Creating Informatics Olympiad Tasks: Exploring the Black Art

Benjamin Burton (Australia)
Mathias Hiron (France)

IOI 2008

Outline

What Makes a Good Task?

Remember your goals!

Examples:

- Does not directly resemble a classic algorithm
- Several solutions of varying difficulty and efficiency
- Concise implementation

Also see Diks et al. (2007), Verhoeff et al. (2006).

Starting Points: Looking Around

Take inspiration from real life.

Example: *Citizenship*

- For each country #i:
  - Live there for x_i years $\rightarrow$ gain citizenship
  - No visits in y_i years $\rightarrow$ lose citizenship
- For how many countries can you *simultaneously* hold citizenship?

Advantages:

- Can lead to highly original problems

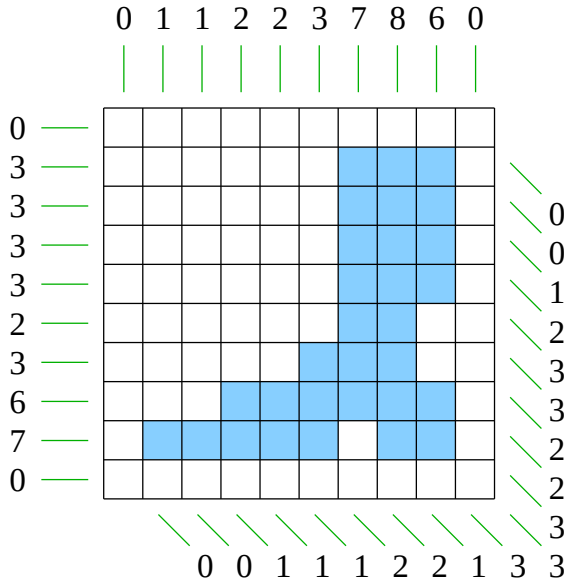
Disadvantages:

- Effort to find solutions
- Problems often NP-hard $\rightarrow$ need modification

Starting Points: Drawing on the Day Job

Draw ideas from research papers / problems in your work.

Example: *Giza*



Advantages:

- Solution already known in advance

Disadvantages:

- Can underestimate difficulty

Starting Points: Modifying a Known Algorithm

Modify a standard algorithm.

Examples:

- Begin with Dijkstra's algorithm for shortest paths
  $\rightarrow$ Allow edges that *reduce* total distance

Advantages:

- Easy to create tasks
- Solution is (partially) known in advance

Disadvantages:

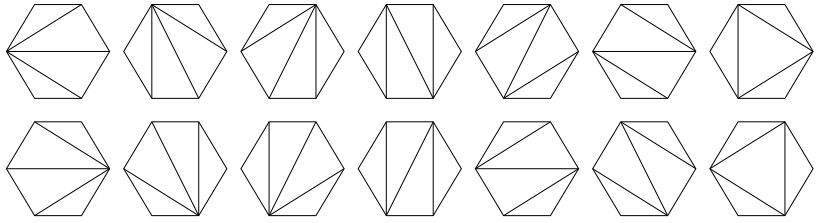
- Tasks often similar to well-known problems
- Difficult to create highly original problems

Starting Points: Borrowing from Mathematics

Draw ideas from mathematics, particularly *combinatorics*:

- Counting / arranging objects
- Naturally leads to *recurrence relations* $\longrightarrow$ dynamic programming

Example: Problems involving triangulations of polygons



Sources of combinatorial recurrence relations:

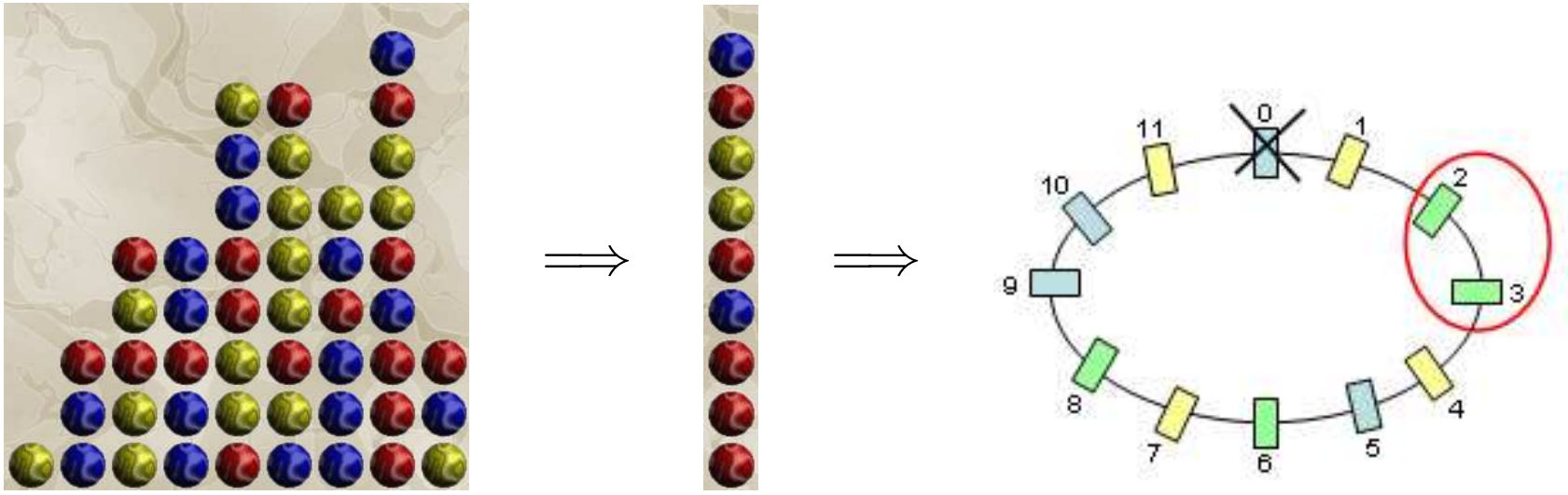
- Books on generating functions
- Online encyclopedia of integer sequences
- Mathematics competitions

Starting Points: Games and Puzzles

Draw ideas from games / puzzles (e.g., `rec.puzzles` newsgroup).

- *Directly:* Play the game / solve the puzzle
- *Indirectly:* Provides interesting objects and rules to play with

Example: *Collier de Bonbons*



Often NP-hard $\rightarrow$ need simplifications / constraints

Starting Points: Filling the Holes in a Domain

Aim to create new tasks in some field.

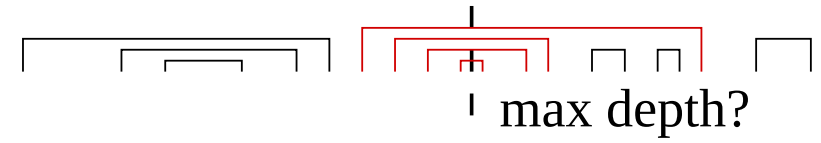
Examples: binary trees, sweep-line / sliding window algorithms

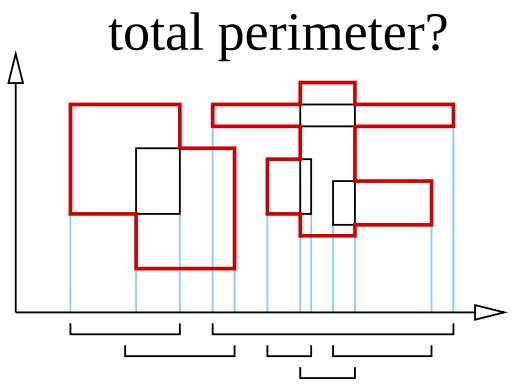
1. Make a list of “old” tasks in this field
2. List the *characteristics* of these tasks
3. Find a combination of characteristics that has not been used

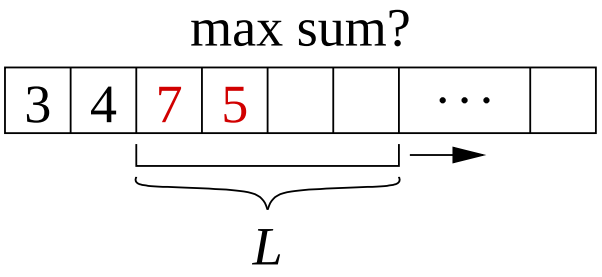
Examples of characteristics:

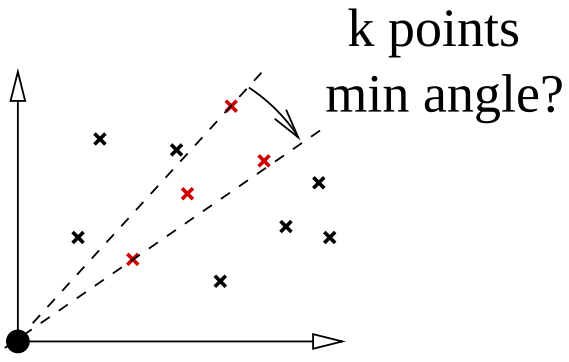
- Objects, attributes, relationships, constraints
- Type of question — existence? maximisation? aggregation?
- One question? Many questions? Interactive?

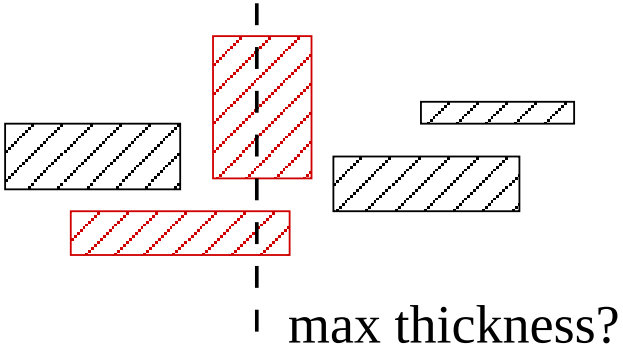
Starting Points: Filling the Holes in a Domain (ctd)







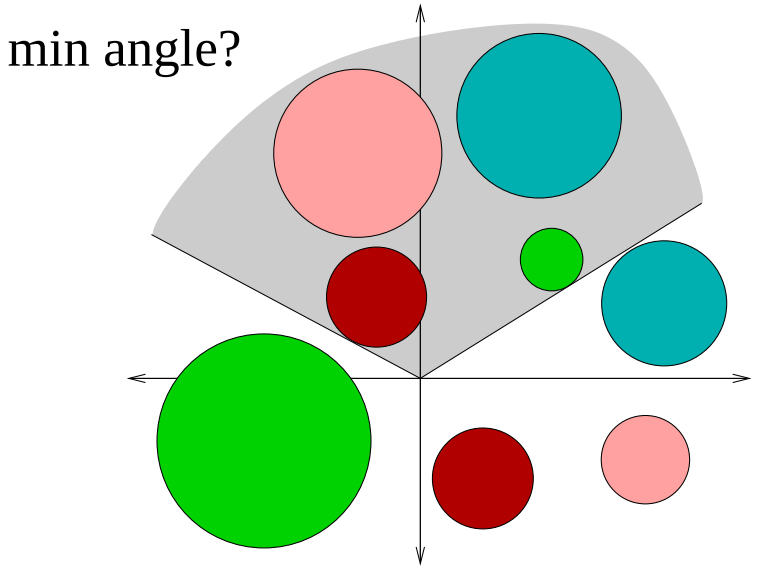




Different characteristics:

- find a point / interval
- among points / intervals
- given / from projections
- inclusions & intersections ok?

Starting Points: Filling the Holes in a Domain (ctd)



Final problem:

- find an *interval*
- among a different *set of intervals*
- obtained by projecting circles
- where inclusions & intersections are allowed

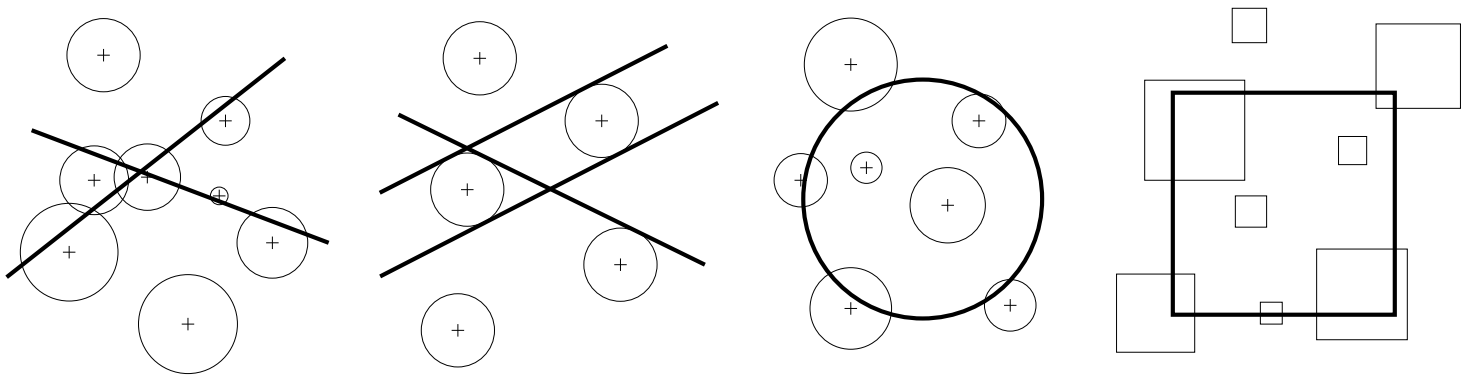
Starting Points: Building from Abstract Ideas

Draw vague sketchings and refine into a usable problem:

- Pick one or two types of objects and draw many of them
- Add some attributes
- Define relationships between objects
- Choose a question

The vague sketchings become more precise as you make decisions.

- Make changes (big then small) if you don’t like where it’s going
- Spend time looking for solutions



Improving the Task

The original idea seldom fits all criteria for a good task.

Tasks can be improved through successive *elementary changes*.

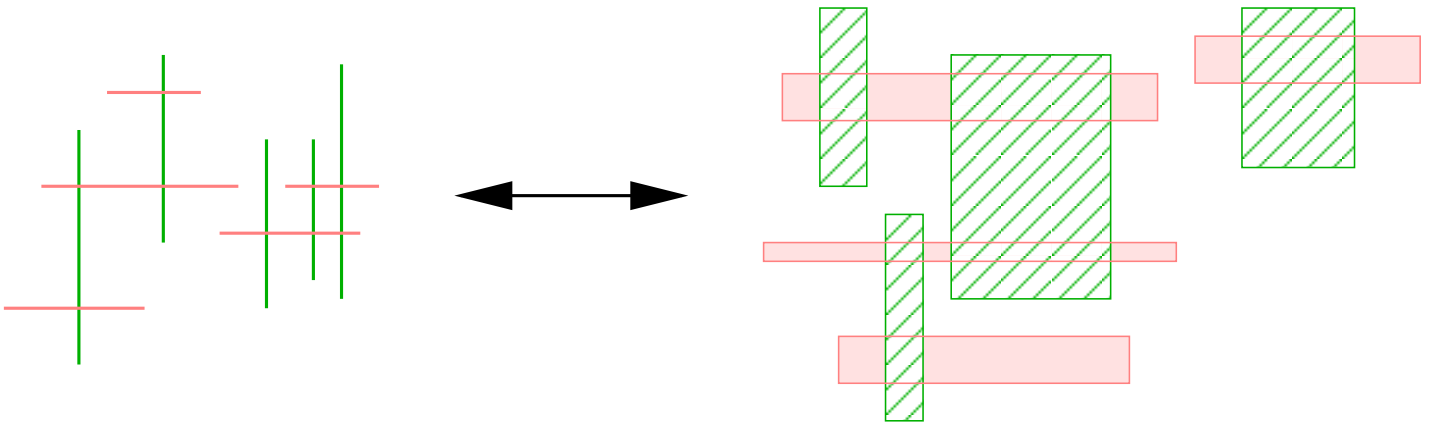
Specific ways to improve a task include:

- Changing the difficulty
- Adding or removing subproblems
- Aiming for a particular solution

Improving the Task: Changing the Difficulty

Ways of changing the difficulty:

- Add or remove dimensions

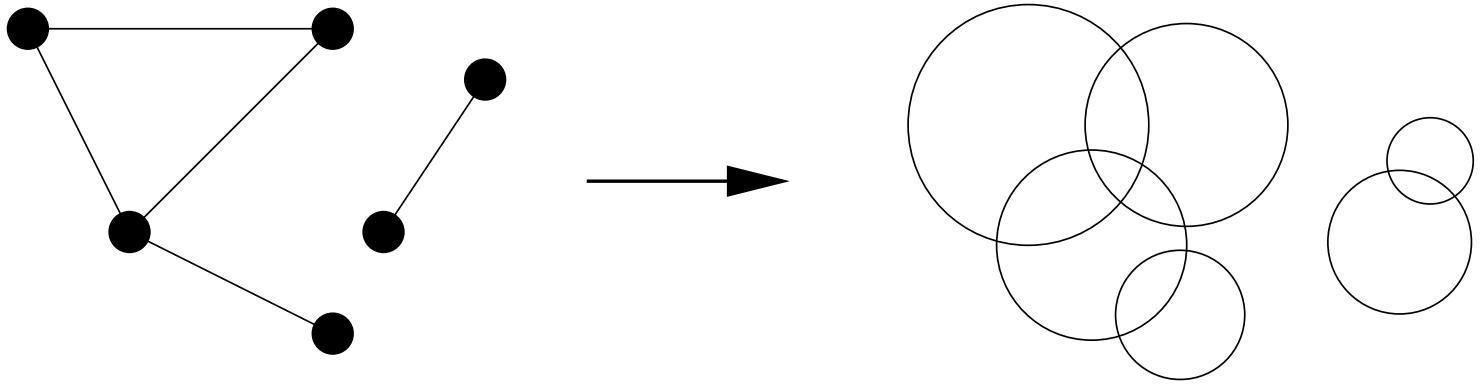


- Make the task dynamic $\rightarrow$ ask many questions
- Ask for a full list of steps instead of a single value
- Reduce the memory limit

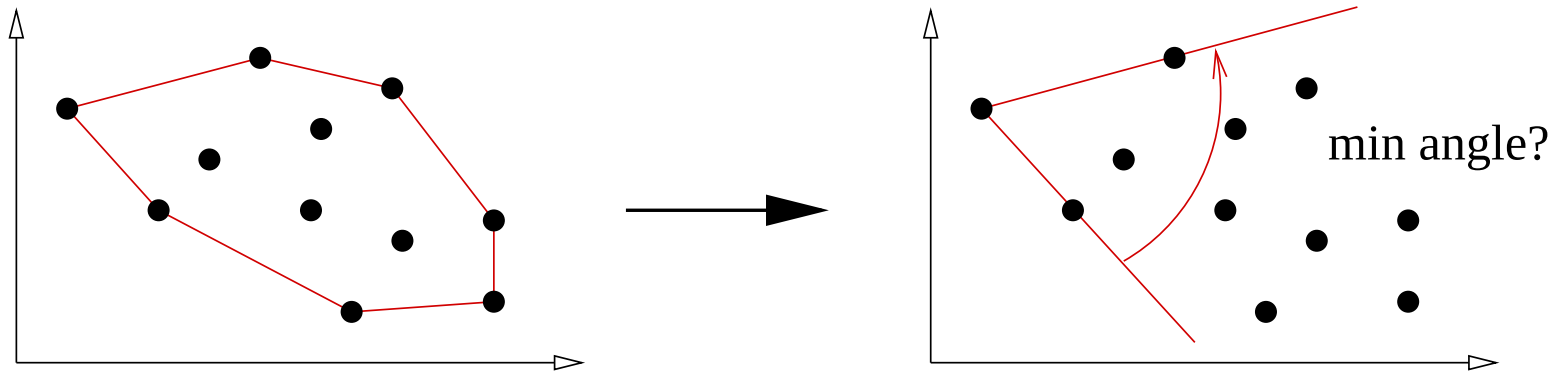
Improving the Task: Adding or Removing Subproblems

Adding a subproblem can make a task more original or more difficult:

- Transform the input to hide the true nature of the task



- Use the output of the task as the input for another problem



Improving the Task: Aiming for a Solution

Looking for a solution gives opportunities to improve a task:

- An idea does not work $\rightarrow$ change the task so that it does work
- Solution is too simple $\rightarrow$ change the task so that it does not work

When looking for a solution, do not give up too early!

Looking for solutions is the most important and time-consuming part of creating difficult and original tasks.