Algorithm Design: Simplicity

Falk Hüffner

Friedrich-Schiller-Universität Jena

GI-Dagstuhl Research Seminar 06362: Algorithm Engineering
September 2006

Outline

1. What is simplicity?
2. Advantages of simplicity for implementation
3. How to achieve simplicity?
   - Modularization
   - General-purpose modelers
   - Trade off guaranteed performance
   - Trade off guaranteed correctness
4. Effects on analysis
   - Example: Exponential-time algorithms

Outline

1. What is simplicity?
2. Advantages of simplicity for implementation
3. How to achieve simplicity?
   - Modularization
   - General-purpose modelers
   - Trade off guaranteed performance
   - Trade off guaranteed correctness
4. Effects on analysis
   - Example: Exponential-time algorithms

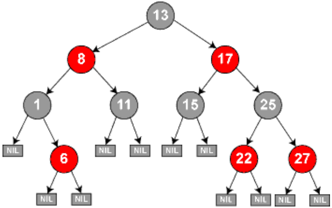
Definition of Simplicity

Definition attempt

An algorithm is simple if it is concise to write down and easy to grasp.

Red-black trees—a complicated data structure?

Red-black trees are balanced binary search trees used to implement associative arrays.



To describe the "insert" function of red-black-trees, Cormen et al. [Introduction to Algorithms, 2001] use 14 pages and about 57 lines of pseudocode. They differentiate nine cases and five different actions for balancing.

Red-black trees—a complicated data structure?

A Haskell implementation [Okasaki, J. Functional Programming '99]:

```
data Color = R | B
data Tree elt = E | T Color (Tree elt) elt (Tree elt)

balance B (T R (T R a x b) y c) z d
     || B (T R a x (T R b y c)) z d
     || B a x (T R (T R b y c) z d)
     || B a x (T R b y (T R c z d)) = T R (T B a x b) y (T B c z d)
balance color a x b = T color a x b

insert x s = makeBlack (ins s)
  where ins E = T R E x E
        ins (T color a y b) | x <  y = balance color (ins a) y b
                            | x == y = T color a y b
                            | x >  y = balance color a y (ins b)
        makeBlack (T _ a y b) = T B a y b
```

Where did the complexity go?

- Algebraic data types
- Pattern matching
- No “optimizations”

. . . but it’s the same algorithm!

Definition of Simplicity?

"Simplicity" of an algorithm is affected by "cultural" factors:

- Means of presentation (notation, assumptions, ...)
- Previous knowledge of the reader.

We will do with the intuitive concept of simplicity.

Outline

Advantages of simplicity for implementation

- quicker to implement
- fewer bugs
- reduced effort for testing
- maintainability: easier to understand and debug
- flexibility: adaption to changing specifications
- employment in resource constrained environments

Advantages of simplicity for implementation

- quicker to implement
- fewer bugs
- reduced effort for testing
- maintainability: easier to understand and debug
- flexibility: adaption to changing specifications
- employment in resource constrained environments

Example

The Advanced Encryption Standard (AES) process, which aimed to find a new standard block cipher, required "algorithm simplicity" as one of the three major algorithm characteristics.

Infeasibly complicated algorithms

Lack of simplicity can make implementation infeasible.

Example

Algorithm for four-coloring planar graphs

[Robertson, Sanders, Seymour&Thomas, STOC '96]

- finds one of 633 "configurations" (subgraphs),
- then applies one of 32 "discharging rules" to eliminate it.

Only algorithm for four-coloring planar graphs, but never implemented.

Really infeasible?

Sometimes, algorithms initially dismissed as too complicated sometimes still find applications.

Example

Fibonacci heaps: Priority queue data structure

- "[...] predominantly of theoretical interest."
  [Cormen et al., Introduction to Algorithms, 2001]
- "[...] sufficiently complicated that you shouldn't mess with them unless you really know what you are doing."
  [Skiena, Algorithm Design Manual, 1998]
- implemented in the widely-used GNU compiler collection (gcc)

Outline

1. What is simplicity?
2. Advantages of simplicity for implementation
3. How to achieve simplicity?
   - Modularization
   - General-purpose modelers
   - Trade off guaranteed performance
   - Trade off guaranteed correctness
4. Effects on analysis
   - Example: Exponential-time algorithms

Modularization

Idea

Impose a hierarchical structure: Decompose the problem into several parts with a narrow intersection, which can then independently designed and understood, and be further subdivided.

Well-known from software engineering.

Example

Task: Based on a packed-based network protocol where packets might get lost or arrive out-of order (IP), design a protocol for serving web pages (http).
Idea: First design an intermediate protocol that provides reliable streaming connections (TCP).

Modularization

Example

Compilers are usually divided into a *lexing*, a *parsing*, and a *translation* phase.
The translation phase is usually broken down further; for example gcc chains more than 100 separate optimization passes.

Modularization

Example

Compilers are usually divided into a *lexing*, a *parsing*, and a *translation* phase.
The translation phase is usually broken down further; for example gcc chains more than 100 separate optimization passes.

Example

Computational geometry tasks: Let S be a set of n $\{$points, line segments, ...$\}$ in the plane.
Idea: First sort by $\{x$-coordinate, slope, ...$\}$.

Use standard algorithm design schemes

- divide&conquer
- dynamic programming
- greedy
- branch&bound
- sweepline
- ...

Use standard algorithm design schemes

- divide&conquer
- dynamic programming
- greedy
- branch&bound
- sweepline
- ...

Advantages:

- simpler to grasp
- exploit existing experience with analysis
- exploit existing experience with implementation

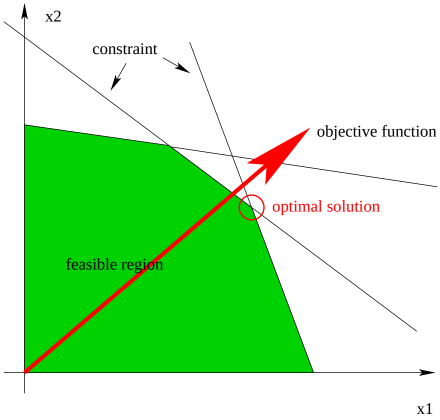
General-purpose modelers

Some general models have been successfully used to solve a wide range of problems:

- linear programs (LPs)
- integer linear programs (ILPs)
- constraint satisfaction problems (CSPs)
- boolean satisfiability problems (SAT)

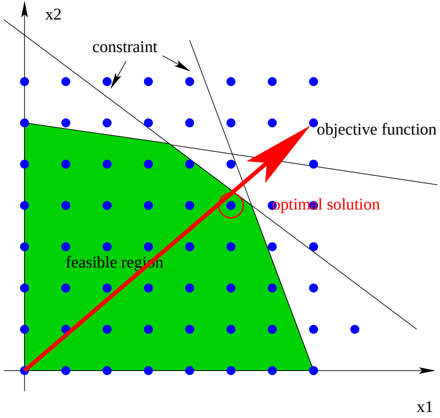
Linear Programming

LP solvers optimize a linear function of a real vector under linear constraints



Linear Programming

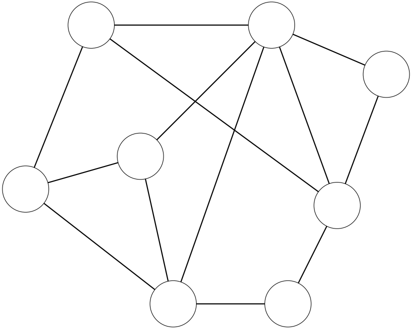
LP solvers optimize a linear function of a real vector under linear constraints



ILPs add the possibility of requiring coefficients to be integral

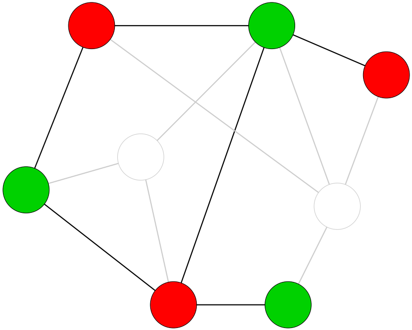
Graph Bipartization

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

$c_1, \ldots, c_n$: binary variables *(cover)*

$s_1, \ldots, s_n$: binary variables *(color)*

ILP for Graph Bipartization

$c_1, \ldots, c_n$: binary variables *(cover)*

$s_1, \ldots, s_n$: binary variables *(color)*

$$\text{minimize} \sum_{i=1}^{n} c_i$$

ILP for Graph Bipartization

$c_1, \ldots, c_n$: binary variables *(cover)*

$s_1, \ldots, s_n$: binary variables *(color)*

$$\text{minimize} \sum_{i=1}^{n} c_i$$

$$\text{s. t. } \forall \{v, w\} \in E : (s_v \neq s_w) \vee c_v \vee c_w$$

ILP for Graph Bipartization

$c_1, \ldots, c_n$: binary variables *(cover)*

$s_1, \ldots, s_n$: binary variables *(color)*

$$\text{minimize} \sum_{i=1}^{n} c_i$$

$$\text{s. t. } \forall \{v, w\} \in E : (s_v \neq s_w) \vee c_v \vee c_w$$

which can be expressed as an ILP constraint as

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

ILP for Graph Bipartization

$c_1, \dots, c_n$: binary variables *(cover)*

$s_1, \dots, s_n$: binary variables *(color)*

$$\text{minimize} \sum_{i=1}^{n} c_i$$

$$\text{s. t. } \forall \{v, w\} \in E : (s_v \neq s_w) \vee c_v \vee c_w$$

which can be expressed as an ILP constraint as

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

When solved by GNU GLPK, faster than a 2600 lines of code problem-specific branch&bound-algorithm

Randomization

"Simplicity [. . .] is the first and foremost reason for using randomized algorithms." [Gupta, Lecture Notes, 2004]

By using nondeterminism and

- accepting a small chance of a high runtime (*Las Vegas algorithms*), or
- accepting a small chance of a nonoptimal output (*Monte Carlo algorithms*),

one can often obtain algorithms that are much simpler than deterministic algorithms.

Quicksort

- Quicksort works by selecting an element as *pivot*, dividing the elements into those smaller than the pivot and those larger than the pivot, and then recursively sorting these subsets.
- Quicksort performs very well, except when the choice of the pivot repeatedly divides the subsequence into parts of very unequal size.
- Even elaborate pivot choice schemes like "median-of-three" cannot eliminate this problem.

Randomized quicksort

- Choose a *random* pivot!
- *expected* runtime $\Theta(n \log n)$
- Disadvantage: with a small probability, the algorithm takes much longer than expected.

Randomized quicksort

- Choose a *random* pivot!
- *expected* runtime $\Theta(n \log n)$
- Disadvantage: with a small probability, the algorithm takes much longer than expected.

Definition

An algorithm employing randomness that always produces a correct result, but carries a small probability of using more resources than expected, is called a *Las Vegas algorithm*.

Randomized quicksort

- Choose a *random* pivot!
- *expected* runtime $\Theta(n \log n)$
- Disadvantage: with a small probability, the algorithm takes much longer than expected.

Definition

An algorithm employing randomness that always produces a correct result, but carries a small probability of using more resources than expected, is called a *Las Vegas algorithm*.

Las Vegas algorithms can often be used to avoid excessive resource usage on corner case inputs, while retaining simplicity.

Min-Cut

Min-Cut: Find a minimum size set of edges in a graph whose removal results in the graph being broken into two or more components.

Randomized approach:

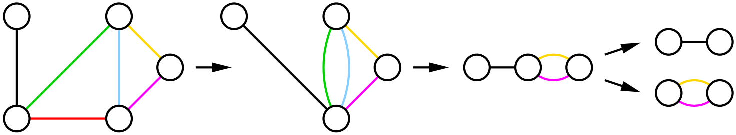
- Pick a random edge and merge its two endpoints.
- Remove all self-loops (but not multiple edges between two vertices).
- Repeat until only two vertices remain.
- The edges between these vertices then form a candidate min-cut.

Min-Cut

Min-Cut: Find a minimum size set of edges in a graph whose removal results in the graph being broken into two or more components.

Randomized approach:

- Pick a random edge and merge its two endpoints.
- Remove all self-loops (but not multiple edges between two vertices).
- Repeat until only two vertices remain.
- The edges between these vertices then form a candidate min-cut.



Randomized algorithm for Min-Cut

- The error probability can be made arbitrarily small by repeating.
- The algorithm is much simpler than deterministic algorithms.
- A variant is also significantly faster.

Randomized algorithm for Min-Cut

- The error probability can be made arbitrarily small by repeating.
- The algorithm is much simpler than deterministic algorithms.
- A variant is also significantly faster.

Definition

An algorithm employing randomness that is always fast, but carries a small probability of producing a nonoptimal solution, is called a *Monte Carlo algorithm*.

Randomized algorithm for Min-Cut

- The error probability can be made arbitrarily small by repeating.
- The algorithm is much simpler than deterministic algorithms.
- A variant is also significantly faster.

Definition

An algorithm employing randomness that is always fast, but carries a small probability of producing a nonoptimal solution, is called a *Monte Carlo algorithm*.

Monte Carlo algorithms can often be significantly simpler and faster than deterministic algorithms.

Outline

Simple algorithm – simple analysis?

Intuitively, a simpler algorithm should be simpler to analyze for performance measures such as worst-case runtime, memory use, or solution quality.

Simple algorithm – simple analysis?

Intuitively, a simpler algorithm should be simpler to analyze for performance measures such as worst-case runtime, memory use, or solution quality.

Example

Vertex Cover: Given a graph, find a subset of its vertices such that every edge has at least one endpoint in the subset.

Simple algorithm – simple analysis?

Intuitively, a simpler algorithm should be simpler to analyze for performance measures such as worst-case runtime, memory use, or solution quality.

Example

Vertex Cover: Given a graph, find a subset of its vertices such that every edge has at least one endpoint in the subset.

Simple greedy strategy: repeatedly choose some edge, take *both* endpoints into the cover, and then deletes them from the graph. Clearly, the solution is at most twice as large as an optimal one.

Shortest Common Superstring

Example

Shortest Common Superstring: given a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of strings, find the shortest string that contains each element of $\mathcal{S}$ as a contiguous substring.

Shortest Common Superstring

Example

Shortest Common Superstring: given a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of strings, find the shortest string that contains each element of $\mathcal{S}$ as a contiguous substring.

Simple greedy strategy: repeatedly merges the two strings with the largest *overlap*, until only one string remains. (The overlap of two strings A and B is the longest string that is both a suffix of A and a prefix of B).

`TCAGAGGC GGCAGAAG AAGTTCAG AAGTTGGG`

`AAGTTCAGAGGC GGCAGAAG AAGTTGGG`

Shortest Common Superstring

Example

Shortest Common Superstring: given a set $\mathcal{S} = \{S_1, \dots, S_n\}$ of strings, find the shortest string that contains each element of $\mathcal{S}$ as a contiguous substring.

Simple greedy strategy: repeatedly merges the two strings with the largest *overlap*, until only one string remains. (The overlap of two strings A and B is the longest string that is both a suffix of A and a prefix of B).

`TCAGAGGC GGCAGAAG AAGTTCAG AAGTTGGG`

`AAGTTCAGAGGC GGCAGAAG AAGTTGGG`

`AAGTTCAGAGGC GGCAGAAG AAGTTGGG`

`GGCAGAAGTTCAGAGGC AAGTTGGG`

Shortest Common Superstring

Example

Shortest Common Superstring: given a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of strings, find the shortest string that contains each element of $\mathcal{S}$ as a contiguous substring.

Simple greedy strategy: repeatedly merges the two strings with the largest *overlap*, until only one string remains. (The overlap of two strings A and B is the longest string that is both a suffix of A and a prefix of B).

```
TCAGAGGC GGCAGAAG AAGTTCAG AAGTTGGG
AAGTTCAGAGGC GGCAGAAG AAGTTGGG
AAGTTCAGAGGC GGCAGAAG AAGTTGGG
GGCAGAAGTTCAGAGGC AAGTTGGG
GGCAGAAGTTCAGAGGC AAGTTGGG
AAGTTGGGCAGAAGTTCAGAGGC
```

Greedy for Shortest Common Superstring

How good is the greedy algorithm?

- No example is known where solution is more than twice as long as an optimal one.
- Conjecture: factor 2 is the worst case
- Only an upper bound of 3.5 has been proven
  [Kaplan&Shafrir, IPL '05]
- The currently "best" algorithm provides factor 2.5
  [Sweedyk, SIAM J. Comput. '99]

Greedy for Shortest Common Superstring

How good is the greedy algorithm?

- No example is known where solution is more than twice as long as an optimal one.
- Conjecture: factor 2 is the worst case
- Only an upper bound of 3.5 has been proven
  [Kaplan&Shafrir, IPL '05]
- The currently "best" algorithm provides factor 2.5
  [Sweedyk, SIAM J. Comput. '99]

Suspicion: Are we only improving analyzability instead of performance, at the cost of simplicity?

Exponential-time algorithms

- In one line of research, increasingly complicated exponential-time algorithm for NP-hard problems were developed
- Progress based on case distinctions
- Experiments often did not show speedups
- A new method of analyzing the recurrences involved by Eppstein [SODA '04] allowed to show an algorithm for DOMINATING SET actually runs in $O(2^{0.598n})$ instead of $O(2^{0.850n})$ [Fomin et al. ICALP '05].

Simplicity and Analysis

How to avoid introducing unneccessary complexity that only improves analyzability?

- Experiments
- Proving lower bounds
- Improving the algorithm analysis tools

Summary

- Simplicity is a valuable property of an algorithm.
- Techniques to achieve simplicity:
  - Modularization
  - Modeling tools
  - Randomization
- One should be wary of sacrificing simplicity for what might only be analyzability.