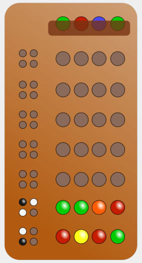
Algorithmic Analysis of Code-Breaking Games

Miroslav Klimoš, Prof. RNDr. Antonín Kučera Ph.D.
Faculty of Informatics, Masaryk University, Brno

Code-breaking games

- 2 players: *codemaker* and *codebreaker*
- codemaker selects a secret code
- codebreaker strives to reveal the code through a series of *experiments* whose outcomes give partial information about the code

Example: Mastermind



- Code: combination of *n coloured pegs*
- Codebreaker makes guesses (experiments)
- Guesses are evaluated with *black and white markers*
- Black marker = correct both color and position
- White marker = the color is present at different position

Example: Counterfeit coin

- Problem of identifying an odd-weight coin using balance scale
- Code: identity of the unique counterfeit coin
- Codebreaker puts coins on the balance scale and observes the outcome (left pan is lighter / heavier / same)

Questions and problems

- How should the codebreaker play in order to minimize the number of experiments needed to undoubtedly determine the code?
- Is there a strategy for experiment selection that guarantees revealing the code after at most *k* experiments?
- What strategy is optimal with respect to the average-case number of experiments, given that the code is selected from the given set with uniform distribution?

Challenges and solutions

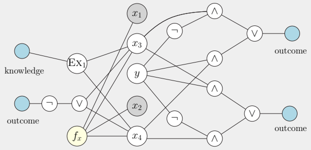
1. Create a general, formal model of code breking games
   - Model based on propositional logic
   - Secret code = valuation of variables
   - Partial information = logical formula

$$\begin{aligned}\Phi_t = \Big\{ & (f_x(\$1) \wedge \neg y) \vee (f_x(\$2) \wedge y), \\ & (f_x(\$1) \wedge y) \vee (f_x(\$2) \wedge \neg y), \\ & \neg f_x(\$1) \wedge \neg f_x(\$2) \ \Big\}\Big).\end{aligned}$$

2. Suggest general strategies for experiment selection
   - *“Select an experiment that minimizes the maximal number of possibilities for the code in the next round”*
   - Several strategies of this kind are formalized within the model

$$f(\Psi) = \frac{\sum_{\varphi \in \Psi} (\#\varphi)^2}{\sum_{\varphi \in \Psi} \#\varphi}$$

3. Propose algorithms for strategy evaluation and synthesis
   - Based on intelligent backtracking
   - Symmetry detection reduces the size of the state-space

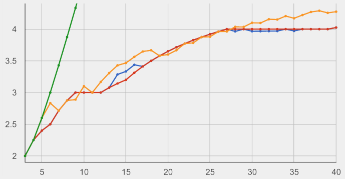


4. Design a computer language for game specification
   - Follows directly from the formal model
   - Built on top of Python for easier generation

```
for m in range(1, N//2 + 1):
  EXPERIMENT("weighing" + str(m), 2*m)
  OUTCOME("lighter", "((%s) & !y) ...
  OUTCOME("heavier", "((%s) & y)  ...
```

5. Implement proposed algorithms in a computer program
   - Command-line tool COBRA written in C++
   - Uses modern SAT solvers for satisfiability queries needed by the algorithms
   - Graph canonization tool Bliss is utilized for symmetry detection

6. Use the tool to create new and reproduce existing results
   - COBRA can easily reproduce some existing results for Mastermind
   - Allows easy analysis of generalizations and other code-breaking games


2
2.5
3
3.5
4
5
10
15
20
25
30
35
40