Implementing Fixed-Parameter Algorithms

Falk Hüffner

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin

20 November 2012
Real-world instances

Recommendation
Use real-world data.

- Easier to “sell”
- Analysis might lead to new insights and approaches
- More fun
Real-world instance sources

Databases

- Biological networks
- Social networks
Real-world instance sources

**Databases**
- Biological networks
- Social networks

**Web crawling**
- DBLP coauthor graph
- Song similarity graph (last.fm)
- Stock correlation graph
- Wikipedia inter-language links
Real-world instance sources

Databases
- Biological networks
- Social networks

Web crawling
- DBLP coauthor graph
- Song similarity graph (last.fm)
- Stock correlation graph
- Wikipedia inter-language links

Cooperation
- Statistics visualization
- Power line network
Parameters

- solution size
- distance from tractable instances (e.g. treewidth)
- structural parameters (e.g. vertex cover size)

Graphana (http://fpt.akt.tu-berlin.de/graphana/) is a tool that calculates parameters of graphs such as treewidth, connectivity, vertex cover size, cluster vertex deletion number, cluster editing number, $h$-index, degeneracy, feedback vertex set size, dominating set size, ...
Parameters

- solution size
- distance from tractable instances (e.g. treewidth)
- structural parameters (e.g. vertex cover size)

Advertisement

Graphana (http://fpt.akt.tu-berlin.de/graphana/) is a tool that calculates parameters of graphs such as treewidth, connectivity, vertex cover size, cluster vertex deletion number, cluster editing number, $h$-index, degeneracy, feedback vertex set size, dominating set size, ...
Randomly generated instances

Advantages of randomly generated instances

- Can have any number and size
- Can track relation between performance and parameters
Randomly generated instances

Advantages of randomly generated instances

- Can have any number and size
- Can track relation between performance and parameters

Types:
- Application oriented:
  - Phylogenetic trees
  - Web graphs
  - DNA sequences
- General
Example: Colorful Components

**COLORFUL COMPONENTS**

**Instance:** An undirected graph $G = (V, E)$ and a coloring of the vertices $\chi : V \rightarrow \{1, \ldots, c\}$.

**Task:** Delete a minimum number $k$ of edges such that all connected components are *colorful*, that is, they do not contain two vertices of the same color.

**Parameter:** $k$
Example: Colorful Components

**COLORFUL COMPONENTS**

**Instance:** An undirected graph $G = (V, E)$ and a coloring of the vertices $\chi : V \rightarrow \{1, \ldots, c\}$.

**Task:** Delete a minimum number $k$ of edges such that all connected components are *colorful*, that is, they do not contain two vertices of the same color.

**Parameter:** $k$

Random model:

- $c$: number of colors;
- $n$: number of vertices;
- $p_v$: probability that a component contains a vertex of a certain color;
- $p_e$: edge probability within component;
- $p_x$: edge probability between components.
A simple solver

Recommendation
Implement a solver that is as simple as possible.

Advantages
- first impression on what solutions look like
- base line for finding bugs

Typically simplest:
- Branching
- Integer Linear Programming (ILP)
Graph Bipartization: Find a minimum size set of vertices in a graph whose removal results in the graph being bipartite.
**Graph Bipartization**

*Graph Bipartization*: Find a minimum size set of vertices in a graph whose removal results in the graph being bipartite.
ILP for Graph Bipartization

c_1, \ldots, c_n : \text{binary variables} \quad (\text{cover})

s_1, \ldots, s_n : \text{binary variables} \quad (\text{color})
ILP for Graph Bipartization

c_1, \ldots, c_n : \text{binary variables} \quad (cover)

s_1, \ldots, s_n : \text{binary variables} \quad (color)

\text{minimize } \sum_{i=1}^{n} c_i
ILP for Graph Bipartization

\[ c_1, \ldots, c_n : \text{binary variables} \quad (\text{cover}) \]
\[ s_1, \ldots, s_n : \text{binary variables} \quad (\text{color}) \]

minimize \[ \sum_{i=1}^{n} c_i \]

s. t. \[ \forall \{v, w\} \in E : (s_v \neq s_w) \lor c_v \lor c_w \]
ILP for Graph Bipartization

\[ c_1, \ldots, c_n : \text{binary variables} \quad (\text{cover}) \]
\[ s_1, \ldots, s_n : \text{binary variables} \quad (\text{color}) \]

minimize \[ \sum_{i=1}^{n} c_i \]

s. t. \[ \forall \{v, w\} \in E : (s_v \neq s_w) \lor c_v \lor c_w \]

which can be expressed as an ILP constraint as

s. t. \[ \forall \{v, w\} \in E : s_v + s_w + (c_v + c_w) \geq 1 \]
\[ \forall \{v, w\} \in E : s_v + s_w - (c_v + c_w) \leq 1 \]
Implementation language

Recommendation

Use a high-level programming language.

Advantages

- more rapid development of typically exponential speedups, but only constant-factor slowdown
- persistent data structures allow simpler and less error-prone implementation of branching algorithms
Debugging

Recommendation

Verify that your solution is a solution.
Debugging

Recommendation
Verify that your solution is a solution.

Recommendation
Make automated tests that compare the result of the simple solver with that of the optimized solver, on randomly generated test cases.
Debugging

Recommendation
Verify that your solution is a solution.

Recommendation
Make automated tests that compare the result of the simple solver with that of the optimized solver, on randomly generated test cases.

Recommendation
Use a test case minimization tool (e.g. http://delta.tigris.org/).
Debugging

Recommendation
Verify that your solution is a solution.

Recommendation
Make automated tests that compare the result of the simple solver with that of the optimized solver, on randomly generated test cases.

Recommendation
Use a test case minimization tool (e.g. http://delta.tigris.org/).

Recommendation
Use version control.
Data reduction

Recommendation
Implement data reduction rules.

Advantages
- can be combined with approximation, heuristics, fixed-parameter, or other exact algorithms
- often very effective, even solve the whole instance
- normally, the more, the better
Heuristic speedups in branching algorithms:

- Heuristic branching priorities
- Lower bounds
Recommendation

Create a benchmark set using the randomized generator and parameter settings that match those measured in the real-world instances.
Create a benchmark set using the randomized generator and parameter settings that match those measured in the real-world instances.

- \( c \in \{3, 5, 8\} \),
- \( n \in \{60, 100, 170\} \),
- \( p_v \in \{0.4, 0.6, 0.9\} \),
- \( p_e \in \{0.4, 0.6, 0.9\} \),
- \( p_x \in \{0.01, 0.02, 0.04\} \).
Comparison of algorithms

Time measurements depend on
- Machine
- Compiler options
- Program name
- Weather
- ...

For exponential-time algorithms, averages of running time are useless.
Comparison of algorithms

Time measurements depend on
- Machine
- Compiler options
- Program name
- Weather
- ...

For exponential-time algorithms, averages of running time are useless.
Comparison of algorithms
Evaluating one algorithm

![Graphs showing time vs. parameters](image-url)
Evaluate solution quality

Solutions are optimal, but might not match real-world truth due to model deficiencies.
Evaluate solution quality

Cluster size vs. total number

- Blue: min-cut without DR
- Green: min-cut with DR
- Red: column generation
Finally

Recommendation

Make source and data available under a free license.