the heading "INPUT") a generic element of its domain of definition and (under the heading "PROPERTY") the property which causes an input to be accepted.

SATISFIABILITY WITH AT MOST 2 LITERALS PER CLAUSE [Cook (1971)]
INPUT: Clauses $C_1, C_2, \ldots, C_p$, each containing at most 2 literals
PROPERTY: The conjunction of the given clauses is satisfiable; i.e., there is a set $S \subseteq \{x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$ such that
a) S does not contain a complementary pair of literals and
b) $S \cap C_k \neq \phi$, $k = 1,2,\ldots,p$.

MINIMUM SPANNING TREE [Kruskal (1956)]
INPUT: G, w, W
PROPERTY: There exists a spanning tree of weight $\le W$.

SHORTEST PATH [Dijkstra (1959)]
INPUT: G, w, W, s, t
PROPERTY: There is a path between s and t of weight $\le W$.

MINIMUM CUT [Edmonds & Karp (1972)]
INPUT: G, w, W, s, t
PROPERTY: There is an s,t cut of weight $\le W$.

ARC COVER [Edmonds (1965)]
INPUT: G, k
PROPERTY: There is a set $Y \subseteq A$ such that $|Y| \le k$ and every node is incident with an arc in Y.

ARC DELETION
INPUT: G, k
PROPERTY: There is a set of k arcs whose deletion breaks all cycles.

BIPARTITE MATCHING [Hall (1948)]
INPUT: $S \subseteq Z_p \times Z_p$
PROPERTY: There are p elements of S, no two of which are equal in either component.

SEQUENCING WITH DEADLINES
INPUT: $(T_1,\ldots,T_n) \in Z^n$, $(D_1,\ldots,D_n) \in Z^n$, k
PROPERTY: Starting at time 0, one can execute jobs $1,2,\ldots,n$, with execution times T_i and deadlines D_i, in some order such that not more than k jobs miss their deadlines.

SOLVABILITY OF LINEAR EQUATIONS
INPUT: (c_{ij}), (a_i)
PROPERTY: There exists a vector (y_j) such that, for each i, $\sum_j c_{ij} y_j = a_i$.

3. NONDETERMINISTIC ALGORITHMS AND COOK'S THEOREM

In this section we state an important theorem due to Cook (1971) which asserts that any language in a certain wide class NP is reducible to a specific set S, which corresponds to the problem of deciding whether a boolean formula in conjunctive normal form is satisfiable.

Let $\mathcal{P}^{(2)}$ denote the class of subsets of $\Sigma^* \times \Sigma^*$ which are recognizable in polynomial time. Given $L^{(2)} \in \mathcal{P}^{(2)}$ and a polynomial p, we define a language L as follows:

$$L = \{x \mid \text{there exists } y \text{ such that } \langle x,y \rangle \in L^{(2)} \text{ and } lg(y) \le p(lg(x))\}.$$

We refer to L as the language derived from $L^{(2)}$ by p-bounded existential quantification.

Definition 4. NP is the set of languages derived from elements of $P^{(2)}$ by polynomial-bounded existential quantification.

There is an alternative characterization of NP in terms of nondeterministic Turing machines. A nondeterministic recognition algorithm A is specified by:

a countable set D (the domain)
a finite alphabet Δ such that $\Delta^* \cap \{\text{ACCEPT},\text{REJECT}\} = \phi$
an encoding function $E: D \to \Delta^*$
a transition relation $\tau \subseteq \Delta^* \times (\Delta^* \cup \{\text{ACCEPT},\text{REJECT}\})$

such that, for every $y_o \in \Delta^*$, the set $\{\langle y_o,y\rangle \mid \langle y_o,y\rangle \in \tau\}$ has fewer than k_A elements, where k_A is a constant. A computation of A on input $x \in D$ is a sequence $y_1,y_2,\ldots$ such that $y_1 = E(x)$, $\langle y_i,y_{i+1}\rangle \in \tau$ for all i, and, if the sequence is finite and ends with y_k, then $y_k \in \{\text{ACCEPT},\text{REJECT}\}$. A string $y \in \Delta^*$ which occurs in some computation is an instantaneous description. A finite computation ending in ACCEPT is an accepting computation. Input x is accepted if there is an accepting computation for x. If $D = \Sigma^*$ then A is a nondeterministic string recognition algorithm and we say that A operates in polynomial time if there is a polynomial $p(\cdot)$ such that, whenever A accepts x, there is an accepting computation for x of length $\leq p(\lg(x))$.

A nondeterministic algorithm can be regarded as a process which, when confronted with a choice between (say) two alternatives, can create two copies of itself, and follow up the consequences of both courses of action. Repeated splitting may lead to an exponentially growing number of copies; the input is accepted if any sequence of choices leads to acceptance.

The nondeterministic 1-tape Turing machines, multitape Turing machines, random-access machines, etc. define classes of nondeterministic string recognition algorithms by restricting the encoding function E and transition relation τ to particularly simple forms. All these classes of algorithms, restricted to operate in polynomial time, define the same class of languages. Moreover, this class is NP.

Theorem 1. $L \in NP$ if and only if L is accepted by a nondeterministic Turing machine which operates in polynomial time.

Proof. $\Rightarrow$ Suppose $L \in NP$. Then, for some $L^{(2)} \in P^{(2)}$ and some polynomial p, L is obtained from $L^{(2)}$ by p-bounded existential quantification. We can construct a nondeterministic machine which first guesses the successive digits of a string y of length $\leq p(\lg(y))$ and then tests whether $\langle x,y\rangle \in L^{(2)}$. Such a machine clearly recognizes L in polynomial time.

$\Leftarrow$ Suppose L is accepted by a nondeterministic Turing machine T which operates in time p. Assume without loss of generality that, for any instantaneous description Z, there are at most two instantaneous descriptions that may follow Z (i.e., at most two primitive transitions are applicable). Then the sequence of choices of instantaneous descriptions made by T in a given computation can be encoded as a string y of 0's and 1's, such that $\lg(y) \leq p(\lg(x))$.

Thus we can construct a deterministic Turing machine T', with $\Sigma^* \times \Sigma^*$ as its domain of inputs, which, on input $\langle x,y\rangle$, simulates the action of T on input x with the sequence of choices y. Clearly T' operates in polynomial time, and L is obtained by polynomial bounded existential quantification from the set of pairs of strings accepted by T'.

The class NP is very extensive. Loosely, a recognition problem is in NP if and only if it can be solved by a backtrack search of polynomial bounded depth. A wide range of important computational problems which are not known to be in P are obviously in NP. For example, consider the problem of determining whether the nodes of a graph G can be colored with k colors so that no two adjacent nodes have the same color. A nondeterministic algorithm can simply guess an assignment of colors to the nodes and then check (in polynomial time) whether all pairs of adjacent nodes have distinct colors.

In view of the wide extent of NP, the following theorem due to Cook is remarkable. We define the satisfiability problem as follows:

SATISFIABILITY
INPUT: Clauses $C_1,C_2,\ldots,C_p$
PROPERTY: The conjunction of the given clauses is satisfiable; i.e., there is a set $S \subseteq \{x_1,x_2,\ldots,x_n;\bar{x}_1,\bar{x}_2,\ldots,\bar{x}_n\}$ such that
a) S does not contain a complementary pair of literals
and b) $S \cap C_k \neq \phi$, $k = 1,2,\ldots,p$.

Theorem 2 (Cook). If $L \in NP$ then $L \propto$ SATISFIABILITY.

The theorem stated by Cook (1971) uses a weaker notion of reducibility than the one used here, but Cook's proof supports the present statement.

Corollary 1. $P = NP \Leftrightarrow$ SATISFIABILITY $\in P$.

Proof. If SATISFIABILITY $\in P$ then, for each $L \in NP$, $L \in P$, since $L \propto$ SATISFIABILITY. If SATISFIABILITY $\notin P$, then, since clearly SATISFIABILITY $\in NP$, $P \neq NP$.

Remark. If $P = NP$ then NP is closed under complementation and polynomial-bounded existential quantification. Hence it is also closed under polynomial-bounded universal quantification. It follows that a polynomial-bounded analogue of Kleene's Arithmetic Hierarchy [Rogers (1967)] becomes trivial if $P = NP$.

Theorem 2 shows that, if there were a polynomial-time algorithm to decide membership in SATISFIABILITY then every problem solvable by a polynomial-depth backtrack search would also be solvable by a polynomial-time algorithm. This is strong circumstantial evidence that SATISFIABILITY $\notin P$.

4. COMPLETE PROBLEMS

The main object of this paper is to establish that a large number of important computational problems can play the role of SATISFIABILITY in Cook's theorem. Such problems will be called complete.

Definition 5. The language L is (polynomial) complete if
a) $L \in NP$
and b) SATISFIABILITY $\propto L$.

Theorem 3. Either all complete languages are in P, or none of them are. The former alternative holds if and only if $P = NP$.

We can extend the concept of completeness to problems defined over countable domains other than Σ^*.

Definition 6. Let D be a countable domain, e a "standard" one-one encoding $e: D \to \Sigma^*$ and T a subset of D. Then T is complete if and only if $e(D)$ is complete.

Lemma 2. Let D and D' be countable domains, with one-one encoding functions e and e'. Let $T \subseteq D$ and $T' \subseteq D'$. Then $T \propto T'$ if there is a function $F: D \to D'$ such that
a) $F(x) \in T' \Leftrightarrow x \in T$
and b) there is a function $f \in \Pi$ such that $f(x) = e'(F(e^{-1}(x)))$ whenever $e'(F(e^{-1}(x)))$ is defined.

The rest of the paper is mainly devoted to the proof of the following theorem.

Origin of NP-Completeness!

Main Theorem. All the problems on the following list are complete.

1. SATISFIABILITY
COMMENT: By duality, this problem is equivalent to determining whether a disjunctive normal form expression is a tautology.

2. 0-1 INTEGER PROGRAMMING
INPUT: integer matrix C and integer vector d
PROPERTY: There exists a 0-1 vector x such that $Cx = d$.

3. CLIQUE
INPUT: graph G, positive integer k
PROPERTY: G has a set of k mutually adjacent nodes.

4. SET PACKING
INPUT: Family of sets $\{S_j\}$, positive integer ℓ
PROPERTY: $\{S_j\}$ contains ℓ mutually disjoint sets.

5. NODE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: There is a set $R \subseteq N'$ such that $|R| \leq \ell$ and every arc is incident with some node in R.

6. SET COVERING
INPUT: finite family of finite sets $\{S_j\}$, positive integer k
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ containing $\leq k$ sets such that $\cup T_h = \cup S_j$.

7. FEEDBACK NODE SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $R \subseteq V$ such that every (directed) cycle of H contains a node in R.

8. FEEDBACK ARC SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $S \subseteq E$ such that every (directed) cycle of H contains an arc in S.

9. DIRECTED HAMILTON CIRCUIT
INPUT: digraph H
PROPERTY: H has a directed cycle which includes each node exactly once.

10. UNDIRECTED HAMILTON CIRCUIT
INPUT: graph G
PROPERTY: G has a cycle which includes each node exactly once.

11. SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
INPUT: Clauses $D_1,D_2,\ldots,D_r$, each consisting of at most 3 literals from the set $\{u_1,u_2,\ldots,u_m\} \cup \{\bar{u}_1,\bar{u}_2,\ldots,\bar{u}_m\}$
PROPERTY: The set $\{D_1,D_2,\ldots,D_r\}$ is satisfiable.

12. CHROMATIC NUMBER
INPUT: graph G, positive integer k
PROPERTY: There is a function $\phi: N \to Z_k$ such that, if u and v are adjacent, then $\phi(u) \neq \phi(v)$.

13. CLIQUE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: N' is the union of ℓ or fewer cliques.

14. EXACT COVER
INPUT: family $\{S_j\}$ of subsets of a set $\{u_i, i = 1,2,\ldots,t\}$
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that the sets T_h are disjoint and $\cup T_h = \cup S_j = \{u_i, i = 1,2,\ldots,t\}$.

15. HITTING SET
INPUT: family $\{U_i\}$ of subsets of $\{s_j, j = 1,2,\ldots,r\}$
PROPERTY: There is a set W such that, for each i, $|W \cap U_i| = 1$.

16. STEINER TREE
INPUT: graph G, $R \subseteq N$, weighting function $w: A \to Z$, positive integer k
PROPERTY: G has a subtree of weight $\leq k$ containing the set of nodes in R.

17. 3-DIMENSIONAL MATCHING
INPUT: set $U \subseteq T \times T \times T$, where T is a finite set
PROPERTY: There is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate.

18. KNAPSACK
INPUT: $(a_1,a_2,\ldots,a_r,b) \in Z^{n+1}$
PROPERTY: $\Sigma a_j x_j = b$ has a 0-1 solution.

19. JOB SEQUENCING
INPUT: "execution time vector" $(T_1,\ldots,T_p) \in Z^p$,
"deadline vector" $(D_1,\ldots,D_p) \in Z^p$
"penalty vector" $(P_1,\ldots,P_p) \in Z^p$
positive integer k
PROPERTY: There is a permutation π of $\{1,2,\ldots,p\}$ such that

$$\left(\sum_{j=1}^{p} [\text{if } T_{\pi(1)}+\cdots+T_{\pi(j)} > D_{\pi(j)} \text{ then } P_{\pi(j)} \text{ else } 0]\right) \leq k \quad .$$

20. PARTITION
INPUT: $(c_1,c_2,\ldots,c_s) \in Z^s$
PROPERTY: There is a set $I \subseteq \{1,2,\ldots,s\}$ such that $\sum_{h\in I} c_h = \sum_{h\notin I} c_h$.

21. MAX CUT
INPUT: graph G, weighting function $w: A \to Z$, positive integer W
PROPERTY: There is a set $S \subseteq N$ such that

$$\sum_{\substack{\{u,v\}\in A\\ u\in S\\ v\notin S}} w(\{u,v\}) \geq W \quad .$$

It is clear that these problems (or, more precisely, their encodings into Σ^*), are all in NP. We proceed to give a series of explicit reductions, showing that SATISFIABILITY is reducible to each of the problems listed. Figure 1 shows the structure of the set of reductions. Each line in the figure indicates a reduction of the upper problem to the lower one.

To exhibit a reduction of a set $T \subseteq D$ to a set $T' \subseteq D'$, we specify a function $F: D \to D'$ which satisfies the conditions of Lemma 2. In each case, the reader should have little difficulty in verifying that F does satisfy these conditions.

SATISFIABILITY $\propto$ 0-1 INTEGER PROGRAMMING

$$c_{ij} = \begin{cases} 1 & \text{if } x_j \in C_i \\ -1 & \text{if } \bar{x}_j \in C_i \\ 0 & \text{otherwise} \end{cases} \qquad \begin{matrix} i = 1,2,\ldots,p \\ j = 1,2,\ldots,n \end{matrix}$$

$b_i = 1 -$ (the number of complemented variables in C_i) ,
$i = 1,2,\ldots,p$.

SATISFIABILITY $\propto$ CLIQUE

$N = \{\langle\sigma,i\rangle \mid \sigma$ is a literal and occurs in $C_i\}$

$A = \{\{\langle\sigma,i\rangle,\langle\delta,j\rangle\} \mid i \neq j$ and $\sigma \neq \bar{\delta}\}$

$k = p$, the number of clauses.

CLIQUE $\propto$ SET PACKING

Assume $N = \{1,2,\ldots,n\}$. The elements of the sets $S_1,S_2,\ldots,S_n$ are those two-element sets of nodes $\{i,j\}$ not in A.

$S_i = \{\{i,j\} \mid \{i,j\} \notin A\}$, $i = 1,2,\ldots,n$

$\ell = k$.

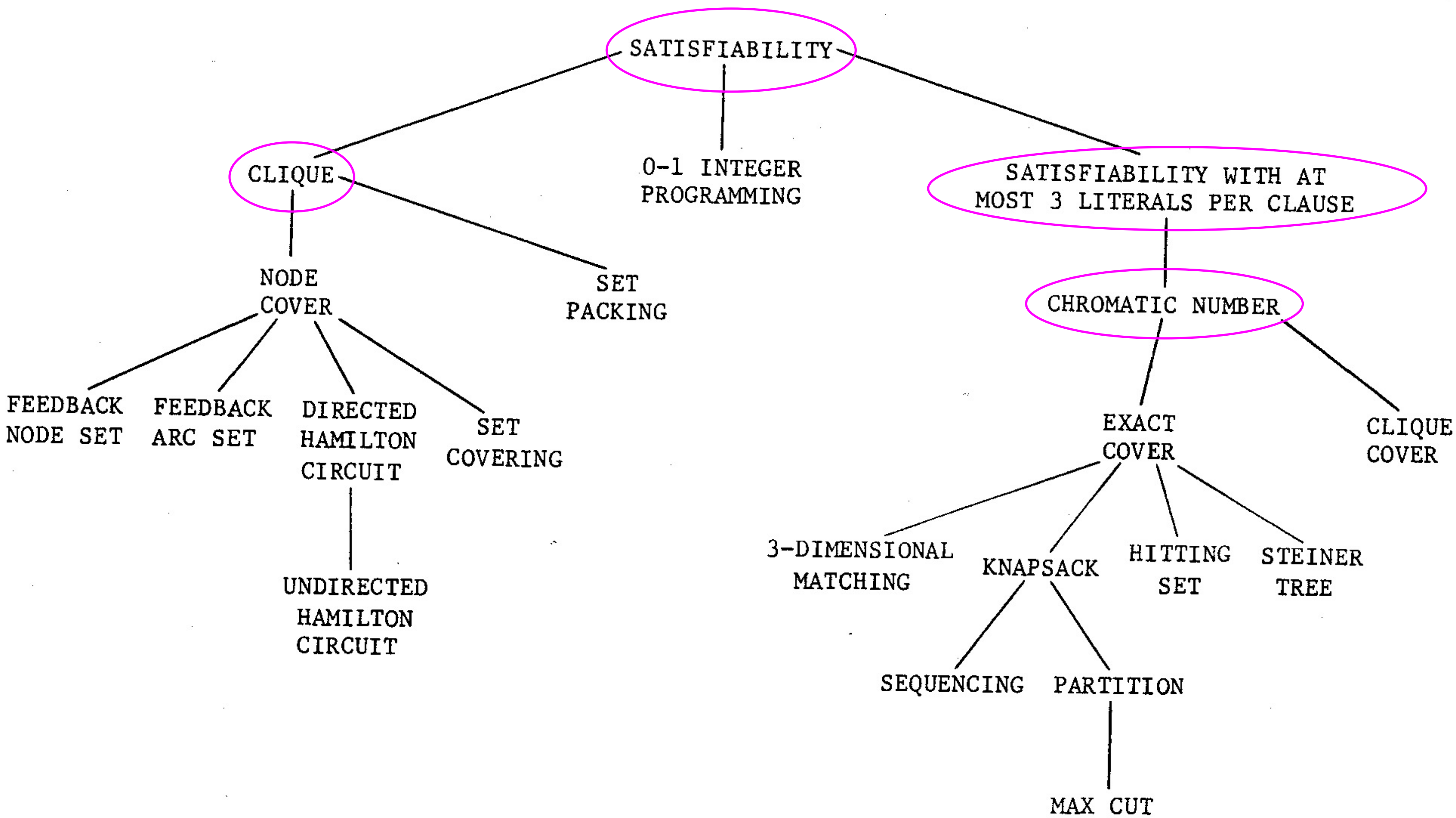


FIGURE 1 - Complete Problems

CLIQUE $\propto$ NODE COVER

G' is the complement of G.
$\ell = |N| - k$

NODE COVER $\propto$ SET COVERING

Assume $N' = \{1,2,\ldots,n\}$. The elements are the arcs of G'. S_j is the set of arcs incident with node j. $k = \ell$.

NODE COVER $\propto$ FEEDBACK NODE SET

$V = N'$
$E = \{\langle u,v\rangle \mid \{u,v\} \in A'\}$
$k = \ell$

NODE COVER $\propto$ FEEDBACK ARC SET

$V = N' \times \{0,1\}$
$E = \{\langle\langle u,0\rangle,\langle u,1\rangle\rangle \mid u \in N'\} \cup \{\langle\langle u,1\rangle,\langle v,0\rangle\rangle \mid \{u,v\} \in A'\}$
$k = \ell$.

NODE COVER $\propto$ DIRECTED HAMILTON CIRCUIT

Without loss of generality assume $A' = Z_m$.

$V = \{a_1,a_2,\ldots,a_\ell\} \cup \{\langle u,i,\alpha\rangle \mid u \in N'$ is incident with $i \in A'$ and $\alpha \in \{0,1\}\}$

$E = \{\langle\langle u,i,0\rangle,\langle u,i,1\rangle\rangle \mid \langle u,i,0\rangle \in V\}$
$\cup \{\langle\langle u,i,\alpha\rangle,\langle v,i,\alpha\rangle\rangle \mid i \in A'$, u and v are incident with i, $\alpha \in \{0,1\}\}$
$\cup \{\langle\langle u,i,1\rangle,\langle u,j,0\rangle\rangle \mid u$ is incident with i and j and $\nexists h$, $i < h < j$, such that u is incident with $h\}$
$\cup \{\langle\langle u,i,1\rangle,a_f\rangle \mid 1 \le f \le \ell$ and $\nexists h > i$ such that u is incident with $h\}$
$\cup \{\langle a_f,\langle u,i,0\rangle\rangle \mid 1 \le f \le \ell$ and $\nexists h < i$ such that u is incident with $h\}$.

DIRECTED HAMILTON CIRCUIT $\propto$ UNDIRECTED HAMILTON CIRCUIT

$N = V \times \{0,1,2\}$
$A = \{\{\langle u,0\rangle,\langle u,1\rangle\},\{\langle u,1\rangle,\langle u,2\rangle\} \mid u \in V\}$
$\cup \{\{\langle u,2\rangle,\langle v,0\rangle\} \mid \langle u,v\rangle \in E\}$

SATISFIABILITY $\propto$ SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE

Replace a clause $\sigma_1 \cup \sigma_2 \cup \cdots \cup \sigma_m$, where the σ_i are literals and $m > 3$, by

$$(\sigma_1 \cup \sigma_2 \cup u_1)(\sigma_3 \cup \cdots \cup \sigma_m \cup \bar{u}_1)(\bar{\sigma}_3 \cup u_1)\cdots(\bar{\sigma}_m \cup u_1),$$

where u_1 is a new variable. Repeat this transformation until no clause has more than three literals.

SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE $\propto$ CHROMATIC NUMBER

Assume without loss of generality that $m \geq 4$.

$N = \{u_1,u_2,\ldots,u_m\} \cup \{\bar{u}_1,\bar{u}_2,\ldots,\bar{u}_m\} \cup \{v_1,v_2,\ldots,v_m\}$
$\cup \{D_1,D_2,\ldots,D_r\}$
$A = \{\{u_i,\bar{u}_i\} \mid i=1,2,\ldots,n\} \cup \{\{v_i,v_j\} \mid i \ne j\} \cup \{\{v_i,x_j\} \mid i \ne j\}$
$\cup \{\{v_i,\bar{x}_j\} \mid i \ne j\} \cup \{\{u_i,D_f\} \mid u_i \notin D_f\} \cup \{\{\bar{u}_i,D_f\} \mid \bar{u}_i \in D_f\}$
$k = r + 1$

CHROMATIC NUMBER $\propto$ CLIQUE COVER

G' is the complement of G
$\ell = k$.

CHROMATIC NUMBER $\propto$ EXACT COVER

The set of elements is

$N \cup A \cup \{\langle u,e,f\rangle \mid u$ is incident with e and $1 \le f \le k\}$.

The sets S_j are the following:

for each f, $1 \le f \le k$, and each $u \in N$,
$\{u\} \cup \{\langle u,e,f\rangle \mid e$ is incident with $u\}$;

for each $e \in A$ and each pair f_1, f_2 such that
$1 \le f_1 \le k$, $1 \le f_2 \le k$ and $f_1 \ne f_2$
$\{e\} \cup \{\langle u,e,f\rangle, f \ne f_1\} \cup \{\langle v,e,g\rangle \mid g \ne f_2\}$,

where u and v are the two nodes incident with e.

EXACT COVER $\propto$ HITTING SET

The hitting set problem has sets U_i and elements s_j, such that $s_j \in U_i \Leftrightarrow u_i \in S_j$.

EXACT COVER $\propto$ STEINER TREE

$N = \{n_o\} \cup \{S_j\} \cup \{u_i\}$
$R = \{n_o\} \cup \{u_i\}$
$A = \{\{n_o,S_j\}\} \cup \{\{S_j,u_i\} \mid u_i \in S_j\}$
$w(\{n_o,S_j\}) = |S_j|$
$w(\{S_j,u_i\}) = 0$
$k = |\{u_i\}|$

EXACT COVER $\propto$ 3-DIMENSIONAL MATCHING

Without loss of generality assume $|S_j| \geq 2$ for each j. Let $T = \{\langle i,j\rangle \mid u_i \in S_j\}$. Let α be an arbitrary one-one function from $\{u_i\}$ into T. Let $\pi: T \to T$ be a permutation such that, for each fixed j, $\{\langle i,j\rangle \mid u_i \in S_j\}$ is a cycle of π.

$$U = \{\langle\alpha(u_i),\langle i,j\rangle,\langle i,j\rangle\rangle \mid \langle i,j\rangle \in T\}$$
$$\cup \{\langle\beta,\sigma,\pi(\sigma)\rangle \mid \text{for all } i,\ \beta \neq \alpha(u_i)\} \quad .$$

EXACT COVER ∝ KNAPSACK

Let $d = |\{S_j\}| + 1$. Let $\epsilon_{ji} = \begin{cases} 1 & \text{if } u_i \in S_j \\ 0 & \text{if } u_i \notin S_j \end{cases}$. Let

$r = |\{S_j\}|$, $a_j = \sum \epsilon_{ji} d^{i-1}$ and $b = \frac{d^t - 1}{d - 1}$.

KNAPSACK ∝ SEQUENCING

$p = r$, $T_i = P_i = a_i$, $D_i = b$.

KNAPSACK ∝ PARTITION

$$s = r+2$$
$$c_i = a_i \ , \quad i = 1,2,\ldots,r$$
$$c_{r+1} = b+1$$
$$c_{r+2} = (\sum_{i=1}^{r} a_i) + 1 - b$$

PARTITION ∝ MAX CUT

$$N = \{1,2,\ldots,s\}$$
$$A = \{\{i,j\} \mid i \in N,\ j \in N,\ i \neq j\}$$
$$w(\{i,j\}) = c_i \cdot c_j$$
$$W = \lceil \tfrac{1}{4} \Sigma c_i^2 \rceil$$

Some of the reductions exhibited here did not originate with the present writer. Cook (1971) showed that SATISFIABILITY ∝ SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE. The reduction

SATISFIABILITY ∝ CLIQUE

is implicit in Cook (1970), and was also known to Raymond Reiter. The reduction

NODE COVER ∝ FEEDBACK NODE SET

was found by the Algorithms Seminar at the Cornell University Computer Science Department. The reduction

NODE COVER ∝ FEEDBACK ARC SET

was found by Lawler and the writer, and Lawler discovered the reduction

EXACT COVER ∝ 3-DIMENSIONAL MATCHING

The writer discovered that the exact cover problem was reducible to the directed traveling-salesman problem on a digraph in which the arcs have weight zero or one. Using refinements of the technique used in this construction, Tarjan showed that

EXACT COVER ∝ DIRECTED HAMILTON CIRCUIT

and, independently, Lawler showed that

NODE COVER ∝ DIRECTED HAMILTON CIRCUIT .

The reduction

DIRECTED HAMILTON CIRCUIT ∝ UNDIRECTED HAMILTON CIRCUIT

was pointed out by Tarjan.

Below we list three problems in automata theory and language theory to which every complete problem is reducible. These problems are not known to be complete, since their membership in *NP* is presently in doubt. The reader unacquainted with automata and language theory can find the necessary definitions in Hopcroft and Ullman (1969).

PSPACE-complete

EQUIVALENCE OF REGULAR EXPRESSIONS
INPUT: A pair of regular expressions over the alphabet $\{0,1\}$
PROPERTY: The two expressions define the same language.

EQUIVALENCE OF NONDETERMINISTIC FINITE AUTOMATA
INPUT: A pair of nondeterministic finite automata with input alphabet $\{0,1\}$
PROPERTY: The two automata define the same language.

CONTEXT-SENSITIVE RECOGNITION
INPUT: A context-sensitive grammar Γ and a string x
PROPERTY: x is in the language generated by Γ.

Membership in NP is still open!

First we show that

SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
∝ EQUIVALENCE OF REGULAR EXPRESSIONS .

The reduction is made in two stages. In the first stage we construct a pair of regular expressions over an alphabet $\Delta = \{u_1, u_2, \ldots, u_n, \bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n\}$. We then convert these regular expressions to regular expressions over $\{0,1\}$.

The first regular expression is $\Delta^n \Delta^*$ (more exactly, Δ is written out as $(u_1+u_2+\cdots+u_n+\bar{u}_1+\cdots+\bar{u}_n)$, and Δ^n represents n copies of the expression for Δ concatenated together). The second regular expression is

$$\Delta^n\Delta^* \cup \bigcup_{i=1}^{n} (\Delta^* u_i \Delta^* \bar{u}_i \Delta^* \cup \Delta^* \bar{u}_i \Delta^* u_i \Delta^*) \cup \bigcup_{h=1}^{r} \theta(D_h)$$

where

$$\theta(D_h) = \begin{cases} \Delta^*\bar{\sigma}_1\Delta^* & \text{if } D_h = \sigma_1 \\ \Delta^*\bar{\sigma}_1\Delta^*\bar{\sigma}_2\Delta^* \cup \Delta^*\bar{\sigma}_2\Delta^*\bar{\sigma}_1\Delta^* & \text{if } D_h = \sigma_1 \cup \sigma_2 \\ \Delta^*\bar{\sigma}_1\Delta^*\bar{\sigma}_2\Delta^*\bar{\sigma}_3\Delta^* \cup \Delta^*\bar{\sigma}_1\Delta^*\bar{\sigma}_3\Delta^*\bar{\sigma}_2\Delta^* & \\ \quad \cup\, \Delta^*\bar{\sigma}_2\Delta^*\bar{\sigma}_1\Delta^*\bar{\sigma}_3\Delta^* \cup \Delta^*\bar{\sigma}_2\Delta^*\bar{\sigma}_3\Delta^*\bar{\sigma}_1\Delta^* & \\ \quad \cup\, \Delta^*\bar{\sigma}_3\Delta^*\bar{\sigma}_1\Delta^*\bar{\sigma}_2\Delta^* \cup \Delta^*\bar{\sigma}_3\Delta^*\bar{\sigma}_2\Delta^*\bar{\sigma}_1\Delta^* & \\ & \text{if } D_h = \sigma_1 \cup \sigma_2 \cup \sigma_3 \,. \end{cases}$$

Now let m be the least positive integer $\geq \log_2|\Delta|$, and let ϕ be a 1-1 function from Δ into $\{0,1\}^m$. Replace each regular expression by a regular expression over $\{0,1\}$, by making the substitution $a \to \phi(a)$ for each occurrence of each element of Δ.

EQUIVALENCE OF REGULAR EXPRESSIONS $\propto$ EQUIVALENCE OF NONDETERMINISTIC FINITE AUTOMATA

There are standard polynomial-time algorithms [Salomaa (1969)] to convert a regular expression to an equivalent nondeterministic automaton. Finally, we show that, for any $L \in NP$,

$$L \propto \text{CONTEXT-SENSITIVE RECOGNITION} \quad .$$

Suppose L is recognized in time $p(\ )$ by a nondeterministic Turing machine. Then the following language $\tilde{L}$ over the alphabet $\{0,1,\#\}$ is accepted by a nondeterministic linear bounded automaton which simulates the Turing machine:

$$\tilde{L} = \{\#^{p(\lg(x))}x\#^{p(\lg(x))} \mid x \in L\} \quad .$$

Hence $\tilde{L}$ is context-sensitive and has a context-sensitive grammar $\tilde{\Gamma}$. Thus $x \in L$ iff

$$\tilde{\Gamma}, \#^{p(\lg(x))}x\#^{p(\lg(x))}$$

is an acceptable input to CONTEXT-SENSITIVE RECOGNITION.

We conclude by listing the following important problems in NP which are not known to be complete.

GRAPH ISOMORPHISM
INPUT: graphs G and G'
PROPERTY: G is isomorphic to G'.

Still open!

NONPRIMES
INPUT: positive integer k
PROPERTY: k is composite.

$\in$P [2004]

LINEAR INEQUALITIES
INPUT: integer matrix C, integer vector d
PROPERTY: $Cx \geq d$ has a rational solution.

APPENDIX I

Notation and Terminology Used in Problem Specification

PROPOSITIONAL CALCULUS

$x_1,x_2,\ldots,x_n$	$u_1,u_2,\ldots,u_m$	propositional variables
$\bar{x}_1,\bar{x}_2,\ldots,\bar{x}_n$	$\bar{u}_1,\bar{u}_2,\ldots,\bar{u}_m$	complements of propositional variables
σ,σ_i		literals
$C_1,C_2,\ldots,C_p$	$D_1,D_2,\ldots,D_r$	clauses
$C_k \subseteq \{x_1,x_2,\ldots,x_n,\bar{x}_1,\bar{x}_2,\ldots,\bar{x}_n\}$		
$D_\ell \subseteq \{u_1,u_2,\ldots,u_m,\bar{u}_1,\bar{u}_2,\ldots,\bar{u}_m\}$		

A clause contains no complementary pair of literals.

SCALARS, VECTORS, MATRICES

Z	the positive integers
Z^p	the set of p-tuples of positive integers
Z_p	the set $\{0,1,\ldots,p-1\}$
k,W	elements of Z
$\langle x,y\rangle$	the ordered pair $\langle x,y\rangle$
(a_i) (y_j) d	vectors with nonnegative integer components
(c_{ij}) C	matrices with integer components

GRAPHS AND DIGRAPHS

$G = (N,A)$ $G' = (N',A')$ finite graphs
N,N' sets of nodes A,A' sets of arcs
s,t,u,v nodes $e,\{u,v\}$ arcs
$(X,\bar{X}) = \{\{u,v\} \mid u \in X \text{ and } v \in \bar{X}\}$ cut

If $s \in X$ and $t \in \bar{X}$, $(X,\bar{X})$ is a s-t cut.

$w: A \to Z$ $w': A' \to Z$ weight functions

The weight of a subgraph is the sum of the weights of its arcs.

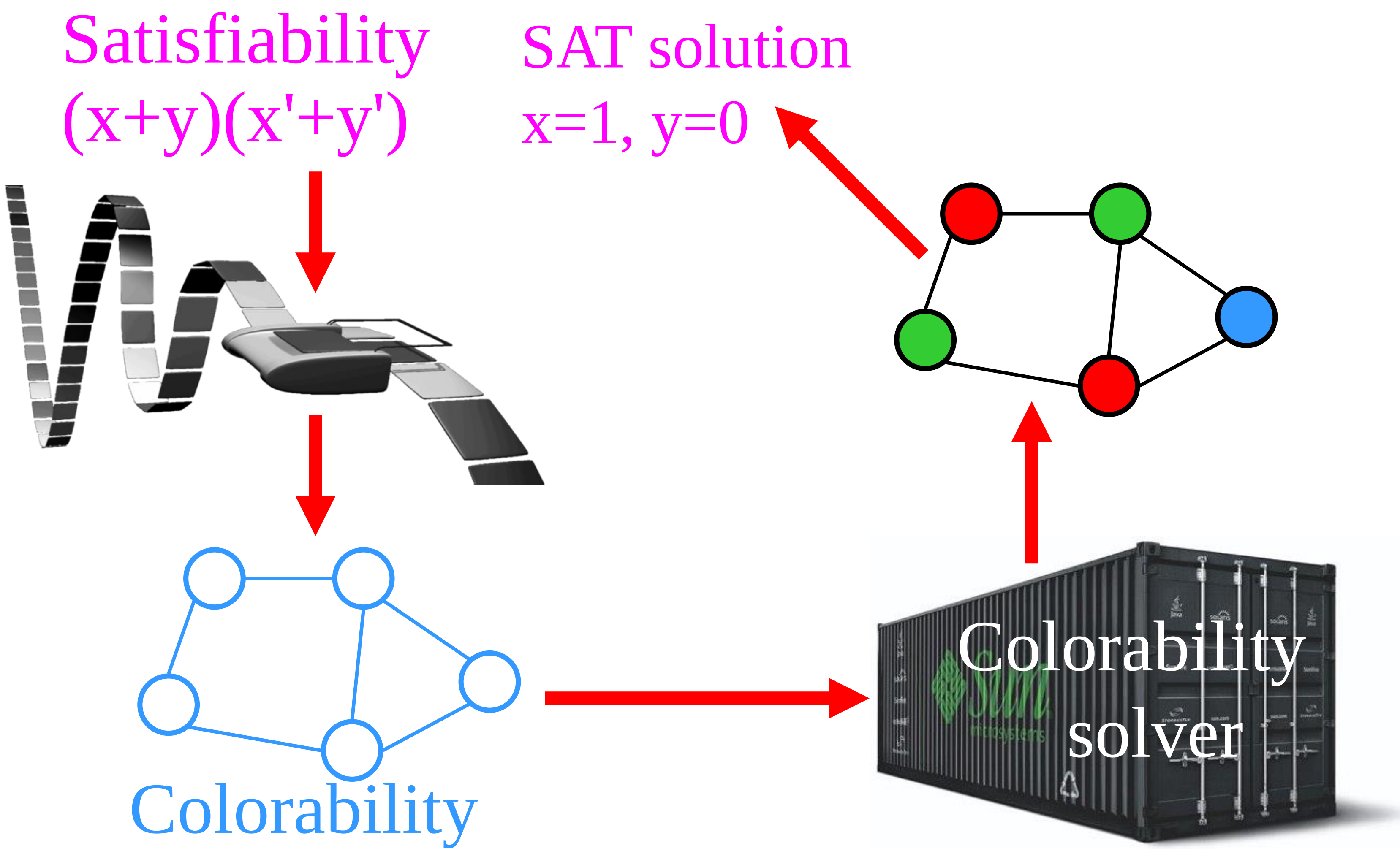
$H = (V,E)$ digraph V set of nodes, E set of arcs
$e,\langle u,v\rangle$ arcs

SETS

ϕ	the empty set
$\lvert S\rvert$	the number of elements in the finite set S
$\{S_j\}$ $\{T_h\}$ $\{U_i\}$	finite families of finite sets

Problem Transformations

Idea: To solve a problem, efficiently transform to another problem, and then use a solver for the other problem:



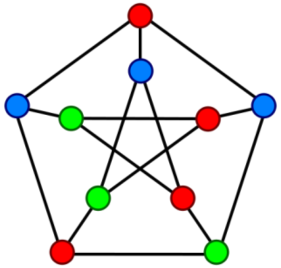
Decision vs. Optimization Problems

Decision problem: "yes" or "no" membership answer.

Ex: Given a Boolean formula, is it satisfiable?

$(x+y+z)$
$\wedge(x'+y'+z)$
$\wedge(x'+y+z')$

Ex: Given a graph, is it 3-colorable?



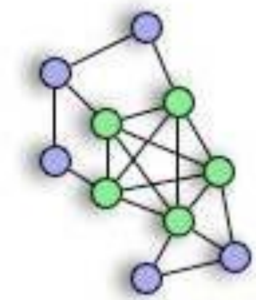
Ex: Given a graph & k, does it contain a k-clique?

Optimization problem: find a (minimal) solution.

Ex: Given a formula, find a satisfying assignment.

Ex: Given a graph, find a 3-coloring.

Ex: Given a graph & k, find a k-qlique.



Theorem: Solving a decision problem is not harder than solving its optimization version.

Theorem: Solving an optimization problem is not (more than polynomially) harder than solving its decision version.

Decision vs. Optimization Problems

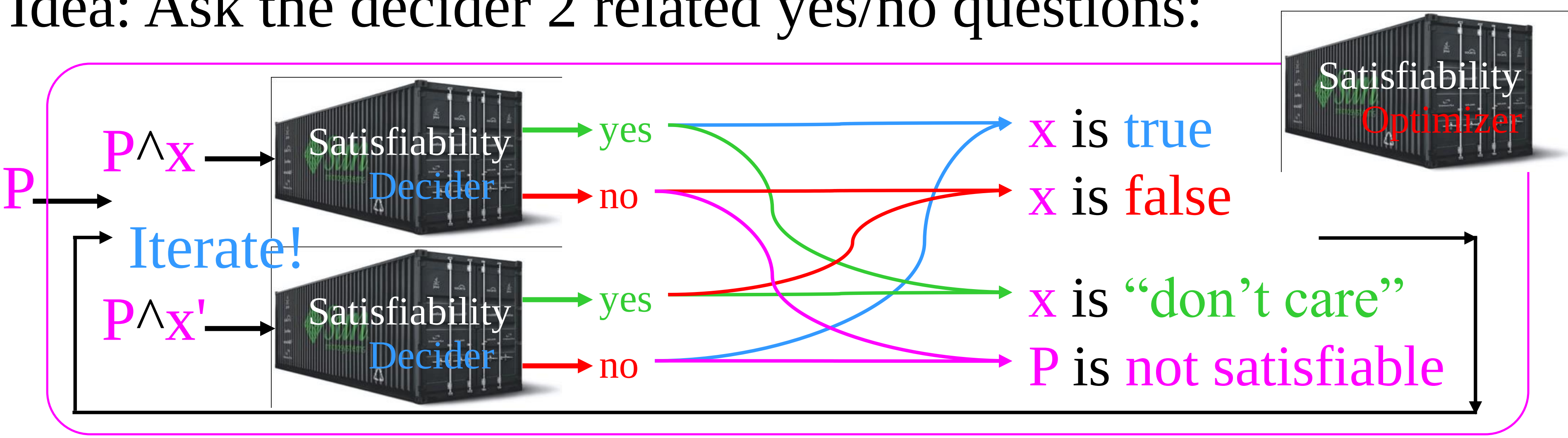
Corollary: A decision problem is in P if and only if its optimization version is in P.

Corollary: A decision problem is in NP if and only if its optimization version is in NP.

Building an optimizer from a decider:

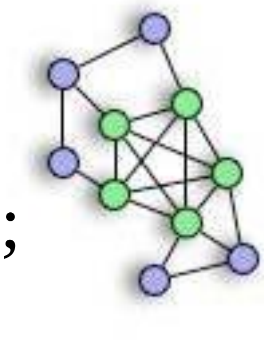
Ex: what is a satisfying assignemnt of $P=(x+y+z)(x'+y'+z)(x'+y+z')$?

Idea: Ask the decider 2 related yes/no questions:



Graph Cliques

Graph clique problem: given a graph and an integer k, is there a subgraph in G that is a complete graph of size k?



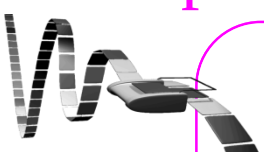
Theorem: The clique problem is NP-complete.

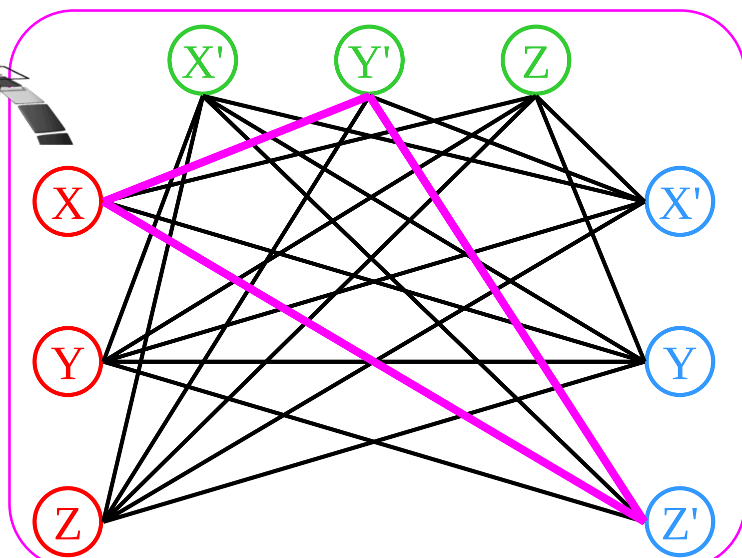
Proof: Reduction from 3-SAT:

Literals become nodes; k clauses induce node groups;

Connect all inter-group compatible nodes / literals.

Example: (x+y+z)(x'+y'+z)(x'+y+z')





k-clique corresponds to 3-SAT solution:
x = true, y = false, z = false

Clique is in NP $\Rightarrow$ clique is NP-complete.

Independent Sets

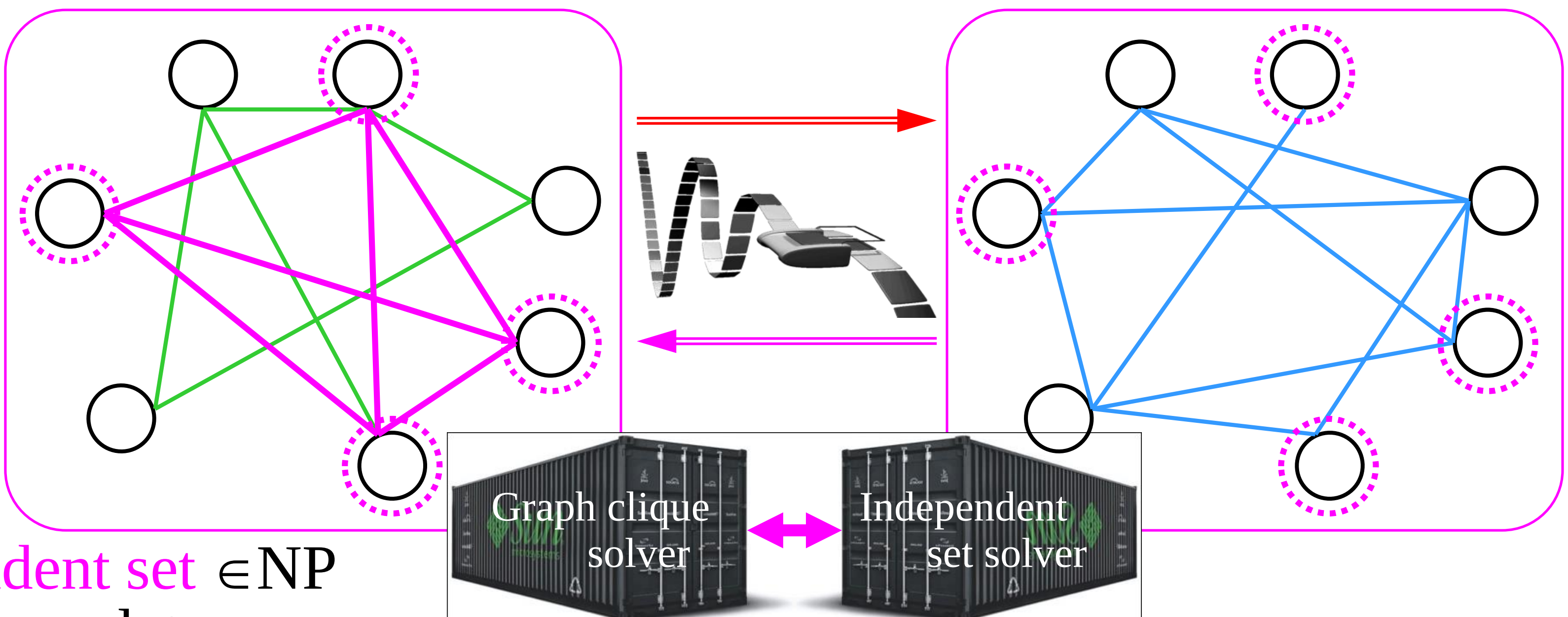
Independent set problem: given a graph and an integer k, is there a pairwise non-adjacent node subset of size k?

Theorem: The independent set problem is NP-complete.

Proof: Reduction from graph clique:

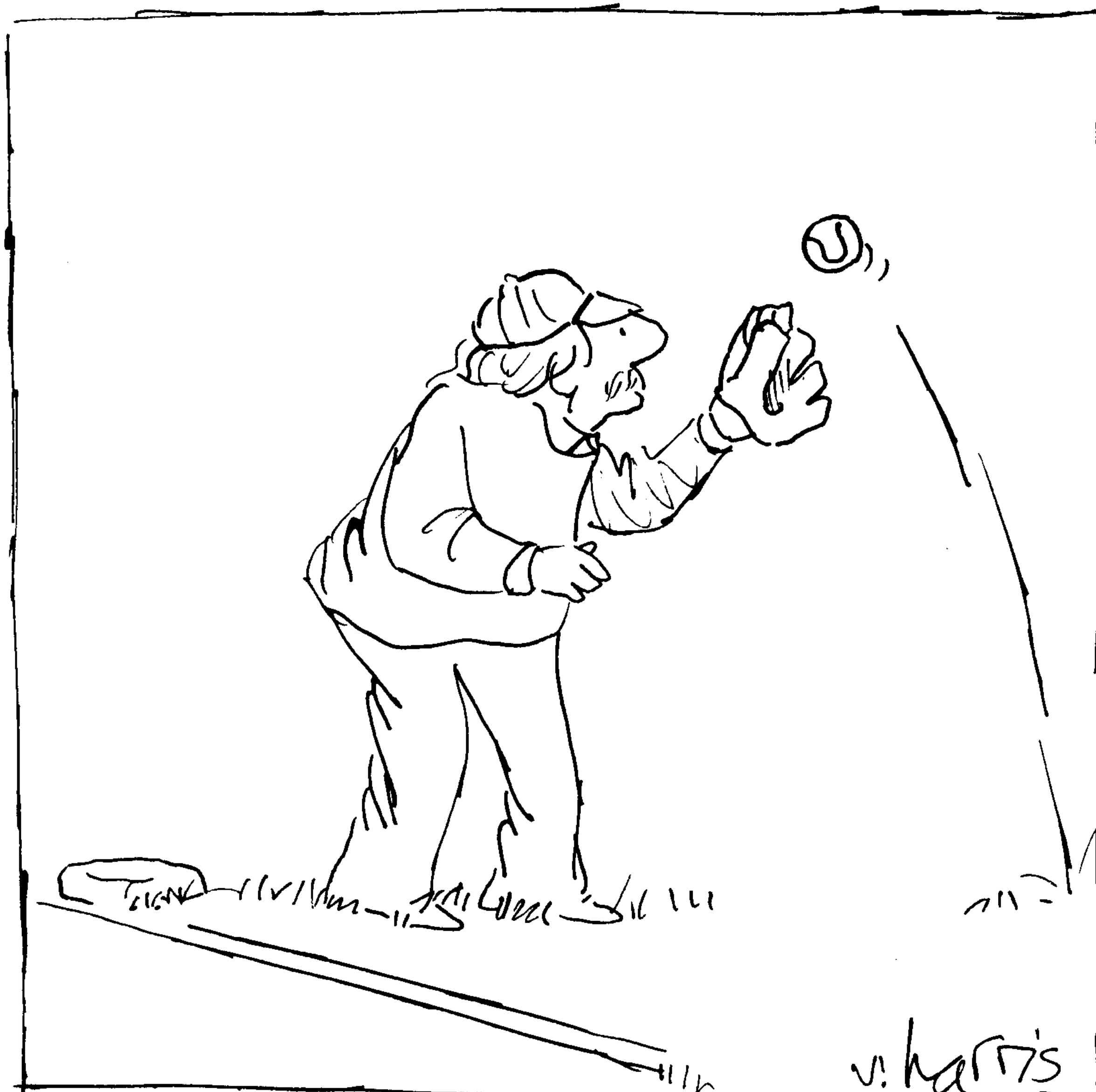
Idea: independent set is an "anti-clique" (i.e., negated clique)

$\Rightarrow$finding a clique reduces to finding an independent set in the complement graph:



Independent set $\in$NP

$\Rightarrow$ NP-complete.



AS SMART AS HE WAS, ALBERT EINSTEIN COULD NOT FIGURE OUT HOW TO HANDLE THOSE TRICKY BOUNCES AT THIRD BASE.

Graph Colorability

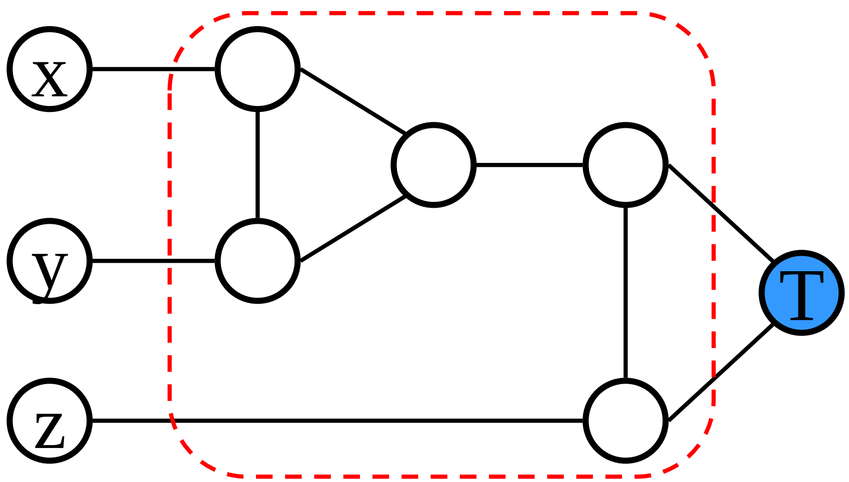
Problem: is a given graph G 3-colorable?

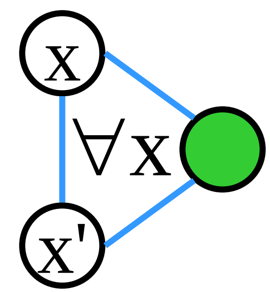
Theorem: Graph 3-colorability is NP-complete.

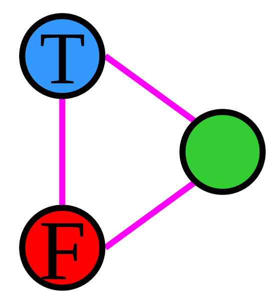
Proof: Reduction from 3-SAT.

Idea: construct a colorability “OR gate” “gadget”:

(x+y+z) $\Rightarrow$

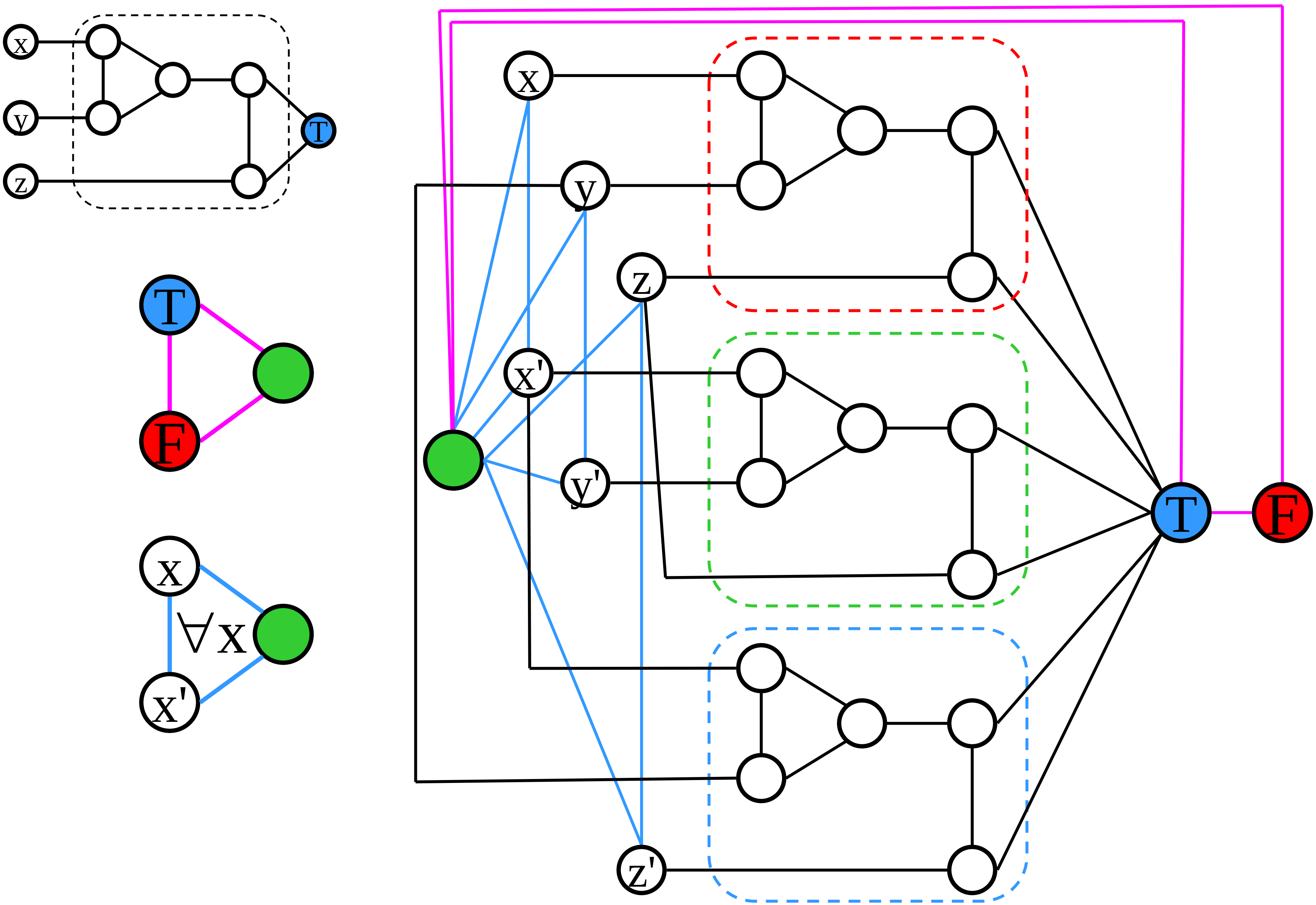






Property: gadget is 3-colorable iff
(x+y+z) is true

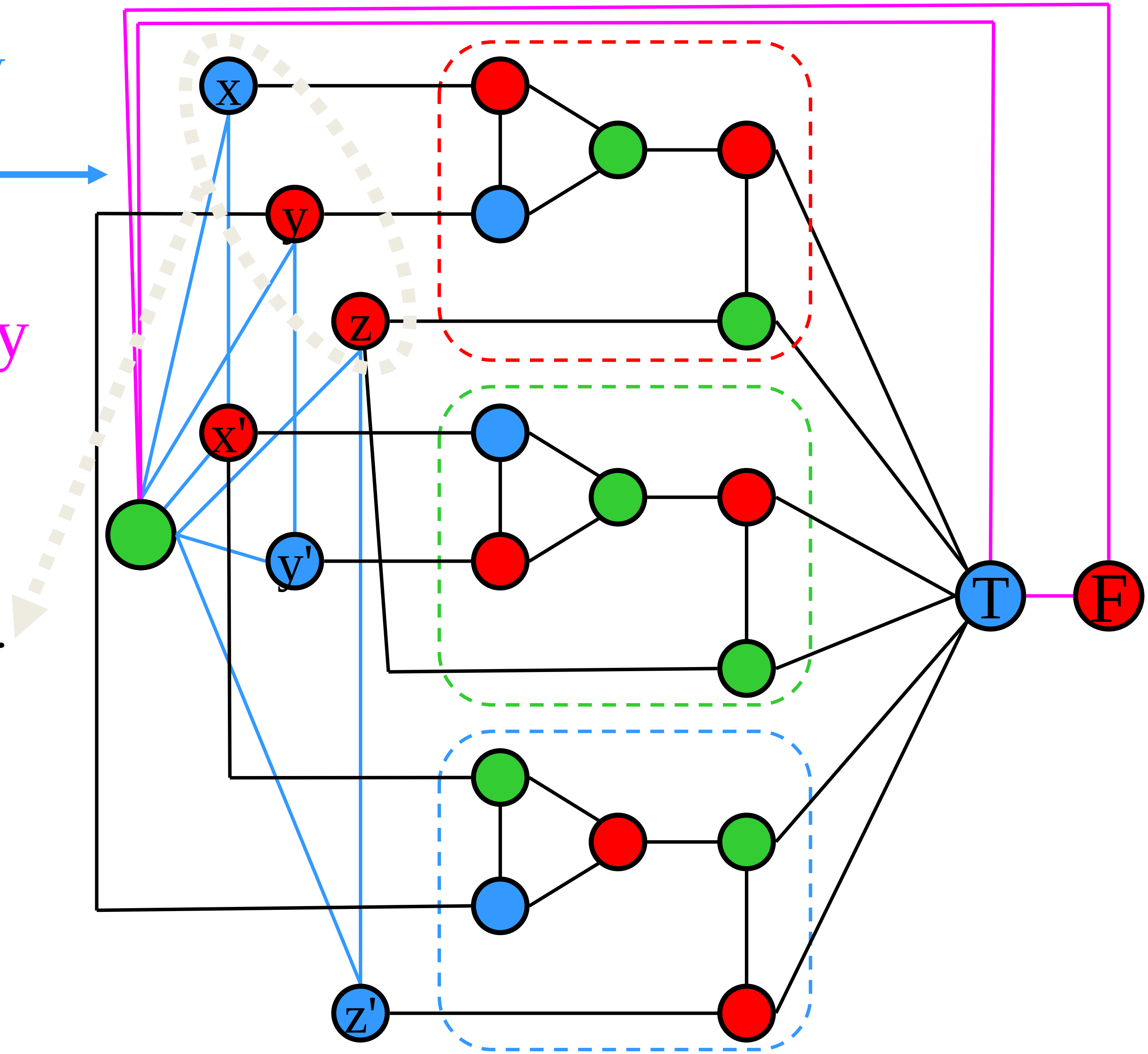
Example: (x+y+z)(x'+y'+z)(x'+y+z')



Example: (x+y+z)(x'+y'+z)(x'+y+z')

3-colorability

Solution:



3-satisfiability

Solution:

x = true

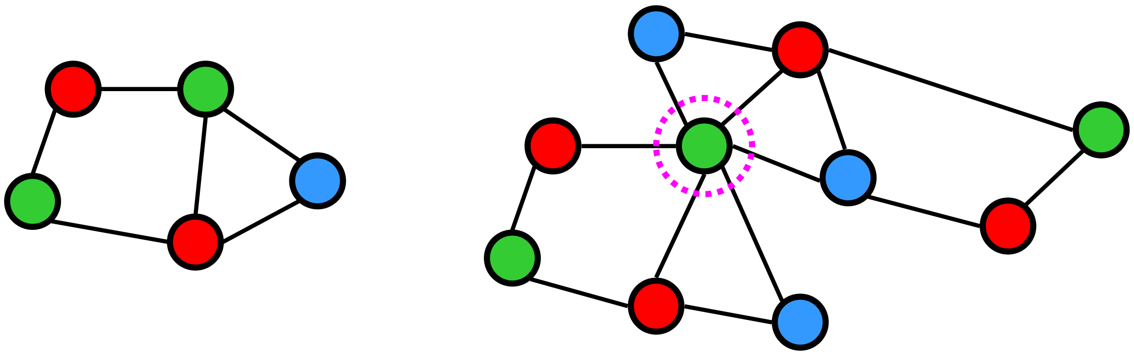
y = false

z = false

What Makes Colorability Difficult?

Q: Are high node degrees the reason that graph colorability is computationally difficult?

A: No!



Graph colorability is easy for max-degree-0 graphs

Graph colorability is easy for max-degree-1 graphs

Graph colorability is easy for max-degree-2 graphs

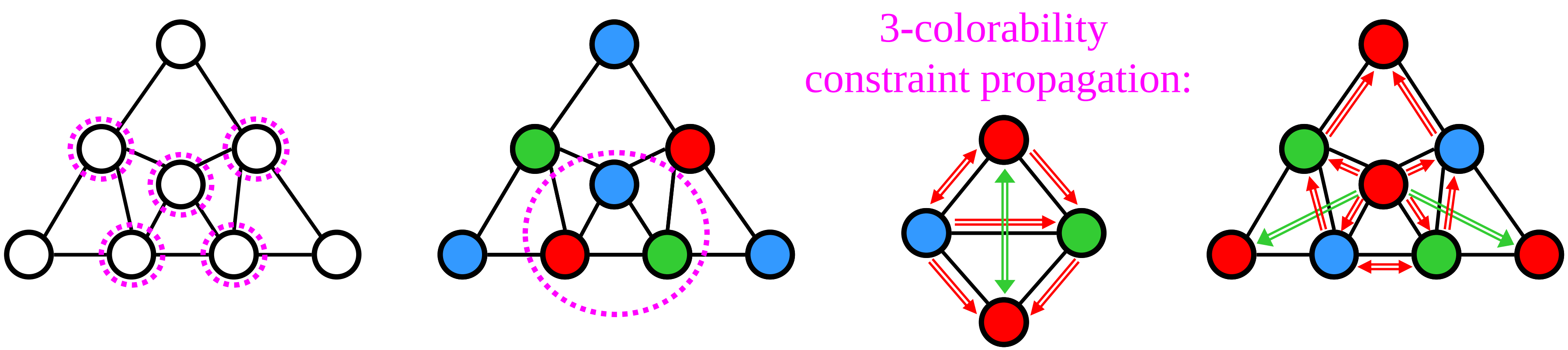
Why?

Theorem: Graph colorability is NP-complete for max-degree-4 graphs.

Restricted Graph Colorability

Theorem: Graph 3-colorability is NP-complete for max-degree-4 graphs.

Proof: Use “degree reduction” gadgets:



Gadget properties:

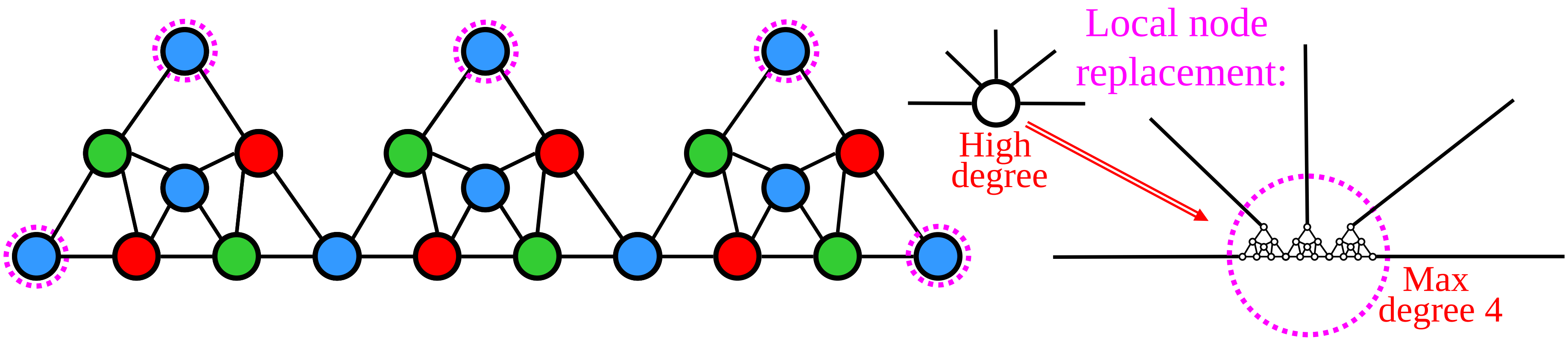
a) Gadget has max-degree of 4

b) Gadget is 3-colorable but not 2-colorable

c) In any 3-coloring all corners get the same color

Restricted Graph Colorability

Idea: combine gadgets into "super nodes"!



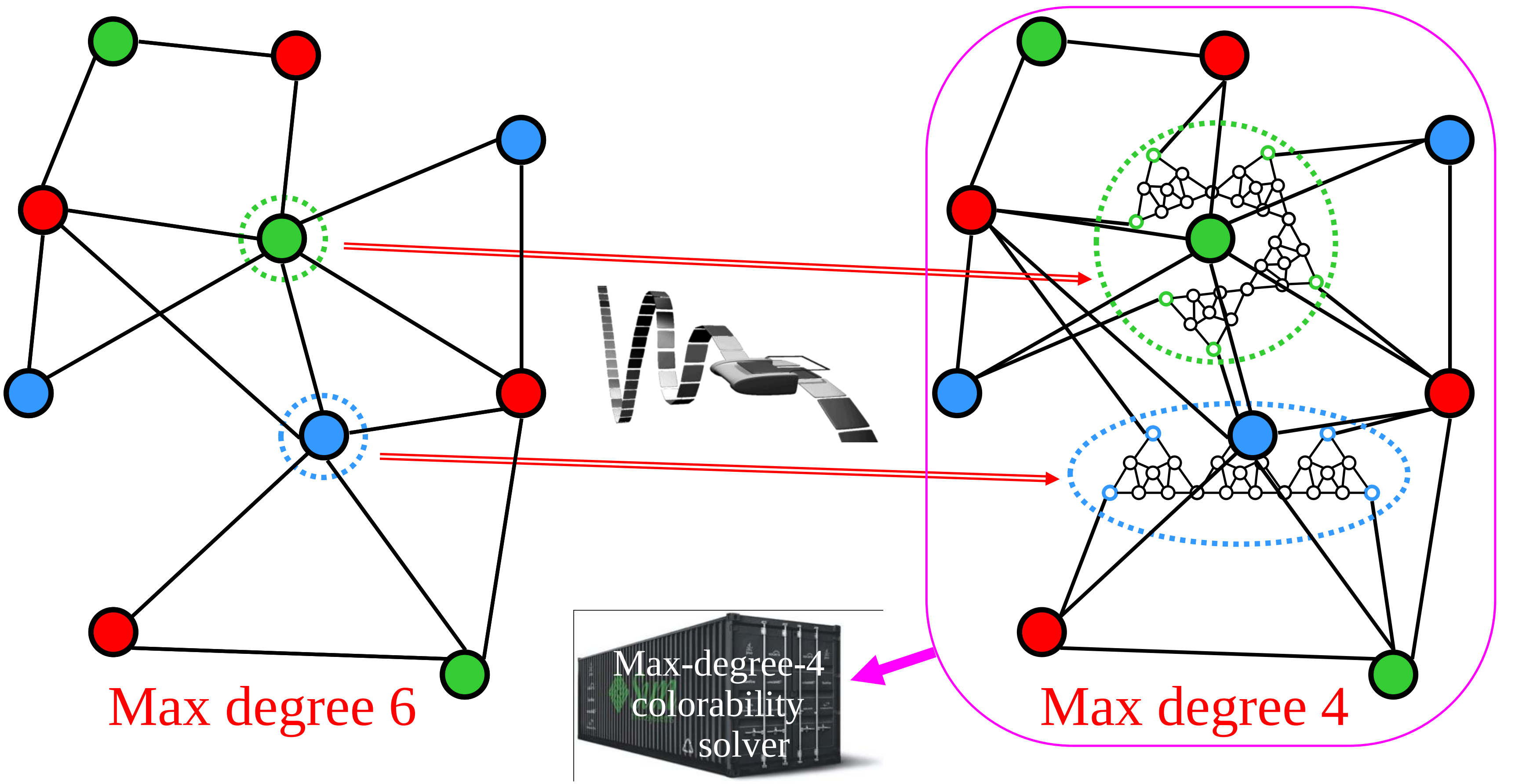
Properties (inherited from simple gadget):

a) Super-node has max-degree of 4
b) Super-node is 3-colorable but not 2-colorable
c) In any 3-coloring all "corners" get the same color

Idea: Use "super nodes" as "fan out" components to reduce all node degrees to 4 or less

Restricted Graph Colorability

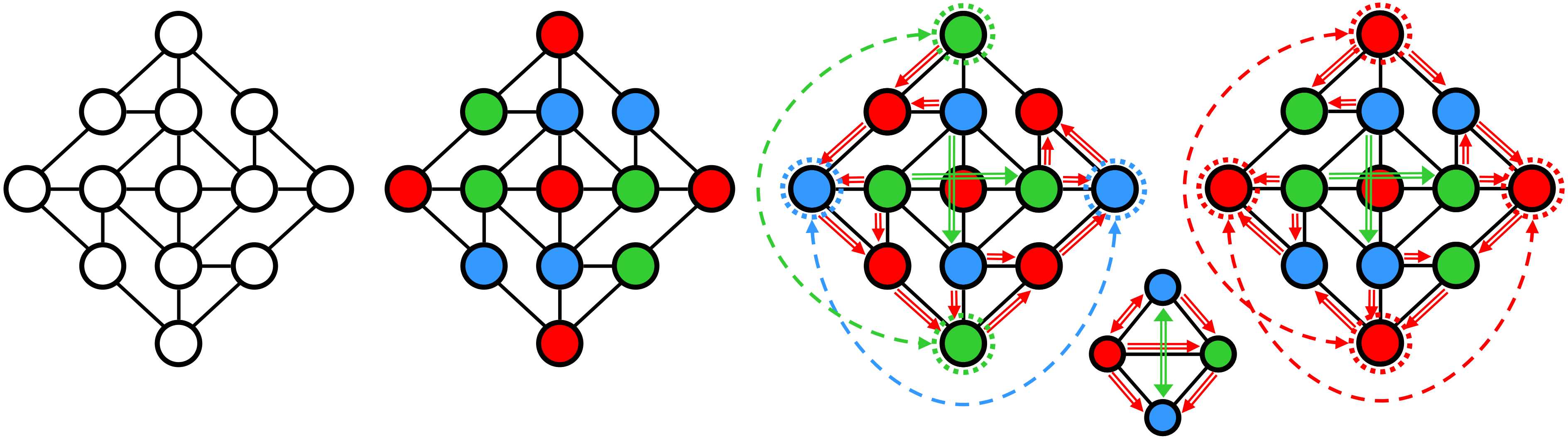
Example: convert high-degree to max-degree-4 graph



Conclusion: Solving max-degree-4 graph colorability is as difficult as solving general graph colorability!

Restricted Graph Colorability

Theorem: Planar graph 3-colorability is NP-complete.
Proof: Use "planarity preserving" gadgets:



Gadget properties:

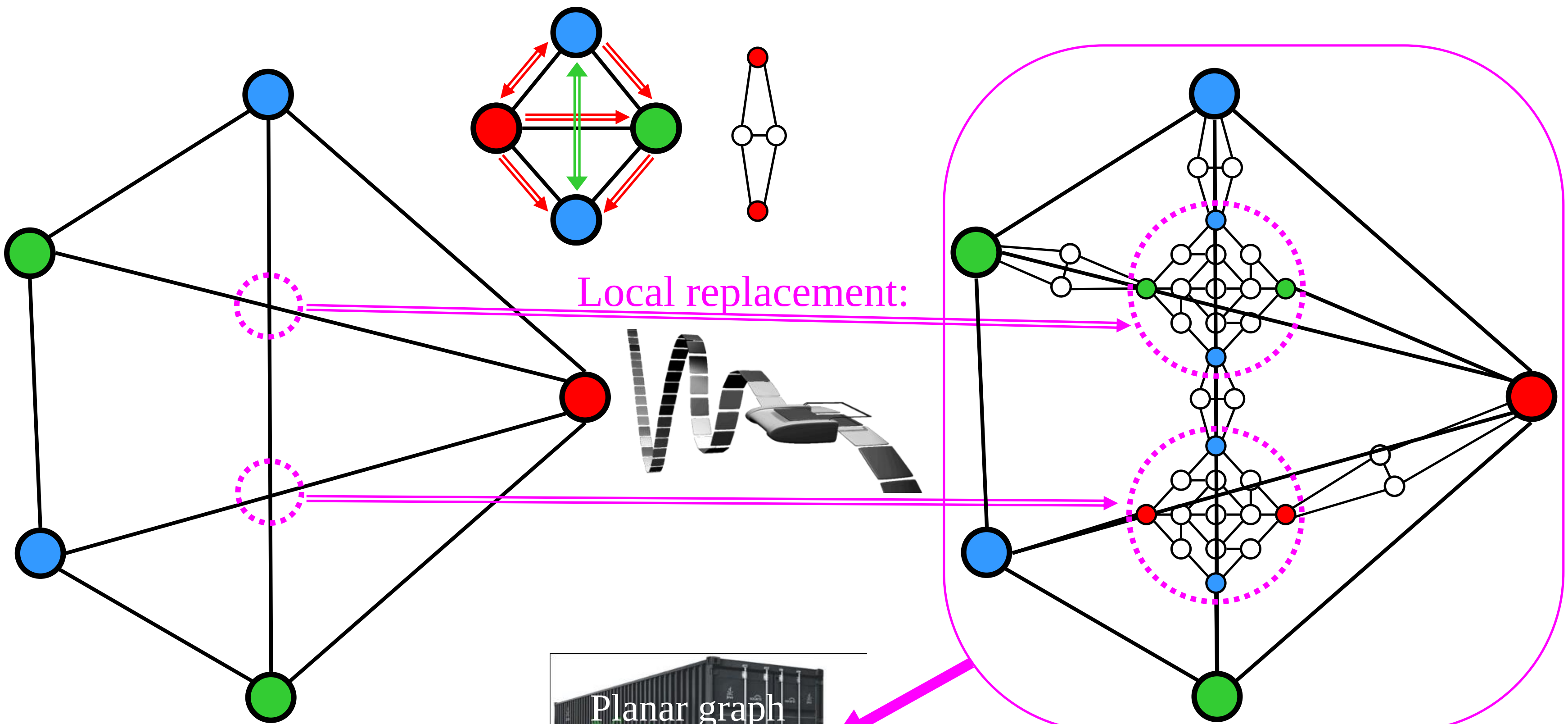
a) Gadget is planar and 3-colorable

b) In any 3-coloring opposite corners get same color

c) Pairs of opposite corners are "independent"

Restricted Graph Colorability

Idea: use gadgets to eliminate edge intersections!



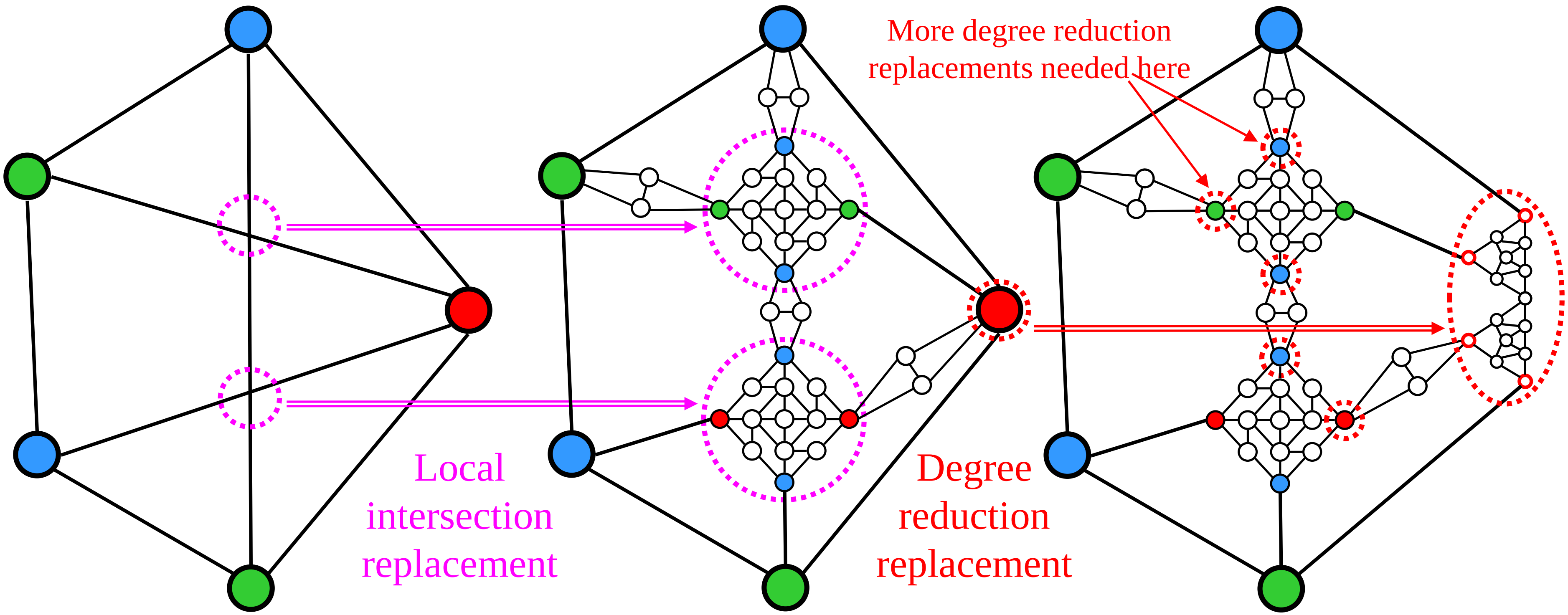
Planar graph is 3-colorable
IFF original graph was

Conclusion: Solving planar graph colorability
is as difficult as solving general graph colorability!

Restricted Graph Colorability

Theorem: Graph colorability is NP-complete for planar graphs with max degree 4.

Proof: Compose max-degree-4 transformation with planarity preserving transformation:



Resulting planar max-deg-4 graph is 3-colorable IFF original graph is!

Planar Graph Colorability

Theorem: Planar graph 1-colorability is trivial. DTIME(n)

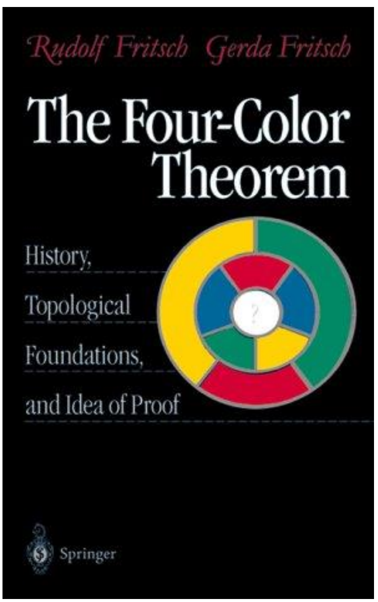
Theorem: Planar graph 2-colorability is easy. DTIME(n)

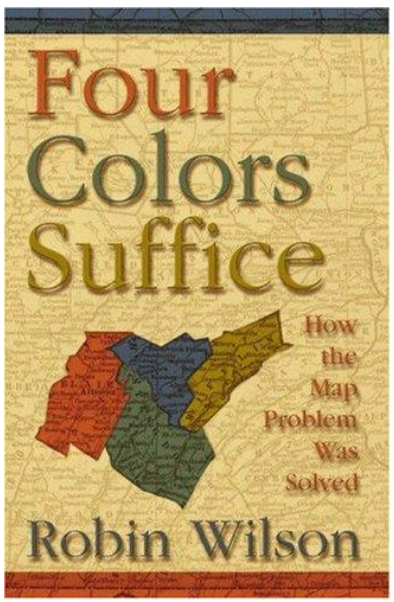
Theorem: Planar graph 3-colorability is NP-complete. Why?

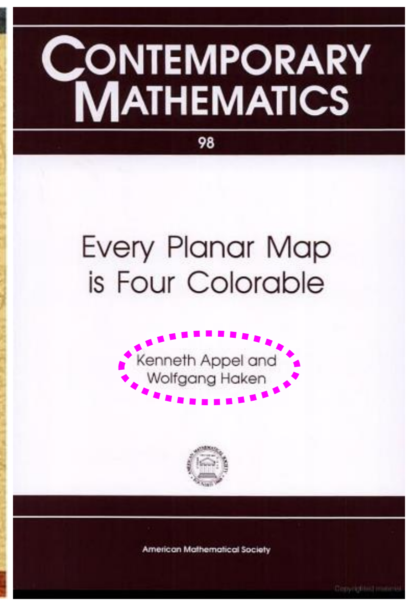
Theorem: Planar graph 4-colorability is trivial. DTIME(1)

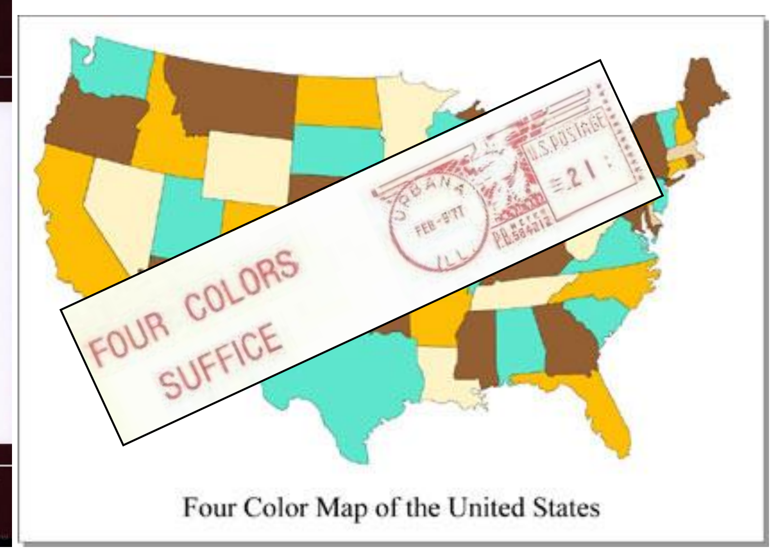
Theorem: All planar graphs have 4-colorings.

Open since 1852; solved by Appel & Haken in 1976 using long computer-assisted proof based on 1936 special cases!









Planar Graph Colorability

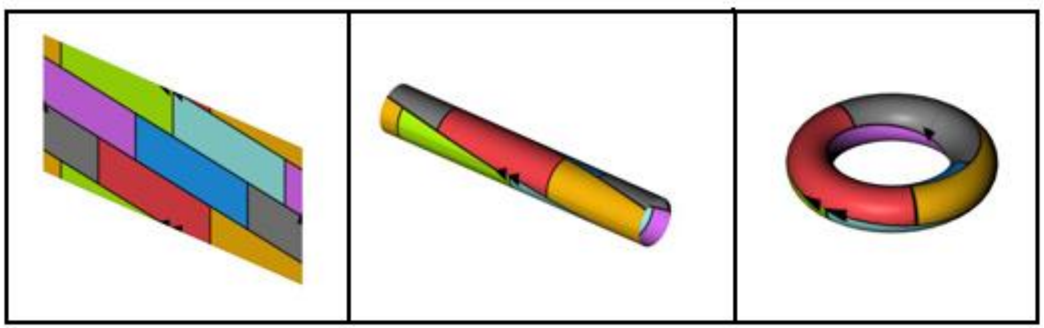
Theorem: Finding planar graph 4-coloring is in DTIME(n^2).

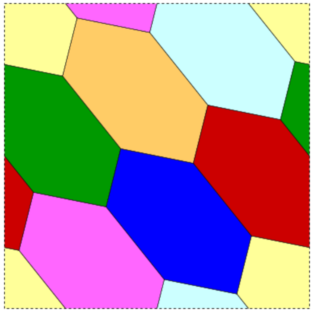
Theorem: Finding planar graph 5-coloring is in DTIME(n).

Theorem: Graph planarity testing is in DTIME(n).

Theorem: 4-coloring a 3-colorable graph is NP-hard.

Theorem: 7 colors are necessary and sufficient on a torus.





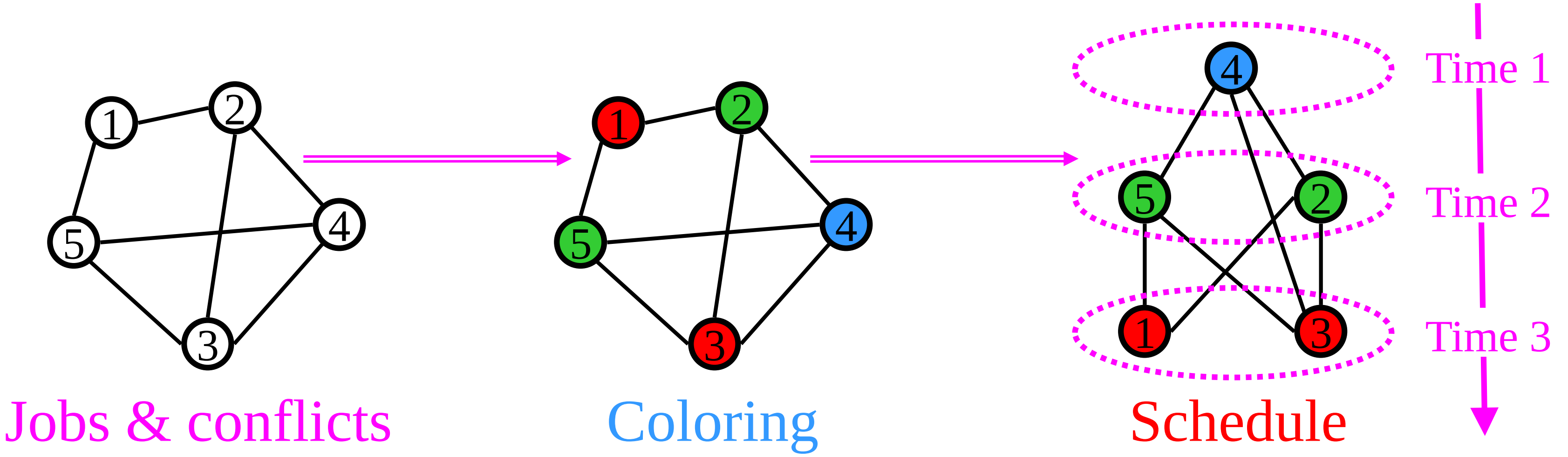
Theorem: For a surface of genus G, the number of colors that are both necessary and sufficient is $\left\lfloor \frac{7+\sqrt{1+48G}}{2} \right\rfloor$

Genus:	0	1	2	3	4	5	6	7	8
# colors:	4	7	8	9	10	11	12	12	13

Applications of Graph Coloring

Job scheduling:

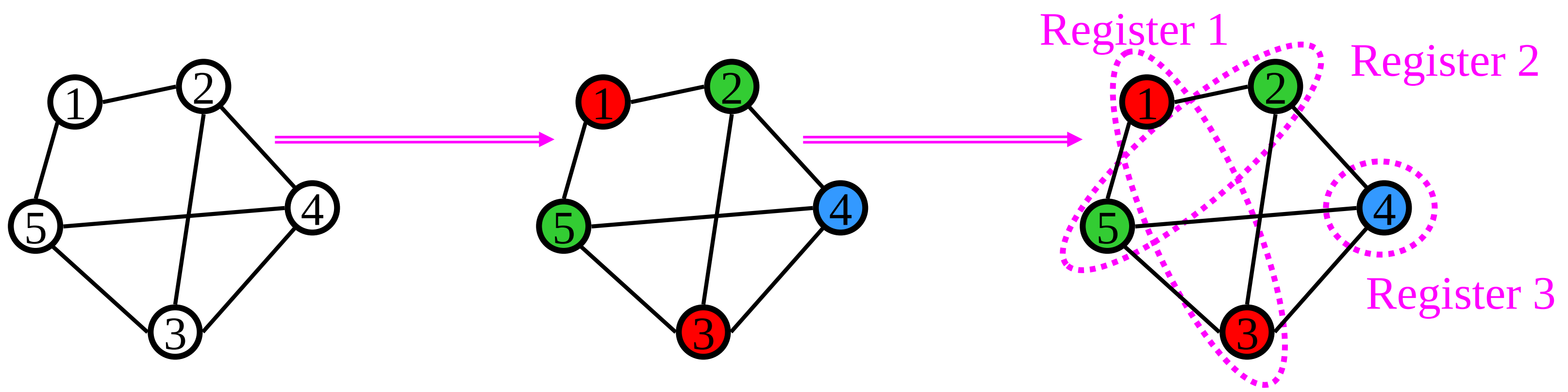
- Need to assign jobs to time slots;
- Some jobs conflict (e.g., use shared resource);
- Model jobs as nodes and conflicts as edges;
- Chromatic number is "minimum makespan" (optimal time to finish all jobs without conflict)



Applications of Graph Coloring

CPU Register allocation:

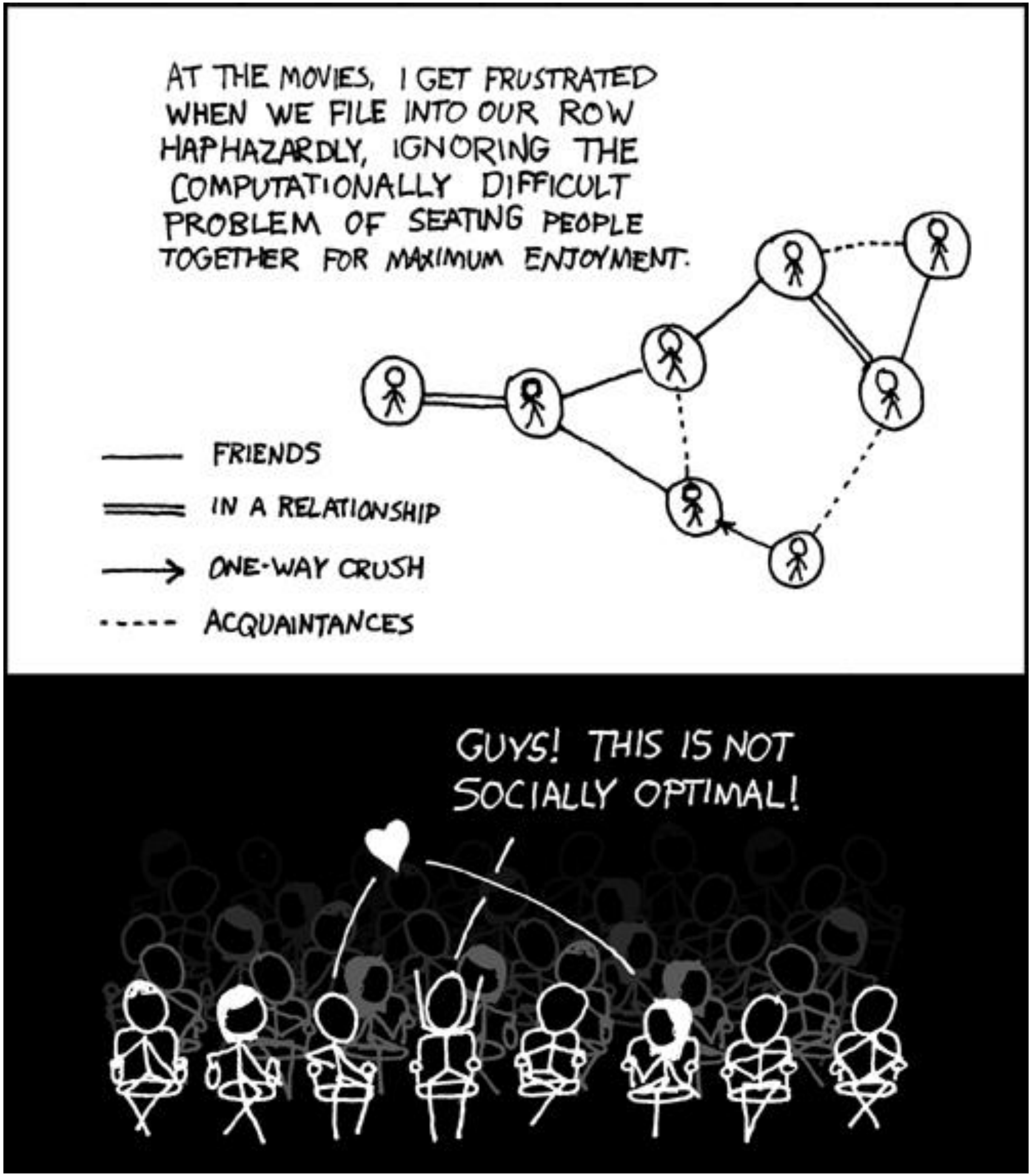
- Compiler optimizes assignment of variables to registers;
- Interference graph: model registers as nodes, and edges represent variables needed simultaneously;
- Chromatic number corresponds to minimum # of CPU registers needed to accommodate all the variables.

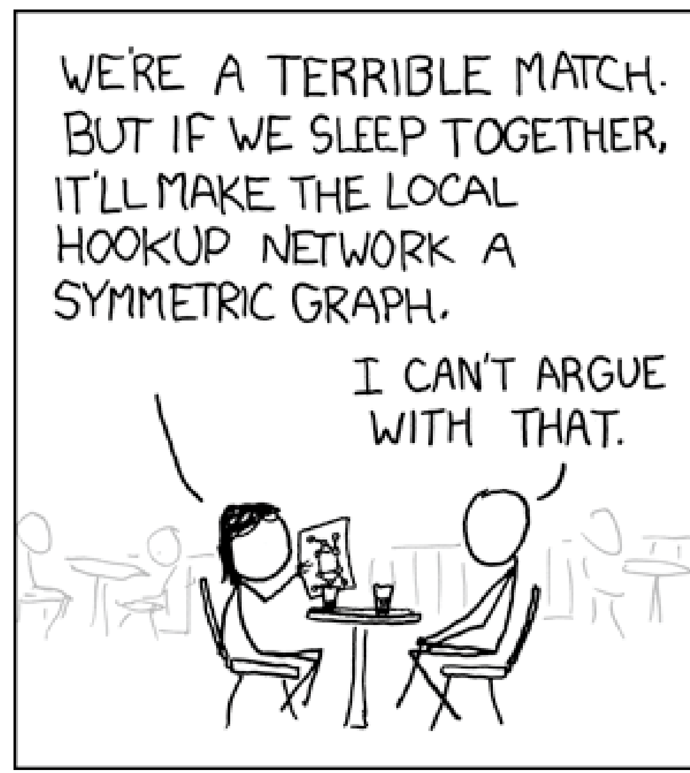


Variables & simultaneity

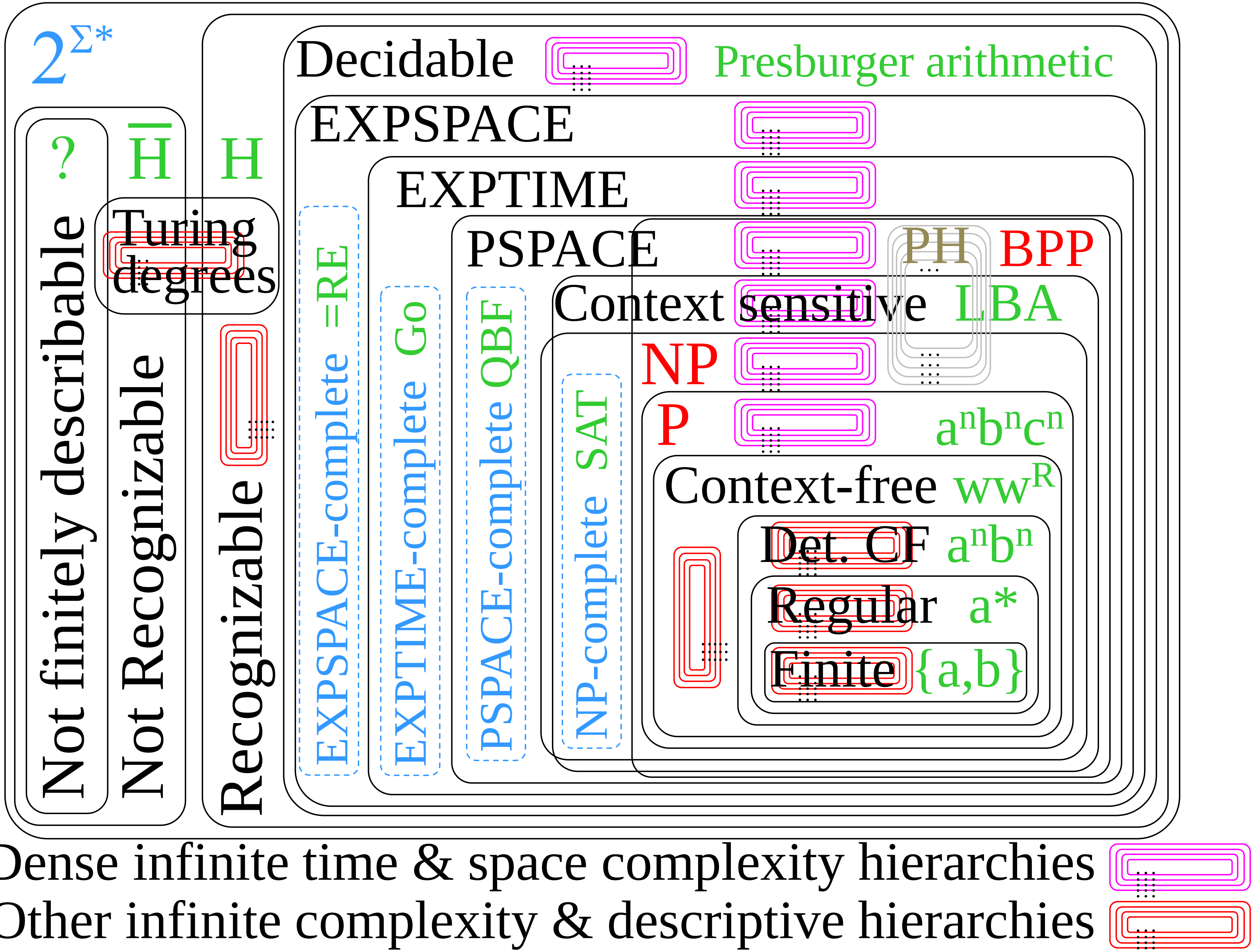
Graph coloring

Register allocation


AT THE MOVIES, I GET FRUSTRATED WHEN WE FILE INTO OUR ROW HAPHAZARDLY, IGNORING THE COMPUTATIONALLY DIFFICULT PROBLEM OF SEATING PEOPLE TOGETHER FOR MAXIMUM ENJOYMENT.
FRIENDS
IN A RELATIONSHIP
ONE-WAY CRUSH
ACQUAINTANCES
GUYS! THIS IS NOT SOCIALLY OPTIMAL!


WE'RE A TERRIBLE MATCH. BUT IF WE SLEEP TOGETHER, IT'LL MAKE THE LOCAL HOOKUP NETWORK A SYMMETRIC GRAPH.
I CAN'T ARGUE WITH THAT.

The Extended Chomsky Hierarchy Reloaded



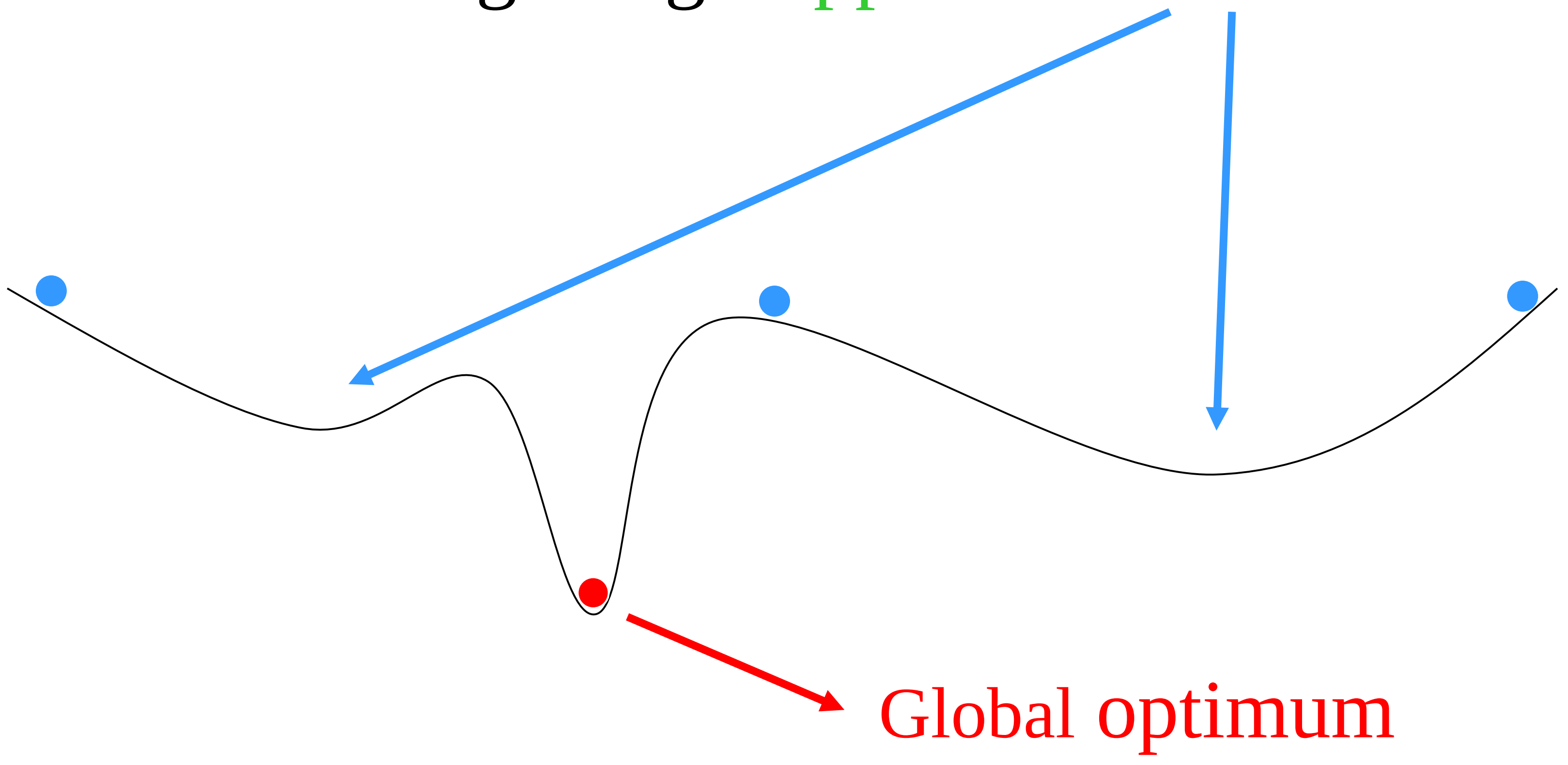
Algorithms

Tradeoff: Execution speed vs. solution quality

Speed	Solution	exact	approximate
	fast	"Short & sweet"	"Quick & dirty"
	slow	"Slowly but surely"	"Too little, too late"

Computational Complexity

Problem: Avoid getting trapped in local minima

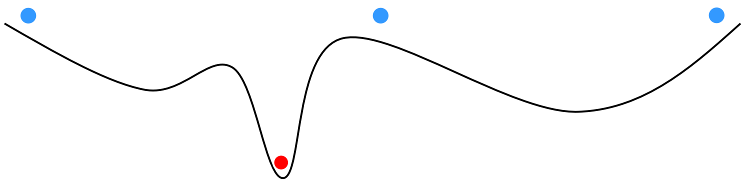


Approximation Algorithms

Idea: Some intractable problems can be efficiently approximated within close to optimal!

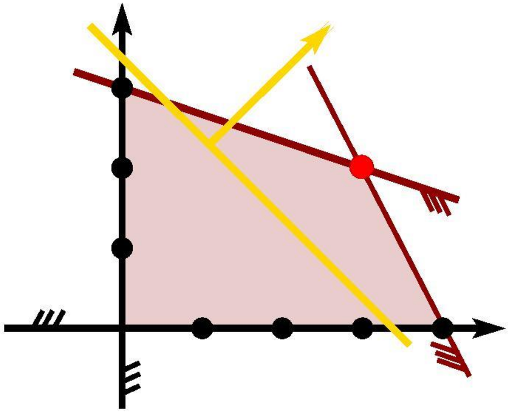
Fast:

- Simple heuristics (e.g., greed)
- Provably-good approximations



Slower:

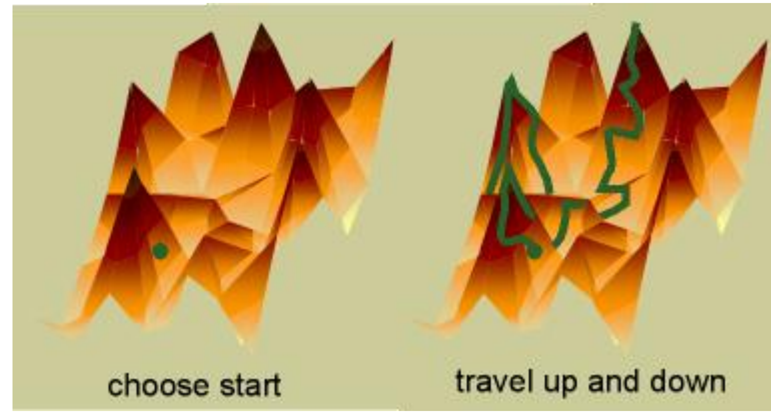
- Branch-and-bound approaches
- Integer Linear Programming relaxation

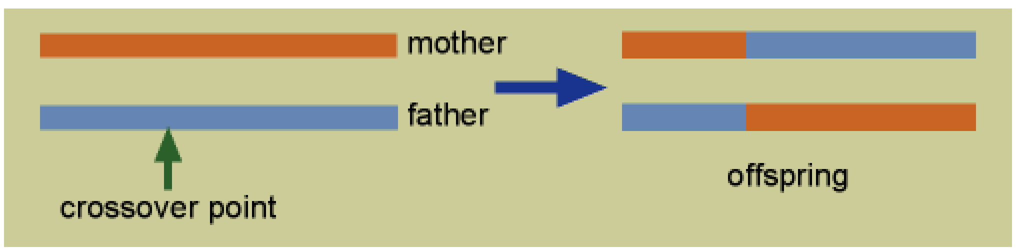


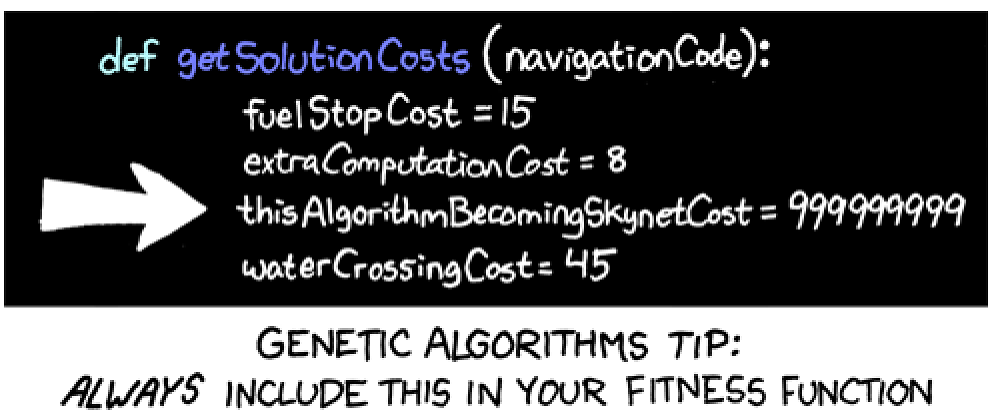
Approximation Algorithms

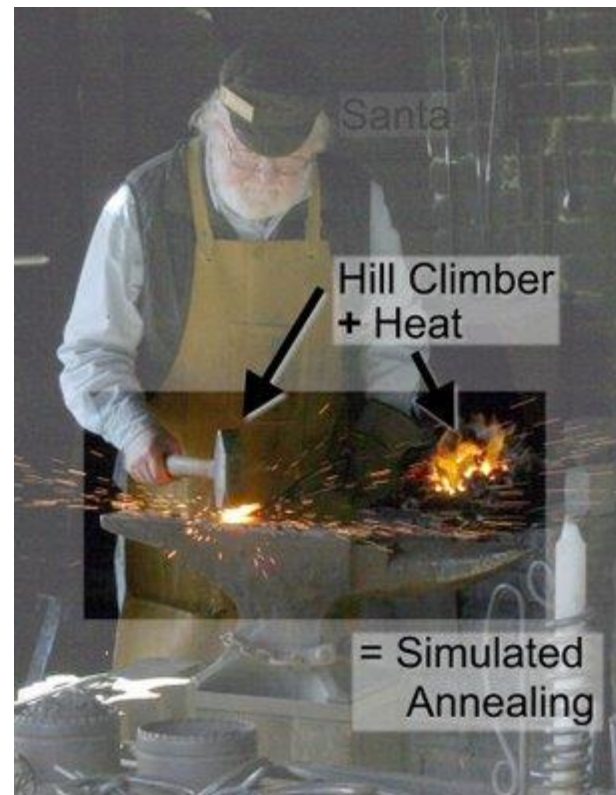
Wishful:

- Simulated annealing
- Genetic algorithms





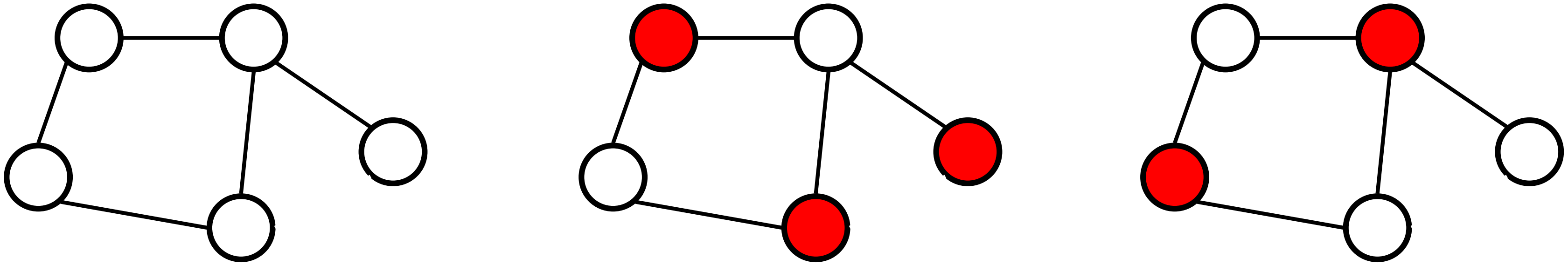




Minimum Vertex Cover

Minimum vertex cover problem: Given a graph, find a minimum set of vertices such that each edge is incident to at least one of these vertices.

Example:



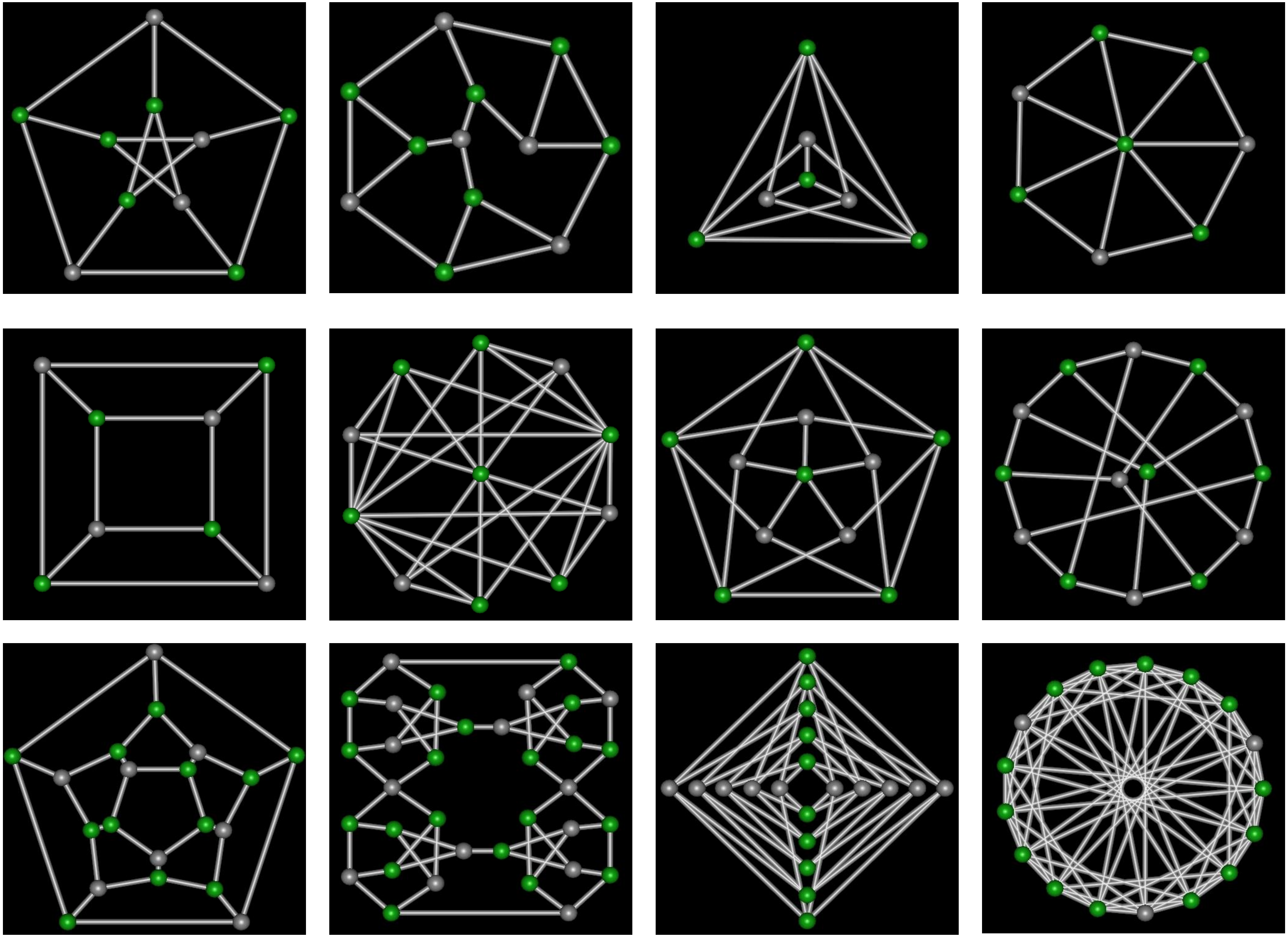
Input graph Heuristic solution Optimal solution

Applications: bioinformtics, communications, civil engineering, electrical engineering, etc.

- One of Karp's original NP-complete problems

Richard Karp

Minimum Vertex Cover Examples



Approximate Vertex Cover

Theorem: The minimum vertex cover problem is NP-complete (even in planar graphs of max degree 3).

Theorem: The minimum vertex cover problem can be solved exactly within exponential time $n^{O(1)}2^{O(n)}$.

Theorem: The minimum vertex cover problem can not be approximated within $\leq$ 1.36*OPT unless P=NP.

Theorem: The minimum vertex cover problem can be approximated (in linear time) within 2*OPT.

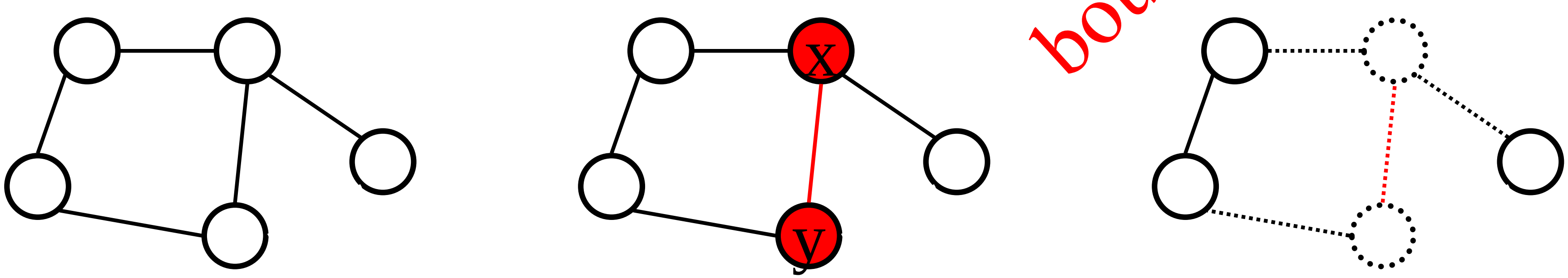
Idea: pick an edge, add its endpoints, and repeat.

Approximate Vertex Cover

Algorithm [Gavril, 1974]: Linear time 2*OPT approximation for minimum vertex cover:

- Pick random edge (x,y)
- Add {x,y} to the heuristic solution
- Eliminate x and y from graph
- Repeat until graph is empty

Best approximation bound known for VC!



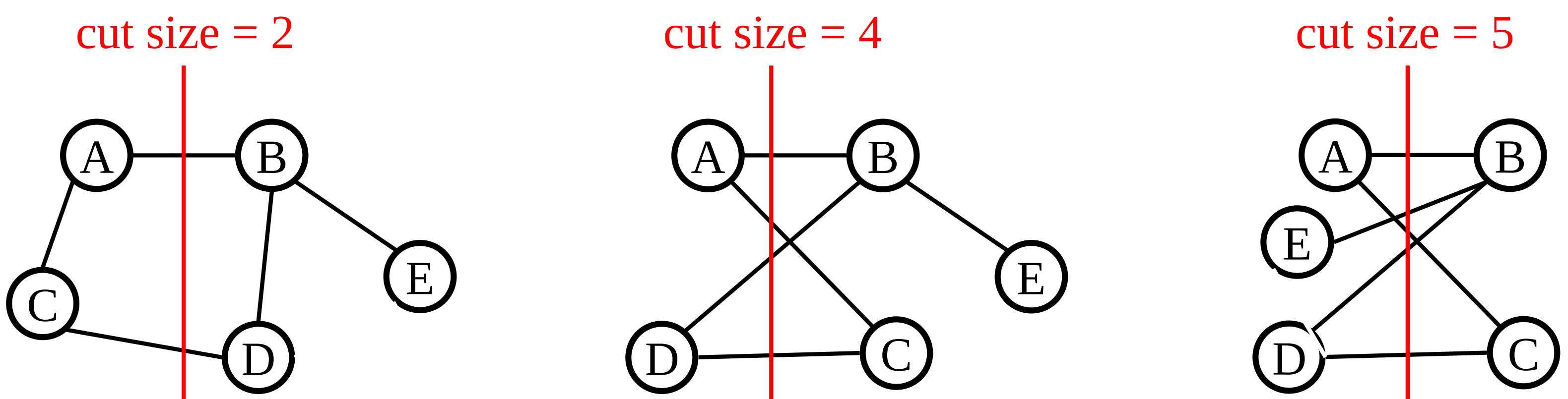
Idea: one of {x,y} must be in any optimal solution.

⇒ Heuristic solution is no worse than 2*OPT.

Maximum Cut

Maximum cut problem: Given a graph, find a partition of the vertices maximizing the # of crossing edges.

Example:



Input graph Heuristic solution Optimal solution

Applications: VLSI circuit design, statistical physics, communication networks.

- One of Karp's original NP-complete problems.

Richard Karp

Maximum Cut

Theorem [Karp, 1972]: The minimum vertex cover problem is NP-complete.

Theorem: The maximum cut problem can be solved in polynomial time for planar graphs.

Theorem: The maximum cut problem can not be approximated within ≤ 17/16*OPT unless P=NP.

=1.0625*OPT

Theorem: The maximum cut problem can be approximated in polynomial time within 2*OPT.

Theorem: The maximum cut problem can be approximated in polynomial time within 1.14*OPT.

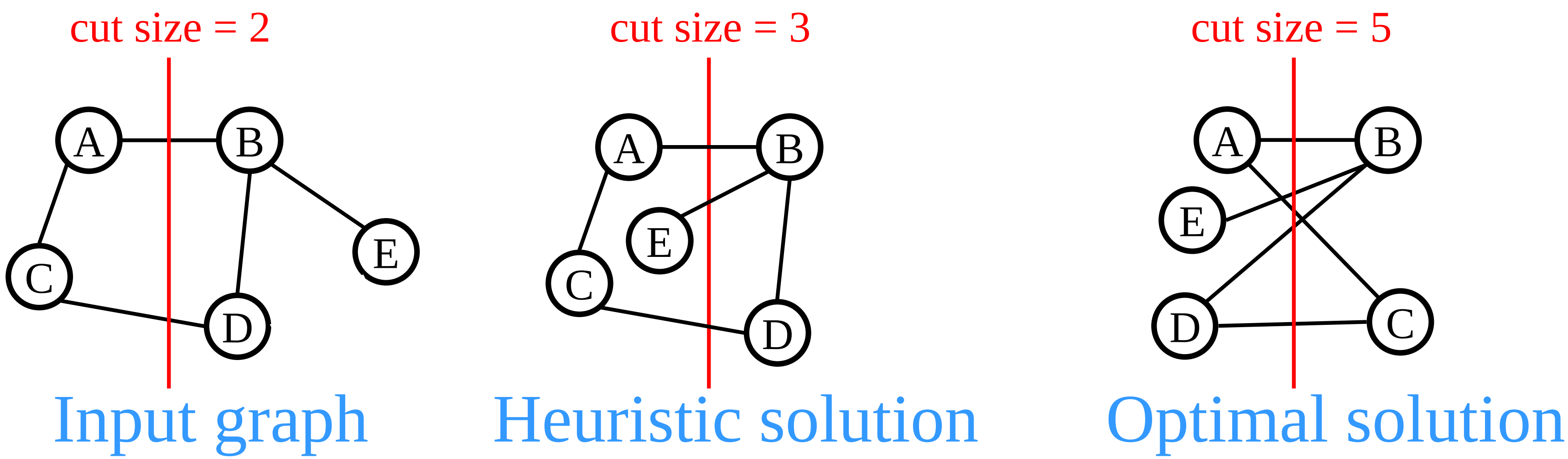
Maximum Cut

Algorithm: 2*OPT approximation for maximum cut:

Start with an arbitrary node partition

– If moving an arbitrary node across the partition will improve the cut, then do so

– Repeat until no further improvement is possible



Idea: final cut must contain at least half of all edges.

$\Rightarrow$ Heuristic solution is no worse than 2*OPT.

Why?

Approximate Traveling Salesperson

Traveling salesperson problem: given a pointset, find shortest tour that visits every point exactly once.

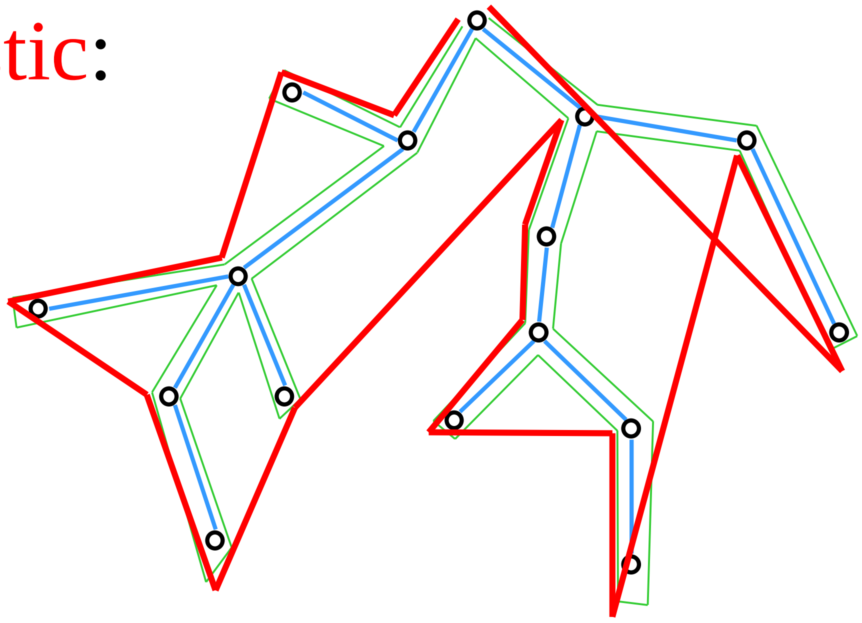
2*OPT metric TSP heuristic:

– Compute MST

– T = Traverse MST

– S = shortcut tour

– Output S

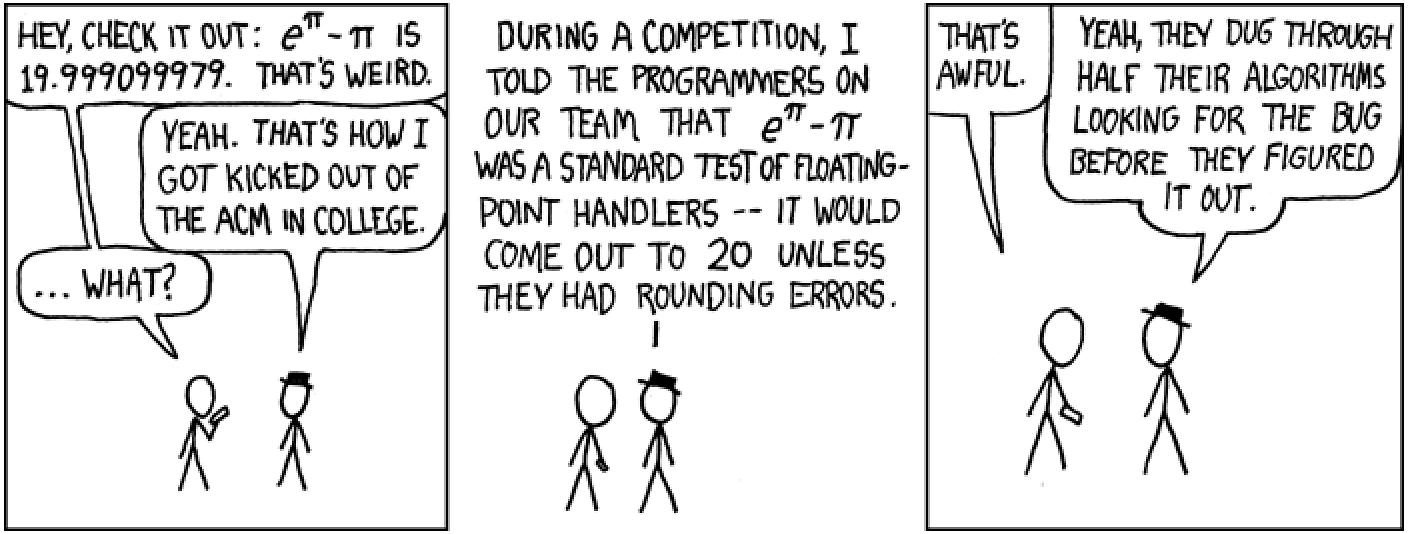


Analysis: S < T = 2*MST < 2*OPT TSP

triangle inequality!

T covers minimum spanning tree twice

TSP minus an edge is a spanning tree



"I gave it the traveling salesman problem. It said he should give up sales and go into banking."

Non-Approximability

- NP transformations typically do not preserve the approximability of the problem!
- Some NP-complete problems can be approximated arbitrarily close to optimal in polynomial time.

Theorem [Arora, 1996] Geometric TSP approximation in polynomial time within $(1+\varepsilon)$*OPT for any $\varepsilon>0$.

- Other NP-complete problems can not be approximated within any constant in polynomial time (unless P=NP).

Theorem: General graph TSP can not be approximated efficiently within K*OPT for any K>0 (unless P=NP).

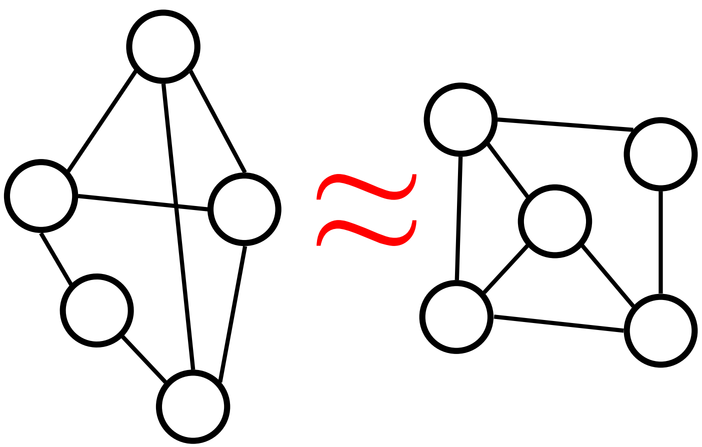
Graph Isomorphism

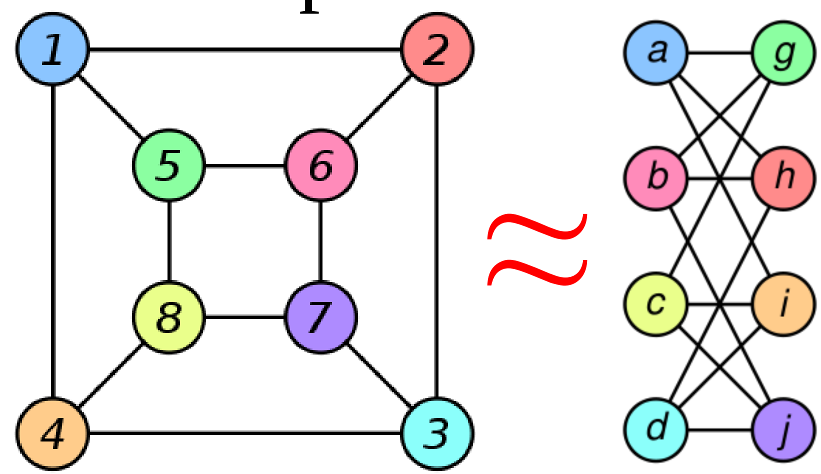
Definition: two graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are isomorphic iff $\exists$ bijection $f:V_1\rightarrow V_2$ such that

$$\forall v_i,v_j\in V_1 \quad (v_i,v_j)\in E_1 \Leftrightarrow (f(v_i),f(v_j))\in E_2$$

Isomorphism $\equiv$ edge-preserving vertex permutation

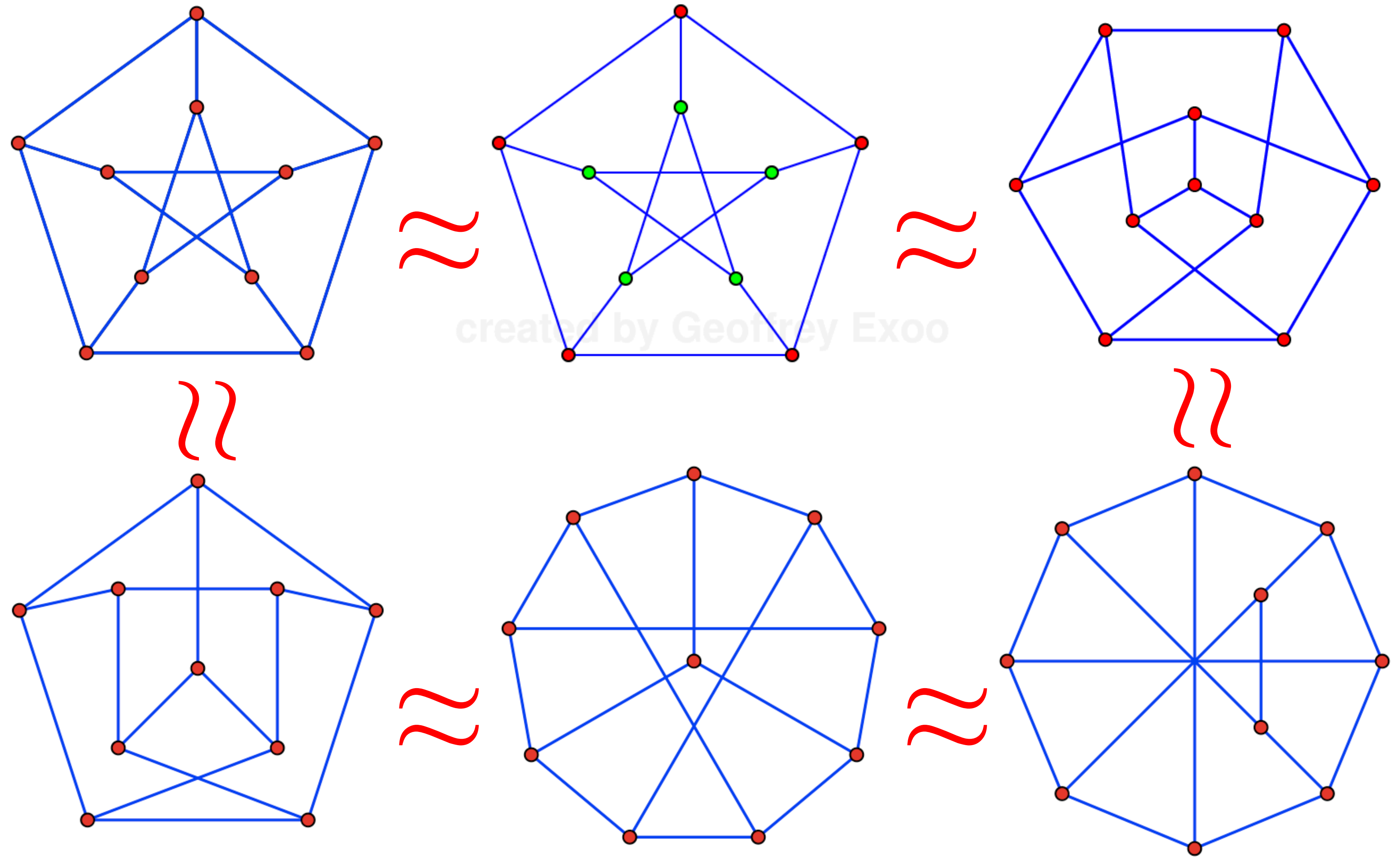
Problem: are two given graphs isomorphic?





Note: Graph isomorphism $\in$NP, but not known to be in P

Graph Isomorphism

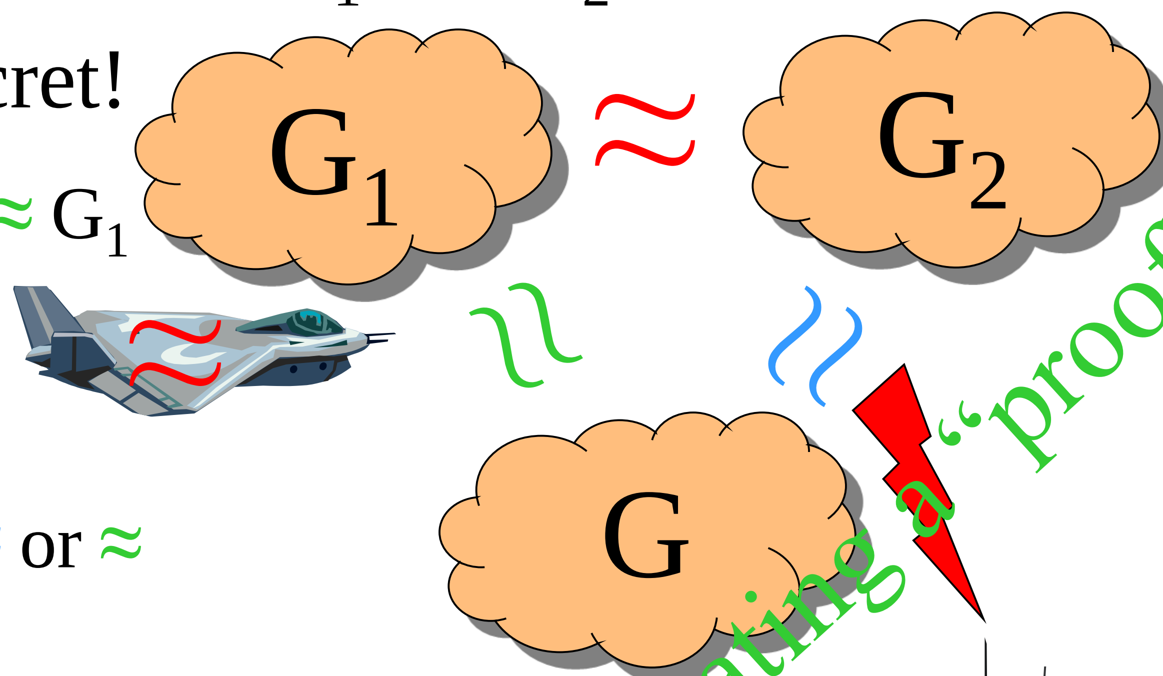


Zero-Knowledge Proofs

Idea: proving graph isomorphism without disclosing it!

Premise: Everyone knows G_1 and G_2 but not $\approx$

$\approx$ must remain secret!

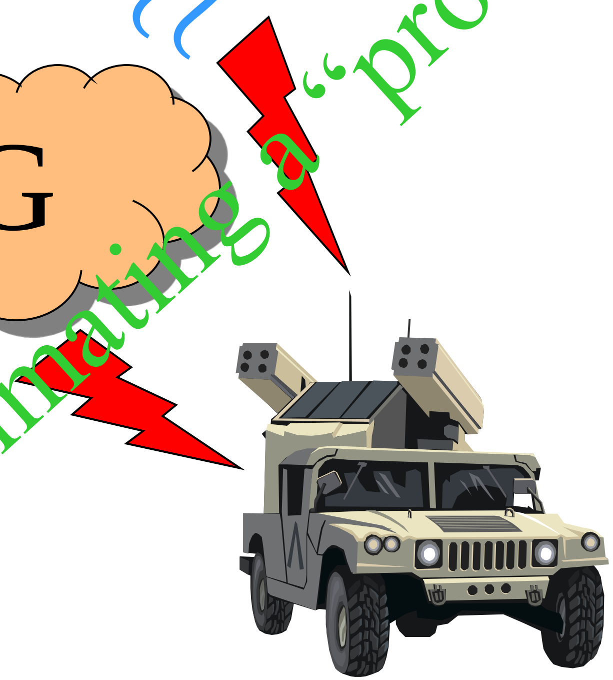


- Create random $G \approx G_1$
- Note: $\approx$ is $\approx(\approx)$
- Broadcast G
- Verifier asks for $\approx$ or $\approx$
- Broadcast $\approx$ or $\approx$
- Verifier checks $G \approx \mathbf{G_1}$ or $G \approx G_2$

Repeat k times

$\Rightarrow$ Probability of cheating: 2^{-k}

Approximating a "proof"!



Zero-Knowledge Proofs

Idea: prove graph 3-colorable without disclosing how!

Premise: Everyone knows G_1 but not its 3-coloring χ which must remain secret!

Create random $G_2 \approx G_1$

Note: 3-coloring $\chi'(G_2)$ is $\approx(\chi(G_1))$

Broadcast G_2

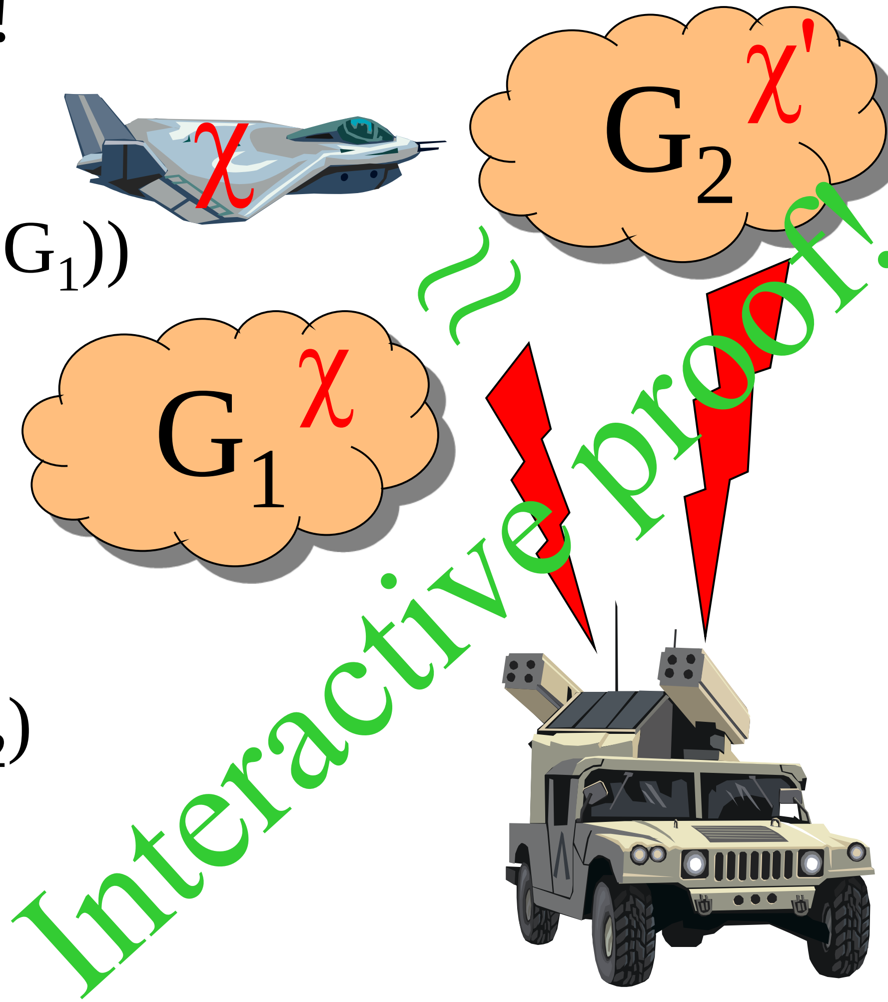
Verifier asks for $\approx$ or χ'

Broadcast $\approx$ or χ'

Verifier checks $G_1 \approx G_2$ or $\chi'(G_2)$

Repeat k times

$\Rightarrow$ Probability of cheating: 2^{-k}



Zero-Knowledge Caveats

- Requires a good random number generator
- Should not use the same graph twice
- Graphs must be large and complex enough

Applications:

- Identification friend-or-foe (IFF)
- Cryptography
- Business transactions

Zero-Knowledge Proofs

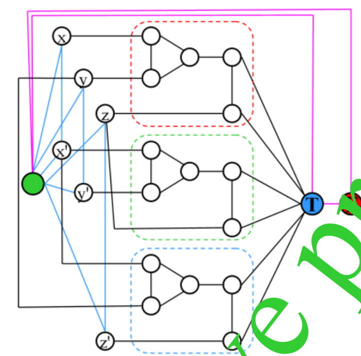
Idea: prove that a Boolean formula P is satisfiable without disclosing a satisfying assignment!

Premise: Everyone knows P but not its secret satisfying assignment V !

$P = (x+y+z)(x'+y'+z)(x'+y+z')$

f

G =



- Convert P into a graph 3-colorability instance G = f(P)
- Publically broadcast f and G
- Use zero-knowledge protocol to show that G is 3-colorable

⇒ P is satisfiable iff G is 3-colorable

⇒ P is satisfiable with probability $1-2^{-k}$

Interactive proof!

Interactive Proof Systems

- Prover has unbounded power and may be malicious
- Verifier is honest and has limited power

Completeness: If a statement is true, an honest verifier will be convinced (with high probability) by an honest prover.

Soundness: If a statement is false, even an omnipotent malicious prover can not convince an honest verifier that the statement is true (except with a very low probability).

- The induced complexity class depends on the verifier's abilities and computational resources:

Theorem: For a deterministic P-time verifier, class is NP.

Def: For a probabilistic P-time verifier, induced class is IP.

Theorem [Shamir, 1992]: IP = PSPACE