Implementation Aspects of Data Reduction

Falk Hüffner

Humboldt-Universität zu Berlin

3 September 2010
Kernelization

Reduces a problem in polynomial time to a decision-equivalent, provably smaller one

Data reduction rule

If applicable, reduces a problem in polynomial time to a smaller one, from whose solution an optimal solution to the original problem can be reconstructed.
Experiments with Data Reduction

- Many works, e.g. on Linear Programming, SAT, Steiner Tree
Experiments with Data Reduction

- Many works, e.g. on Linear Programming, SAT, Steiner Tree
- But few systematic studies on general NP-hard problems in the parameterized context
Case Studies

Dominating Set on random planar graphs, \( n \in \{500, 1500, 4000\} \)

- Kernel size: 67k
Case Studies

Dominating Set on random planar graphs, \( n \in \{500, 1500, 4000\} \)

- Kernel size: 67\(k\)
- With kernelization rules: \(\approx 70\%\) of vertices removed
Case Studies

Dominating Set on random planar graphs, \( n \in \{500, 1500, 4000\} \)


- Kernel size: 67k
- With kernelization rules: \( \approx 70\% \) of vertices removed
- With additional, “theoretically powerless” rules: \( \approx 99\% \) of vertices removed

Internet graphs: 99.9% removed
Dominating Set on random planar graphs, 
\( n \in \{500, 1500, 4000\} \)


- Kernel size: 67k
- With kernelization rules: \( \approx 70\% \) of vertices removed
- With additional, “theoretically powerless” rules: \( \approx 99\% \) of vertices removed
- Internet graphs: 99.9% removed
Case Studies

Dominating Set on random planar graphs, \( n \in \{500, 1500, 4000\} \)

- Kernel size: 67\(k\)
- With kernelization rules: \(\approx 70\%\) of vertices removed
- With additional, “theoretically powerless” rules: 
  \(\approx 99\%\) of vertices removed
- Internet graphs: 99.9\% removed
Case Studies

Dominating Set on random planar graphs, \( n \in \{500, 1500, 4000\} \)


- Kernel size: 67k
- With kernelization rules: \( \approx 70\% \) of vertices removed
- With additional, “theoretically powerless” rules: \( \approx 99\% \) of vertices removed
- Internet graphs: 99.9\% removed

**Lesson**

Try all reduction rules, independent of proven effect.
Case Studies (II)

Solve Clique as Vertex Cover on the complement graph \((n \approx 1000)\)

[Abu-Khzam et al., ALENEX ’04]

- 3 global reductions: LP, flow, crown reduction
Case Studies (II)

Solve Clique as Vertex Cover on the complement graph \( (n \approx 1000) \)
[Abu-Khzam et al., ALENEX ’04]

- 3 global reductions: LP, flow, crown reduction
- Times vary from 0.07 s to 1 h
Case Studies (II)

Solve Clique as Vertex Cover on the complement graph ($n \approx 1000$)

[Abu-Khzam et al., ALENEX ’04]

- 3 global reductions: LP, flow, crown reduction
- Times vary from 0.07 s to 1 h
- Between 0 % and 100 % of edges removed
Case Studies (II)

Solve Clique as Vertex Cover on the complement graph \((n \approx 1000)\)

[Abu-Khzam et al., ALENEX '04]

- 3 global reductions: LP, flow, crown reduction
- Times vary from 0.07 s to 1 h
- Between 0 % and 100 % of edges removed
Case Studies (II)

Solve Clique as Vertex Cover on the complement graph ($n \approx 1000$)

[Abu-Khzam et al., ALENEX ’04]

- 3 global reductions: LP, flow, crown reduction
- Times vary from 0.07 s to 1 h
- Between 0 % and 100 % of edges removed

Lesson

Try cheapest rules first.
Solve Clique as Vertex Cover on the complement graph $(n \approx 1000)$

[Abu-Khzam et al., ALENEX ’04]

- 3 global reductions: LP, flow, crown reduction
- Times vary from 0.07 s to 1 h
- Between 0 % and 100 % of edges removed

**Lesson**

Try cheapest rules first.

**Vertex Cover on planar graphs** [Alber, Dorn & Niedermeier, Discrete Appl. Math.]

- 60-70 % reduction
Case Studies (III)

Cluster Editing [Böcker, Briesemeister & Klau, Algorithmica ’10]

- Parameter-dependent reduction rules
Case Studies (III)

Cluster Editing [Böcker, Briesemeister & Klau, Algorithmica '10]

- Parameter-dependent reduction rules
Case Studies (III)

Cluster Editing [Böcker, Briesemeister & Klau, Algorithmica ’10]

- Parameter-dependent reduction rules

Where to take $k$ from?

- Upper bound (e.g. from heuristic solution)
Cluster Editing [Böcker, Briesemeister & Klau, Algorithmica ’10]

- Parameter-dependent reduction rules

Where to take $k$ from?

- Upper bound (e.g. from heuristic solution)
- Try them increasingly
Case Studies (III)

Cluster Editing [Böcker, Briesemeister & Klau, Algorithmica ’10]

- Parameter-dependent reduction rules

Where to take $k$ from?

- Upper bound (e.g. from heuristic solution)
- Try them increasingly
Case Studies (III)

Cluster Editing [Böcker, Briesemeister & Klau, Algorithmica ’10]

- Parameter-dependent reduction rules

Where to take $k$ from?

- Upper bound (e.g. from heuristic solution)
- Try them increasingly

**Lesson**

Consider using parameter-dependent reduction rules.
Case Studies (III)

Lesson

Consider solving a harder problem than the one you need to solve.
Case Studies (III)

Lesson

Consider solving a harder problem than the one you need to solve.
Implementing Data Reduction

Claim
Since data reduction is polynomial, but solving is exponential, running time for reduction does not matter much.
Claim

Since data reduction is polynomial, but solving is exponential, running time for reduction does not matter much.

[Dehne et al., IWPEC '06]
Implementing Data Reduction

Problem
In a branching, we still need the unmodified graph.
Implementing Data Reduction

Problem
In a branching, we still need the unmodified graph.

Possible solutions
1. Copy whole graph in each step
Implementing Data Reduction

Problem
In a branching, we still need the unmodified graph.

Possible solutions
1. Copy whole graph in each step
2. Use a persistent data structure
Implementing Data Reduction

Problem
In a branching, we still need the unmodified graph.

Possible solutions
1. Copy whole graph in each step
2. Use a persistent data structure
3. Use an “undo” function for each branch or reduction that undos all changes.
Persistent data structures

Definition

A persistent data structure is a data structure which always preserves the previous version of itself when it is modified.
Persistent data structures

Definition

A *persistent data structure* is a data structure which always preserves the previous version of itself when it is modified.

In purely functional programming languages, all data structures are persistent.
Persistent data structures

Definition

A *persistent data structure* is a data structure which always preserves the previous version of itself when it is modified.

In purely functional programming languages, all data structures are persistent.

E. g. persistent big-endian Patricia trees:

- $O(\log n + \deg v)$ neighborhood enumeration
- $O(\log n)$ edge test
- $O(\log n)$ edge insertion/deletion
- $O(\log n)$ vertex insertion
- $O(\log n \deg v)$ vertex deletion
Persistent data structures

Advantages
- No linear copy overhead
- Very easy to implement
- Little error prone
- Quick and easy operations like intersection of neighbor sets

Disadvantages
- Logarithmic overhead on all operations
## Implicit undo data structures

### Case Studies

**Implementation issues**

**Implicit undo data structures**

[Abu-Khzam, Langston, Mouawad & Nolan, FAW ’10]

- **IM**
  
  \[
  \begin{bmatrix}
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
  \hline
  0 & -1 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\
  1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
  2 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
  3 & 2 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
  4 & -1 & -1 & -1 & 1 & -1 & 1 & 0 & -1 \\
  5 & -1 & -1 & -1 & 2 & 1 & -1 & -1 & 0 \\
  6 & -1 & -1 & -1 & -1 & 2 & -1 & -1 & -1 \\
  7 & -1 & -1 & -1 & -1 & 2 & -1 & -1 & -1 \\
  \end{bmatrix}
  \]

- **DEGREE**
  
  \[
  \begin{bmatrix}
  0 & 3 \\
  1 & 1 \\
  2 & 1 \\
  3 & 3 \\
  4 & 3 \\
  5 & 3 \\
  6 & 1 \\
  7 & 1 \\
  \end{bmatrix}
  \]

- **AL**
  
  \[
  \begin{bmatrix}
  1 & 2 & 3 \\
  1 & 0 \\
  2 & 0 \\
  3 & 0 & 4 & 5 \\
  4 & 3 & 5 & 6 \\
  5 & 3 & 4 & 7 \\
  6 & 4 \\
  7 & 5 \\
  \end{bmatrix}
  \]

- **LIST**
  
  \[
  0 1 2 3 4 5 6 7
  \]

- **IDXLIST**
  
  \[
  [0 1 2 3 4 5 6 7]
  \]

[Abu-Khzam, Langston, Mouawad & Nolan, FAW ’10]
Implicit undo data structures

- $O(\text{deg } v)$ neighborhood enumeration
- $O(1)$ edge test
- $O(1)$ edge insertion/deletion
- $O(1)$ vertex insertion
- $O(\text{deg } v)$ vertex deletion

Advantages
- Very little time overhead
- 5–10 times faster than simple adjacency list

Disadvantages
- Large memory overhead
- Nontrivial graph modifications (e.g., edge contraction) become complicated
Caching

Example

Keep a sorted map from vertex degree to the list of vertices of that degree.
**Example**

Keep a sorted map from vertex degree to the list of vertices of that degree.

**Problem**

Need to find edges whose common neighbors induce a clique.
Caching

Example
Keep a sorted map from vertex degree to the list of vertices of that degree.

Problem
Need to find edges whose common neighbors induce a clique.

Solution
Record for each edge \( \{u, v\} \) the number of edges in the graph \( G[N(u) \cap N(v)] \), using a priority queue.

[Gramm, Guo, Hüffner & Niedermeier, ACM J. Exp. Algorithmics ’08]
Definition

*Data reduction* is polynomial-time preprocessing of instances of NP-hard problems that allows retrieving an optimal solution.
Model extensions

Definition

*Data reduction* is polynomial-time preprocessing of instances of NP-hard problems that allows retrieving an optimal solution.

Result

When not using branching, even preprocessing that is not provably polynomial-time can help.

[Hüffner, Betzler & Niedermeier, J. Comb. Optim. ’09]
Definition

*Data reduction* is polynomial-time preprocessing of instances of NP-hard problems that allows retrieving an *optimal* solution.

Result

When not using branching, even preprocessing that is not provably polynomial-time can help.

[Hüffner, Betzler & Niedermeier, J. Comb. Optim. '09]

Result

Using non-optimality-preserving data reductions, a “kernel” guaranteeing approximation factor 1.5 can be found for Vertex Cover.

[Asgeirsson & Stein, WEA ’07]
Outlook

- Order of data reduction rules
Outlook

- Order of data reduction rules
- Graph data reduction language and framework
Outlook

- Order of data reduction rules
- Graph data reduction language and framework
- Data reduction and enumeration