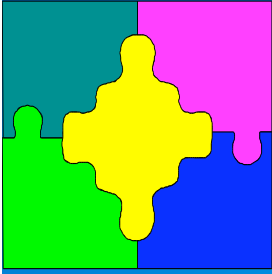


CSE460: Optimisation & Constraint Solving

Bernd Meyer

Introduction

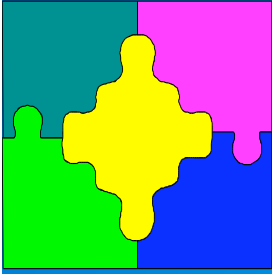
- ◆ Constraint optimization problems
- ◆ Industrial applications
- ◆ Types of problems
- ◆ Main approaches
- ◆ Overview of CSE 460



Industrial Applications

Constraint optimization problems are very important to industry

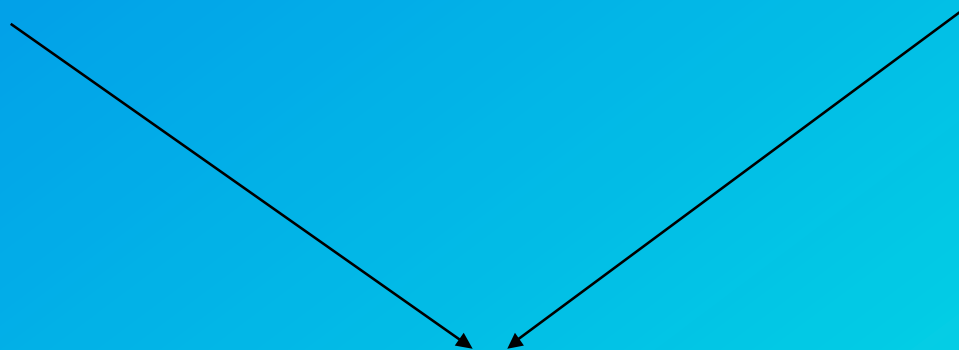
- ◆ Timetabling
- ◆ Scheduling
- ◆ Rostering
- ◆ Fleet Planning
- ◆ Routing
- ◆ Hub location
- ◆ Layout in windowing systems / GUI toolkits
- ◆ Graph layout
- ◆ Protein folding
- ◆



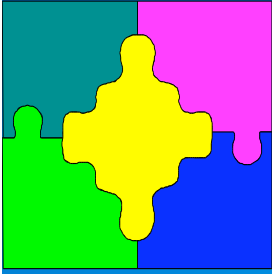
Constraint Optimisation Problems

Constraint Solving
(finding some solution)

Optimization
(finding the best solution)



Constraint optimization



Constraint Problems

- ◆ ***Variable***: a place holder for an unknown value

X, Y, Z, Temp

(We use identifiers starting with an uppercase letter)

- ◆ ***Function symbols***: map values to values

+, -, *, sin, /

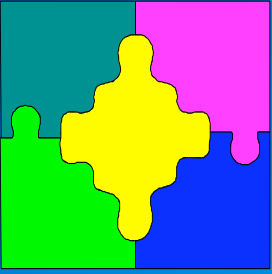
- ◆ ***Relation symbols***: relationship between values

=, ≤, ≠

- ◆ ***Primitive constraint***: a relation symbol with arguments

- ◆ ***Constraint***: a nested conjunction/disjunction of primitive constraints

$$X \leq 3 \wedge X = Y \wedge Y \geq 4$$



Modelling with constraints

- ◆ Constraints describe *idealized behaviour* of objects in the real world

$$V1 = I1 \times R1$$

$$V2 = I2 \times R2$$

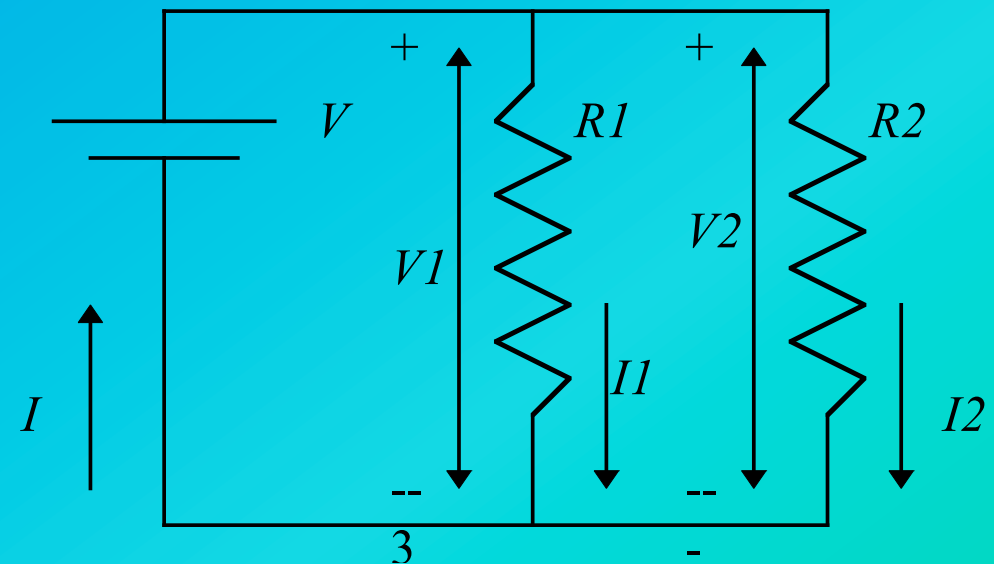
$$V - V1 = 0$$

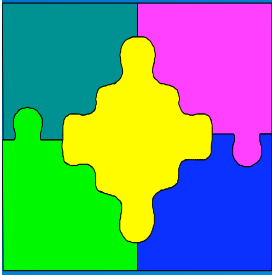
$$V - V2 = 0$$

$$V1 - V2 = 0$$

$$I - I1 - I2 = 0$$

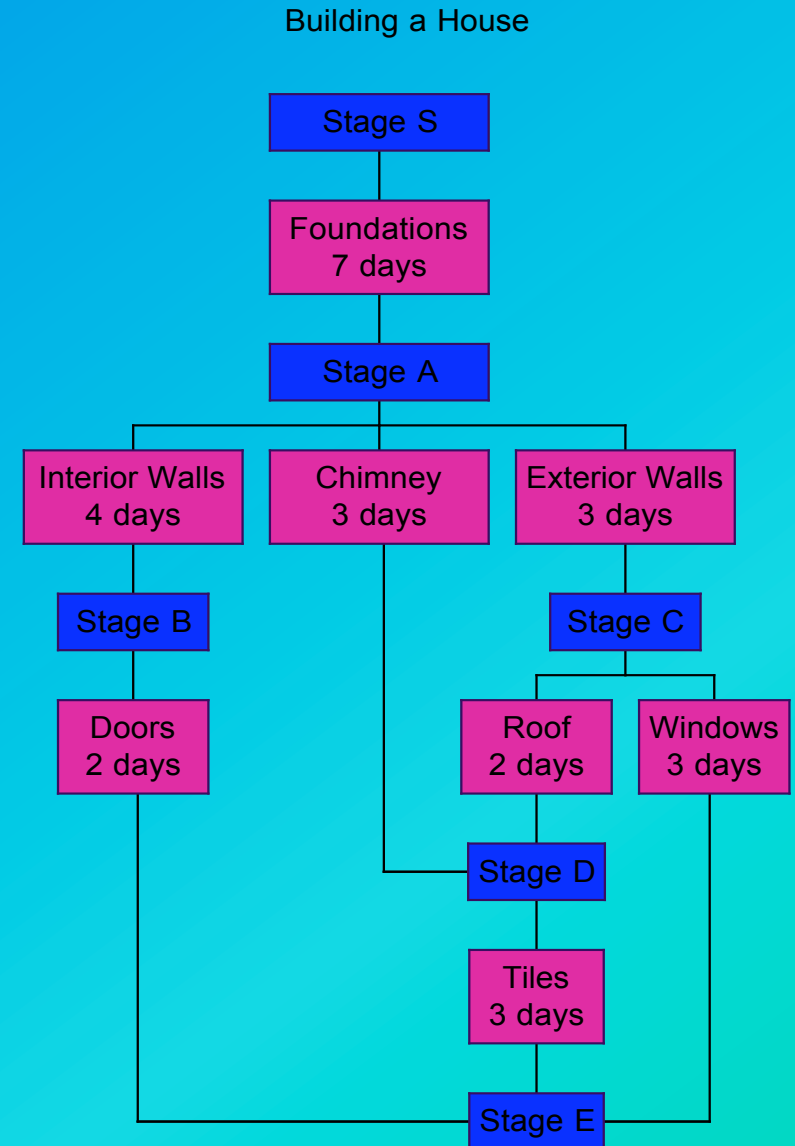
$$-I + I1 + I2 = 0$$



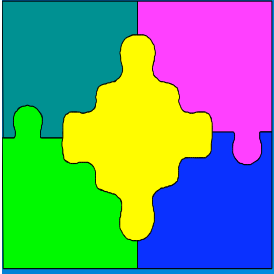


Modelling with constraints

start	$T_S \geq 0$
foundations	$T_A \geq T_S + 7$
interior walls	$T_B \geq T_A + 4$
exterior walls	$T_C \geq T_A + 3$
chimney	$T_D \geq T_A + 3$
roof	$T_D \geq T_C + 2$
doors	$T_E \geq T_B + 2$
tiles	$T_E \geq T_D + 3$
windows	$T_E \geq T_C + 3$



Note that such a model does not have to be complete !



Satisfiability

Valuation: an assignment of values to variables

$$\theta = \{X \mapsto 3, Y \mapsto 4, Z \mapsto 2\}$$

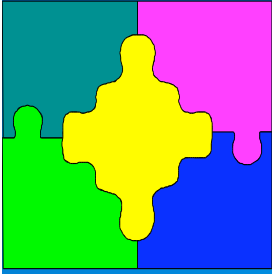
$$\theta(X + 2Y) = (3 + 2 \times 4) = 11$$

we also write: $(X + 2Y)\theta$

Solution: valuation which satisfies constraint

$$\theta(X \geq 3 \wedge Y = X + 1)$$

$$= (3 \geq 3 \wedge 4 = 3 + 1) = \textit{true}$$



Satisfiability (Cont)

Satisfiable: constraint has a solution

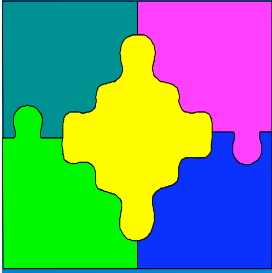
Unsatisfiable: constraint does not have a solution

$$X \leq 3 \wedge Y = X + 1$$

satisfiable

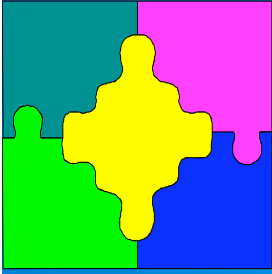
$$X \leq 3 \wedge Y = X + 1 \wedge Y \geq 6$$

unsatisfiable



Optimization

- ◆ Often not just any solution, but a “*best*” solution is sought
- ◆ This is an *optimization problem*
- ◆ We need an *objective function* so that we can rank solutions, that is a mapping from solutions to a real value
- ◆ optimization problem = constraint C & objective function f
- ◆ A valuation $v1$ is *preferred* to valuation $v2$ if
 - $f(v1) < f(v2)$ for a minimization problem
 - $f(v1) > f(v2)$ for a maximization problem
- ◆ An (*optimal*) *solution* to an optimisation problem is a solution of C such that no other solution of C is preferred to it.



Optimization Example

An optimization problem
 $\min f \equiv X^2 + Y^2$ subject to

$$C \equiv X + Y \geq 4$$

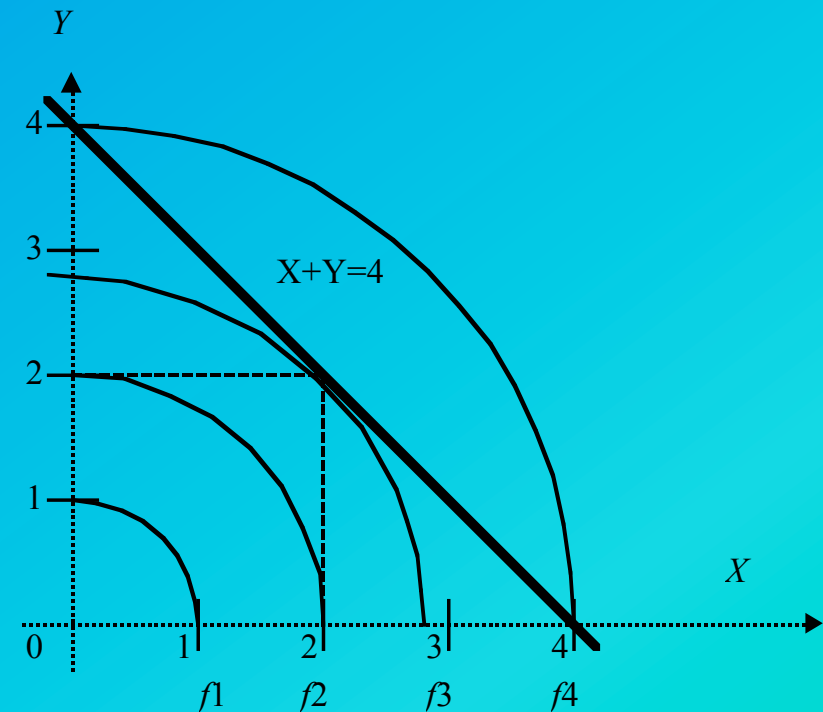
Find the closest point to the
origin satisfying the C .

Some solutions and f value

$$\{X \mapsto 0, Y \mapsto 4\} \quad 16$$

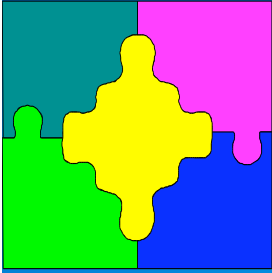
$$\{X \mapsto 3, Y \mapsto 3\} \quad 18$$

$$\{X \mapsto 2, Y \mapsto 2\} \quad 8$$



Optimal solution

$$\{X \mapsto 2, Y \mapsto 2\}$$



Optimization

- ◆ Some optimization problems have *no solution*.

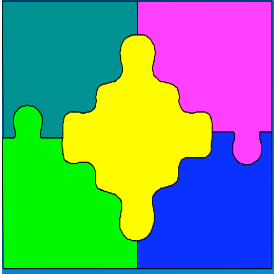
- ◆ Constraint has no solution (“*over-constrained*”)

$$\min X^2 \text{ subject to } X \geq 2 \wedge X \leq 0$$

- ◆ Problem has no optimum (“*unbounded*”)

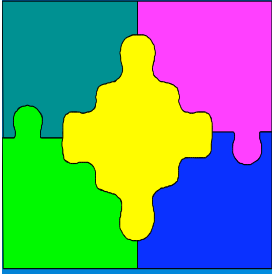
$$\min X \text{ subject to } X \leq 0$$

(For any solution there is more preferable one)



Dynamic Problems

- ◆ In many real-world applications have to solve a series of similar problems
For example rescheduling aircraft crews when someone calls in sick.
- ◆ Often we want to find a solution close to the old solution
I.e. we want to reschedule as few crew as possible
- ◆ These are called *dynamic problems*
- ◆ **Methods to solve these are also often called “adaptive”**

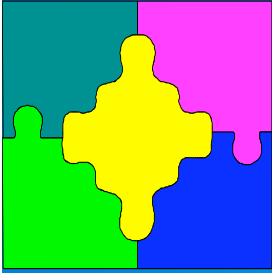


Types of Constraint Problems

- ◆ Continuous variable domain & linear constraints
 - *linear programming*
- ◆ Continuous variable domain & non-linear constraints
 - *non-linear programming*
- ◆ Discrete variable domain
 - *integer programming*
 - *mixed integer programming*
 - *finite domain constraints*

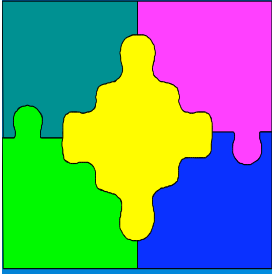
Non-linear and integer/finite domain problems are mostly *hard*

But: many classes of *convex non-linear* problems can be solved efficiently



Main Approaches to Solving Constraint Problems

- ◆ ***Operations research*** (mathematics)
 - ◆ Linear programming
 - ◆ Lagrangean methods
 - ◆ Integer programming
- ◆ ***Local Search***
 - ◆ Stochastic techniques e.g. simulated annealing
 - ◆ Discrete Lagrangean methods
- ◆ ***AI/Constraint programming***
 - ◆ Backtracking
 - ◆ Consistency based methods
 - ◆ Hybrid solving methods



Main Approaches to Modelling Constraint Problems

Modelling of constraint problems is often
problem

constraint-solving technique specific

because of need to encode problem specific heuristics.

There are some *generic* approaches

- ◆ *Operations research*

- ◆ *GAMS, AMPL* (linear (+ non-linear) mathematical programming)

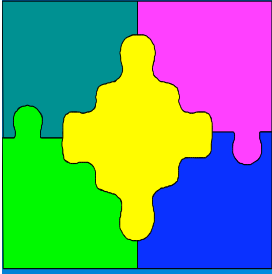
- ◆ *Constraint programming*

- ◆ CLP

- ◆ ILOG Solver

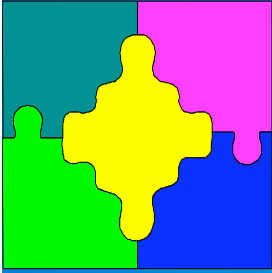
- ◆ OPL

- ◆ in practice also: Spreadsheet approaches



CSE460 Syllabus

- ▼ Introduction; Computational Complexity;
- ▼ Linear Programming; Simplex Method; 1
- ▼ Integer and Mixed Integer Programming;
- ▼ Network Flow; Branch & Bound Methods; 2
- ▼ Game Theory; Duality
- ▲ Euler-Lagrange Multiplier Methods; Kuhn-Tucker Conditions 3
- ▲ Non-linear Programming;
- ▲ (Interior Point Methods, if time allows) 4
- ▲ Introduction to Stochastic Constraint Solving and Optimization;
- ▲ Heuristics; 5
- ▲ Tabu Search, Simulated Annealing;
- ▲ Genetic Algorithms; 6
- ▲ Ant Colony Optimization
- ▲ Constraints in Meta-Heuristics 7



CSE460/461 Textbooks

There are several recommended books for this subject:

▼ *Introduction to Mathematical Programming*. Wayne L. Winston. Duxbury Press, 1995.

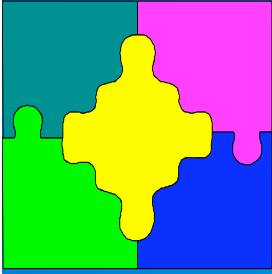
▼ *New Ideas in Optimization*, selected chapters. David Corne, Marco Dorigo and Frank Glover (Eds.). McGraw Hill, 2000.

▲ *Constraint Programming-An Introduction*. Kim Marriott and Peter Stuckey. MIT Press, 1998.

▲ *The OPL Optimization Programming Language*. Pascal van Hentenryck. MIT Press, 1999.

The last two books are more focused on constraint programming and are used with CSE 461 “Optimization and cvonstraint solving modelling languages”, but are usefeful for CSE 460, too.

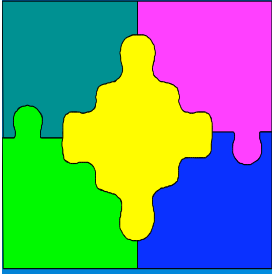
In addition to these books, selected research papers/texts will be referenced. The lecture material will be available on the Web.



CSE460/461 Further Reading

These books are some other good references on a more advanced level:

- ▼ *An Introduction to Optimization*. E. Chong. Wiley, 2001.
- ▼ *Handbook of Metaheuristics*. G.A. Kochenberger and F. Glover. Kluwer, 2003.
- ▼ *Numerical Optimization*, Nocedal, Jorge Wright, Stephen J. Springer, 1999
- ▼ *Practical Optimization Methods with Mathematica*. M. Bhatti. Springer, 2004
- ▼ *Numerical Methods Using Matlab*. John H. Mathews. Prentice Hall, 2004.
- ▼ *Numerical Recipes in C*. William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery. *Online at www.nr.com*



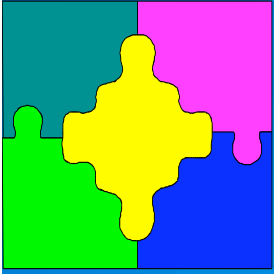
CSE460 General Info

- ◆ ***Lectures:*** S1/25 on Wednesday from 2pm-3pm
S1/25 on Friday from 2pm-4pm
only Weeks 1-7
- ◆ ***Lecturer:*** Bernd Meyer.
- ◆ ***Consultation:*** 4pm-6pm Wednesdays Rm 148, Blg 75.

There will be two ***assignments*** based on the course material.

- ◆ Assignment 1, due Friday 26th of August, 40%
- ◆ Assignment 2, due Friday 30th September, 60%

No Exam!



Summary

- ◆ Constraint problems
- ◆ Industrial applications
- ◆ Types of problems
- ◆ Main approaches
- ◆ Overview of CSE460

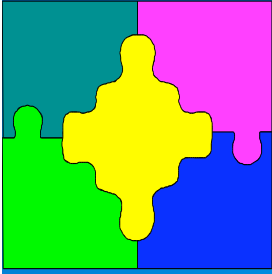
CSC460: Constraint Solving & Optimisation

--Complexity--

- ◆ Complexity classes
- ◆ NP-hardness & NP-completeness
- ◆ Proving NP-hardness & completeness
- ◆ Sample NP-complete problems

This lecture is based on the ``bible'' of NP-completeness:

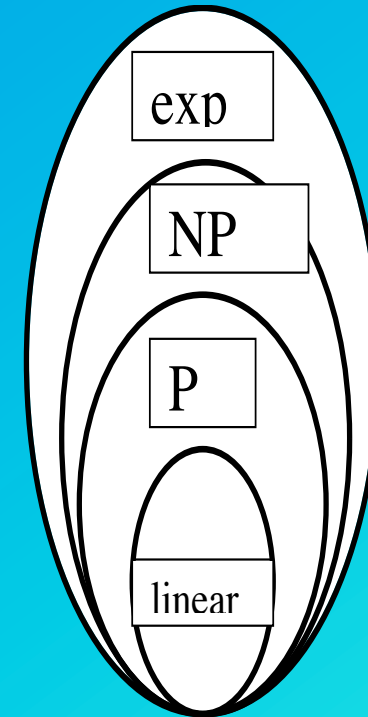
Garey & Johnson: *Computers & Intractability--A Guide to the Theory of NP-Completeness*. Freeman Press, 1979.

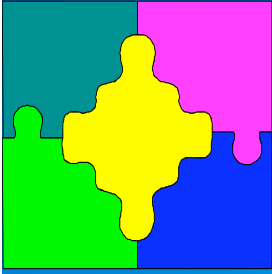


Complexity Classes

- ◆ Linear time
- ◆ Polynomial time (P)
- ◆ Non-deterministic Polynomial time (NP)
- ◆ Exponential time

Most *interesting* constraint problems are *NP-hard*



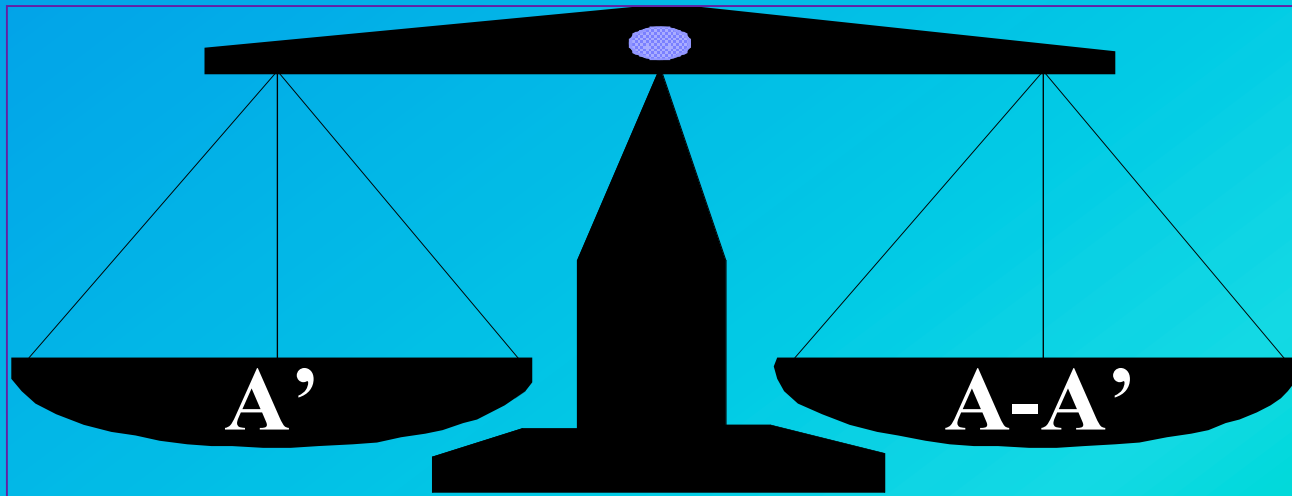


Partition

PARTITION

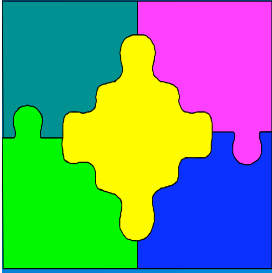
INSTANCE: A finite set A and a positive integer weight $w(a)$ for each $a \in A$

QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} w(a) = \sum_{a \in A - A'} w(a)$



For example the objects in A have weights 1, 2, 3, 4, 5, 5, 6.

Is this a yes instance?



NP Time

A problem Π is in NP if there is a *non-deterministic* algorithm to solve Π .

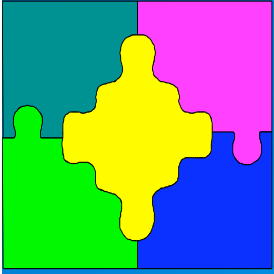
The algorithm

- A) *Guesses* a solution to Π if one exists
- B) *Checks* in polynomial time that this is a solution

For example PARTITION is in NP:

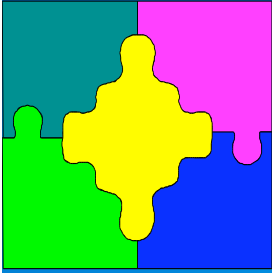
- A) Guess elements in A'
- B) Check that
$$\sum_{a \in A'} w(a) = \sum_{a \in A - A'} w(a)$$

Is PARTITION also in P?



Decision Problems

- ◆ The theory of NP-completeness is in terms of *decision problems*
- ◆ A *decision problem* Π consists of a set of *instances* D_Π and a subset of *yes-instances* Y_Π



Satisfiability

- ◆ A *literal* is either a Boolean variable u or its negation $\neg u$
- ◆ A *clause* is a disjunction of literals

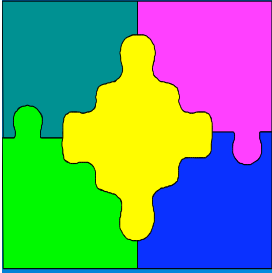
SATISFIABILITY

INSTANCE: A set U of variables and a conjunction C of clauses over the variables U

QUESTION: Is there a satisfying truth assignment for C ?

For example $U = \{X, Y\}$ and $C = (X \vee \neg Y) \wedge (\neg X \vee Y)$.

This is a yes-instance since $\{X \rightarrow true, Y \rightarrow true\}$ satisfies C .



NP-hardness & NP-completeness

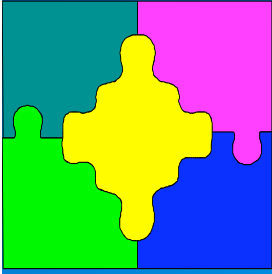
A *polynomial transformation* from decision problem Π to Π' is a function $F: D_{\Pi} \rightarrow D_{\Pi'}$ such that

- ◆ F is computable in polynomial time
- ◆ For all $I \in D_{\Pi}$, $I \in Y_{\Pi}$ if and only if $F(I) \in Y_{\Pi'}$

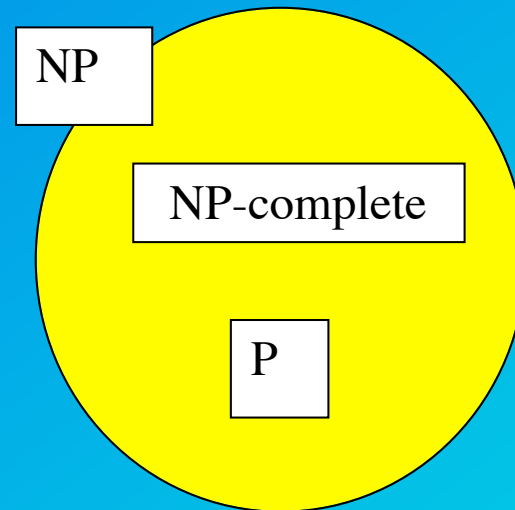
A decision problem Π is *NP-hard* if for each problem Π' in NP there is a polynomial transformation from Π' to Π .

I.e. if you can solve Π in polynomial time you can solve any NP problem in polynomial time.

A decision problem Π is *NP-complete* if it is both NP-hard and in NP.

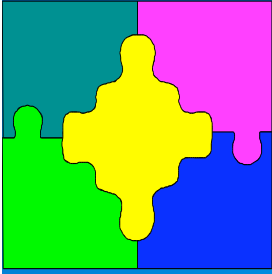


Structure of NP



No-one knows if $NP=P$. It is one of the biggest open questions in computer science/mathematics.

However almost everyone believes $NP \neq P$



Proving NP-hardness

- ◆ The first NP-complete problem:

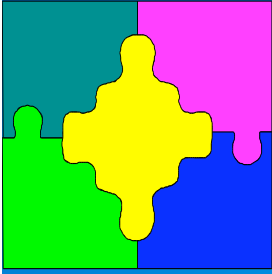
Cook's Theorem

SATISFIABILITY is NP-complete

- ◆ Using this and the following Lemma we can prove other problems are NP-hard.

Lemma

A decision problem Π is NP-hard if there is a polynomial transformation from some NP-hard problem Π' to Π .



Proving NP-hardness (Cont.)

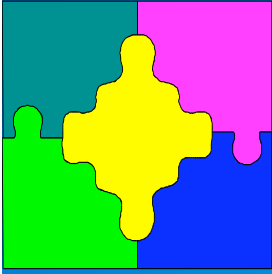
We can use the following steps to prove NP-hardness of Π :

- Select a similar known NP-hard problem Π'
- Construct a transformation F from Π' to Π
- Prove that F is computable in polynomial time

If in addition, Π is in NP, then Π is NP-complete

Lots of examples in (Garey & Johnson, 1979) of proving NP-completeness.

We will look at a simple example.



Sequencing Within Intervals

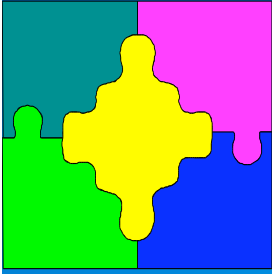
SEQUENCING WITHIN INTERVALS

INSTANCE: A finite set T of tasks and for each task $t \in T$

- ◆ A non-negative integer *release time* $r(t)$
- ◆ A non-negative integer *finish time* $f(t)$
- ◆ A non-negative integer *duration* $d(t)$

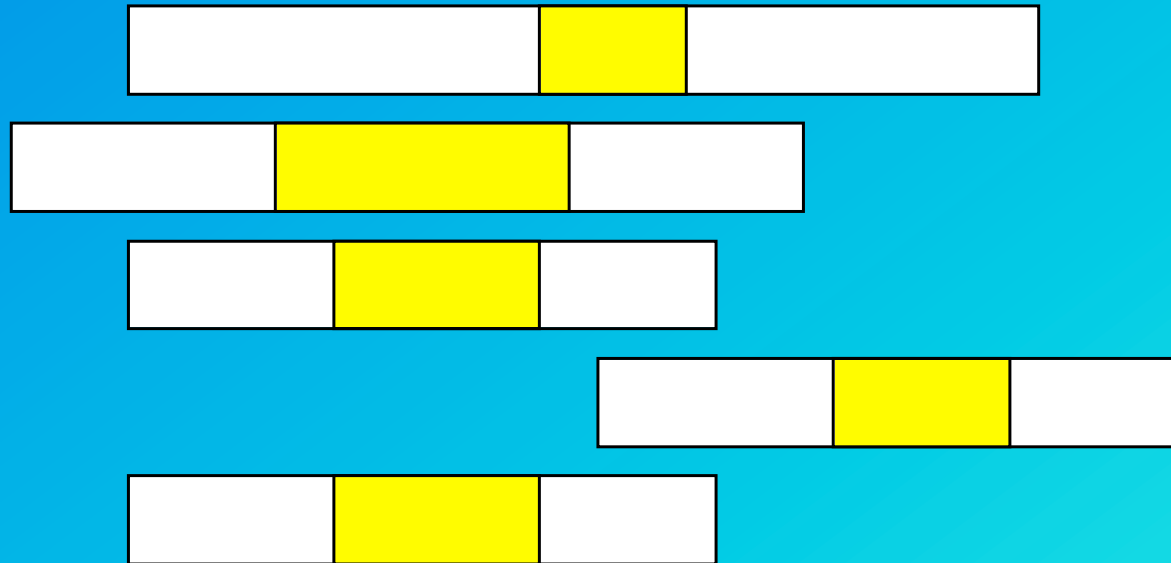
QUESTION: Is there a *feasible schedule* s such that for each task $t \in T$

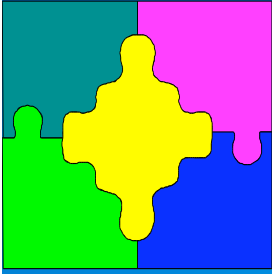
- ◆ t starts after its release time: $r(t) \leq s(t)$,
- ◆ t finishes before its finish time: $s(t) + d(t) \leq f(t)$,
- ◆ t does not *overlap* any other task:
for all $t' \in T - \{t\}$, *either* $s(t) \geq s(t') + d(t')$ *or* $s(t') \geq s(t) + d(t)$



Example of Sequencing

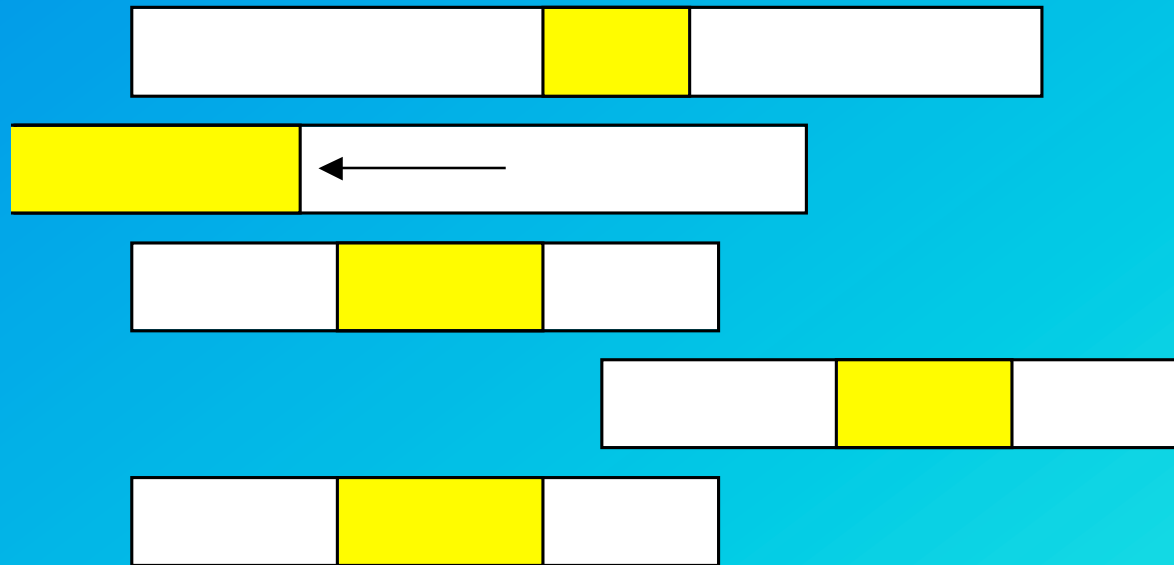
PROBLEM INSTANCE

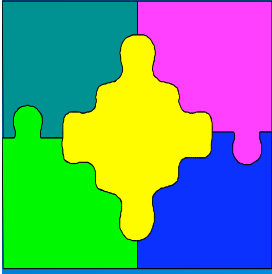




Example of Sequencing

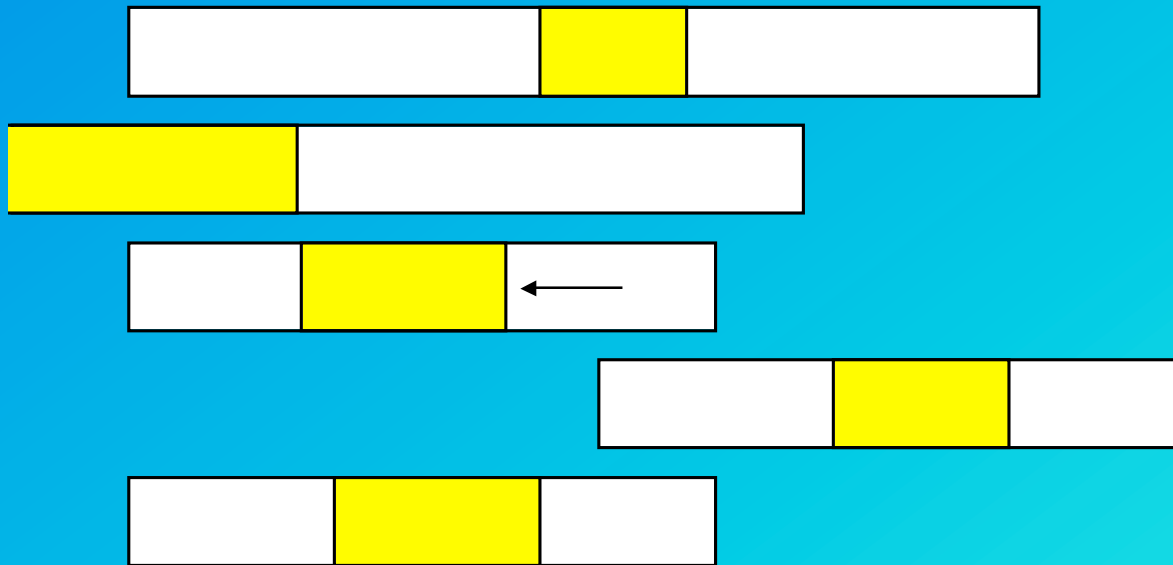
PARTIAL SOLUTION

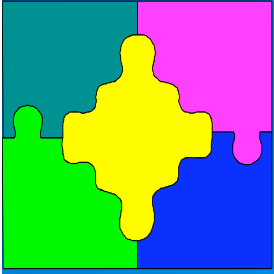




Example of Sequencing

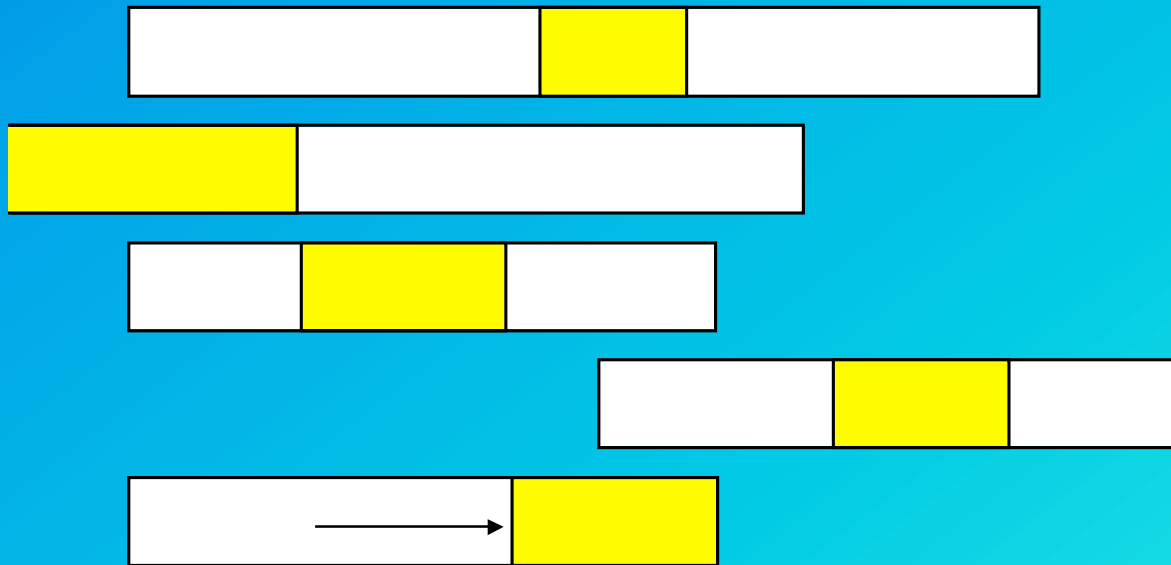
PARTIAL SOLUTION

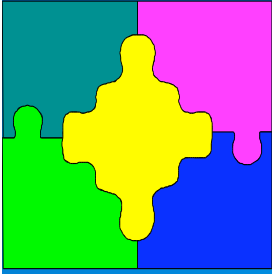




Example of Sequencing

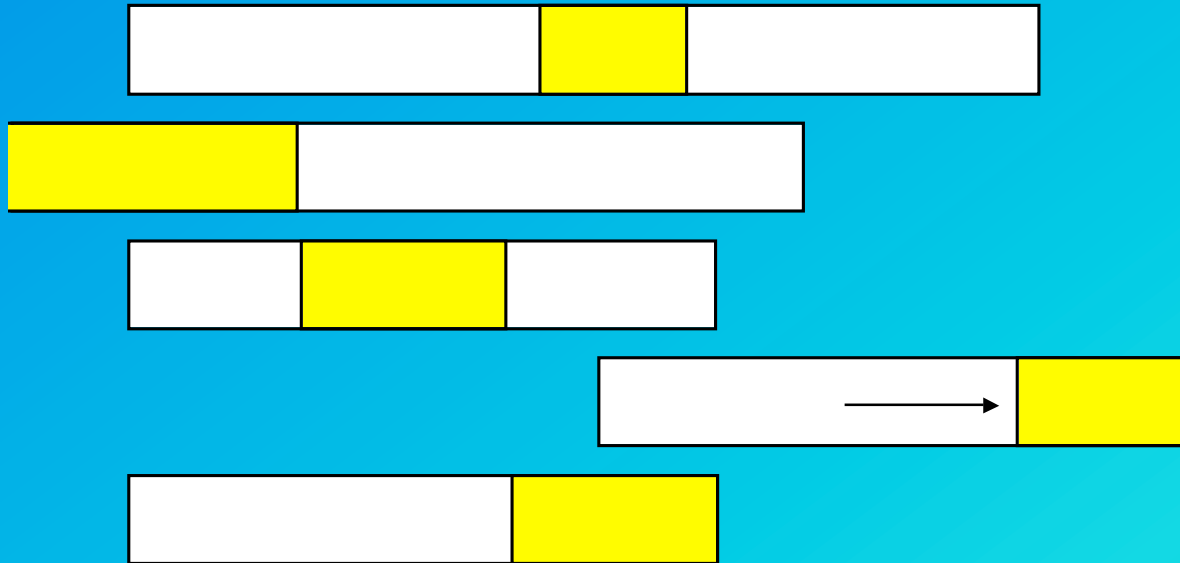
PARTIAL SOLUTION

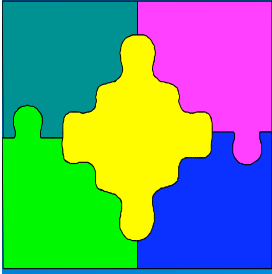




Example of Sequencing

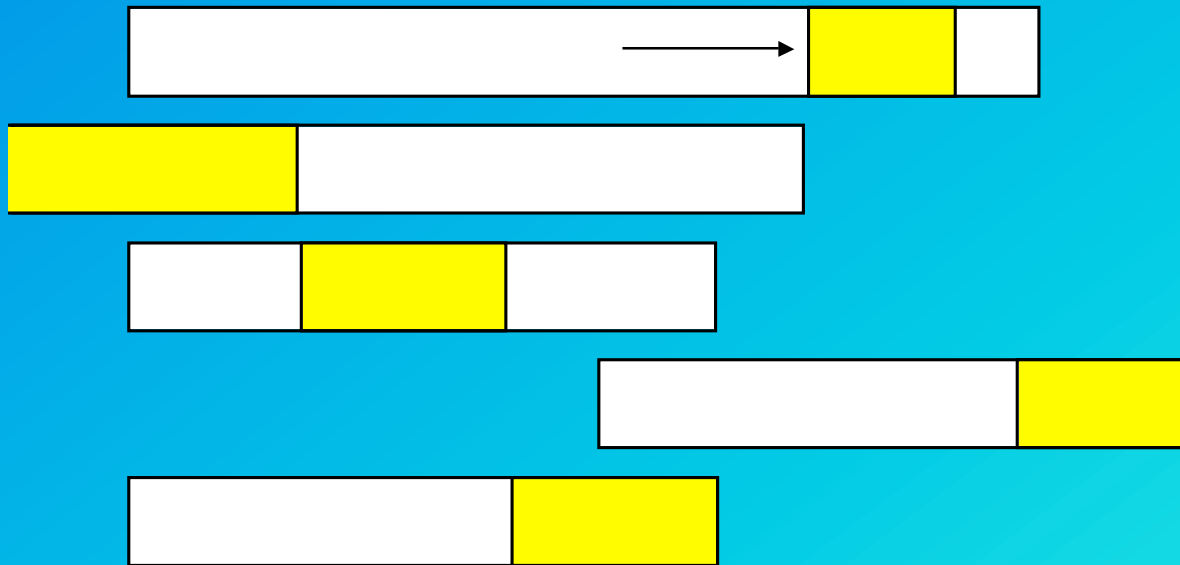
PARTIAL SOLUTION

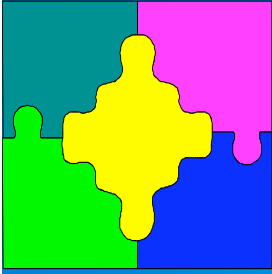




SOLUTION

SOLUTION





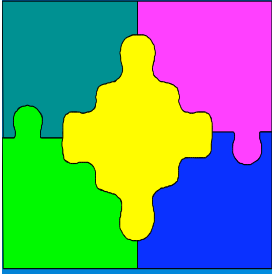
Proving NP-hardness (rept.)

We can use the following steps to prove NP-hardness of Π :

- Select a similar known NP-hard problem Π'
- Construct a transformation F from Π' to Π
- Prove that F is computable in polynomial time

If in addition, Π is in NP, then Π is NP-complete

To prove NP-hardness of “sequencing” we choose to reduce partitioning to sequencing.

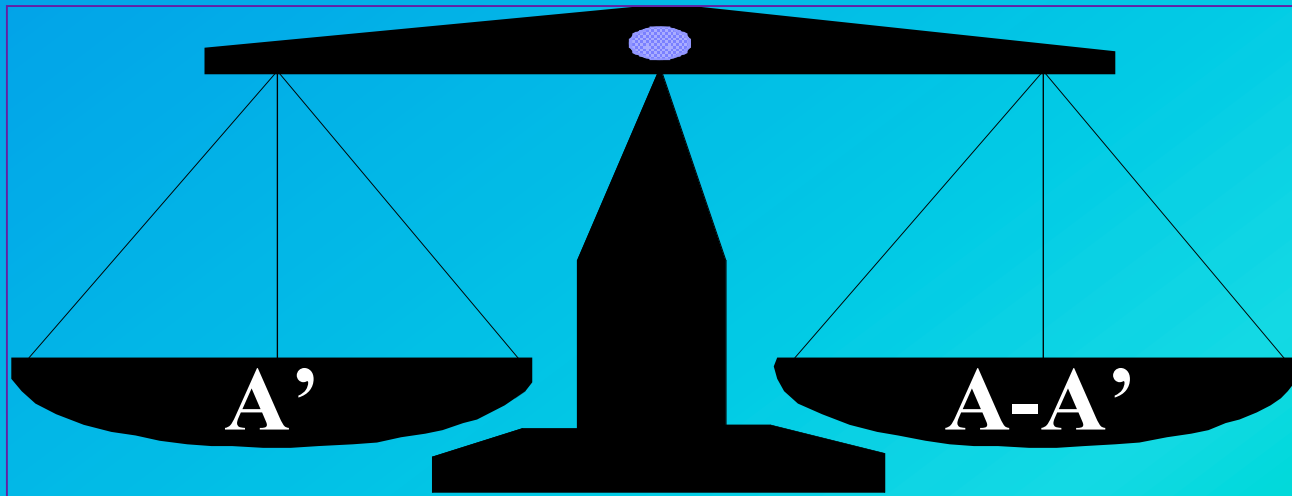


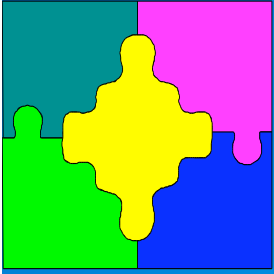
Partition (rept.)

PARTITION

INSTANCE: A finite set A and a positive integer weight $w(a)$ for each $a \in A$

QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} w(a) = \sum_{a \in A - A'} w(a)$





Proof of NP-Hardness of Sequencing Within Intervals

We give a polynomial reduction from PARTITION to SEQUENCING WITHIN INTERVALS

INSTANCE of PARTITION: A finite set A and a positive integer weight $w(a)$ for each $a \in A$

QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} w(a) = \sum_{a \in A - A'} w(a)$

Let $B = \sum_{a \in A} w(a)$. If B is odd stop with NO. Otherwise continue...

We encode each $a \in A$ using a task t_a .

Each t_a has duration $w(a)$, release time 0 and finish time $B+1$.

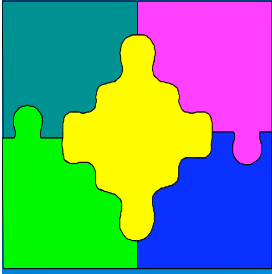
There is also an enforcer task \underline{t} .

\underline{t} has duration 1, release time $B/2$ and finish time $1+B/2$

This forces the other tasks to be scheduled in two intervals each of length B

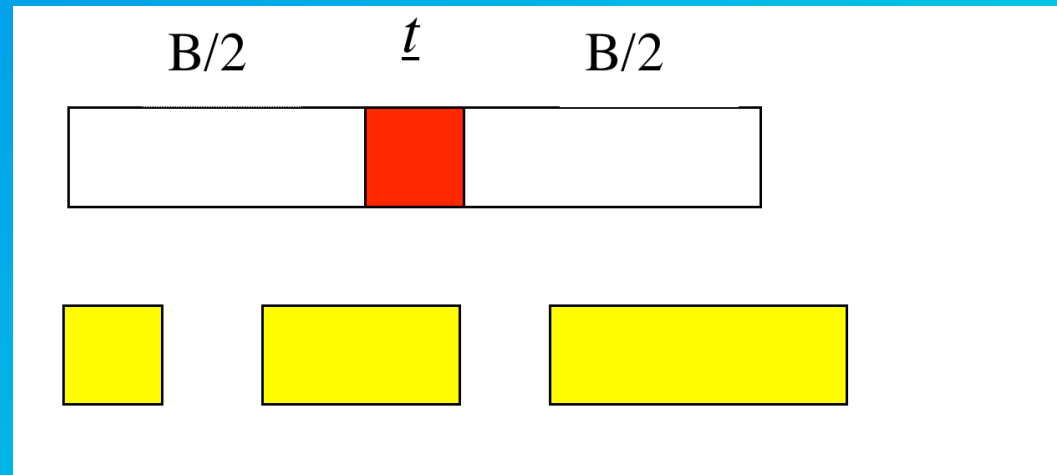
These tasks can be sequenced iff the original set can be partitioned.

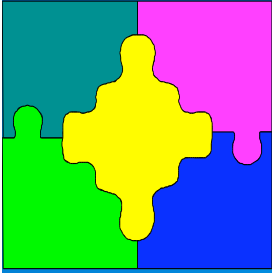
Clearly this transformation takes polynomial (linear) time.



Example Reduction

- ◆ For example the objects in A have weights 1, 2, 3. Then B is 6.





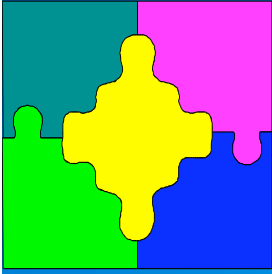
Some Other NP-Complete Problems

- ◆ TRAVELLING SALESPERSON
- ◆ KNAPSACK
- ◆ INTEGER PROGRAMMING
- ◆ PLANAR SUBGRAPH
- ◆ TIMETABLE DESIGN
- ◆

See Garey & Johnson 1979.

One problem which is *not* NP-Hard is

- ◆ LINEAR PROGRAMMING



Summary

- ◆ Non-deterministic Polynomial Time (NP)
- ◆ NP-hardness & NP-completeness
- ◆ Proving NP-hardness
- ◆ Selected NP-complete problems