

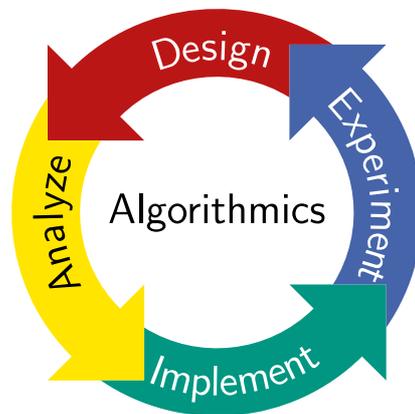
# Algorithm Engineering

Peter Sanders · Dorothea Wagner

Received: date / Accepted: date

**Zusammenfassung** bleibt leer

**Schlüsselwörter** keine keywords



## 1 Einleitung

Effiziente Algorithmen und Datenstrukturen sind eine Grundvoraussetzung für anspruchsvolle Computeranwendungen. Algorithmik – die systematische

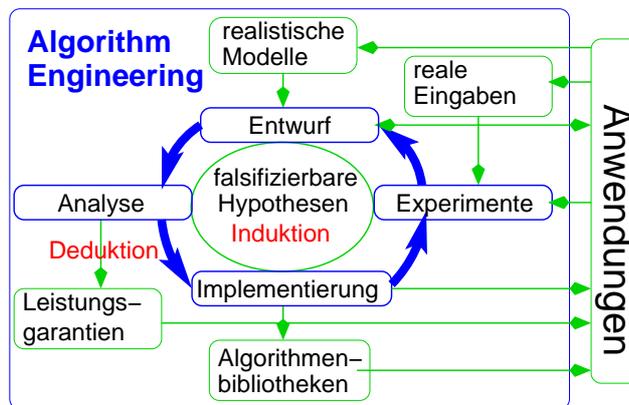
---

Teilweise unterstützt durch die DFG im Rahmen des Schwerpunktprogramms SPP 1307 Algorithm Engineering unter SA 933/4-3 und WA 654/15-3.

Karlsruher Institut für Technologie  
Tel.: +49-721-608-47580  
Fax: +49-721-608-43088  
E-Mail: {sanders,dorothea.wagner}@kit.edu

Entwicklung effizienter Algorithmen – ist deshalb in etlichen Bereichen entscheidend für die Umsetzung technologischer Möglichkeiten und somit von großer Bedeutung für Technik, Wirtschaft, Wissenschaft und unser tägliches Leben.

Traditionell hat sich die Algorithmik der Methodik der *Algorithmentheorie* bedient, die aus der Mathematik stammt: Algorithmen werden für einfache und abstrakte Problem- und Maschinenmodelle entworfen. Hauptergebnis sind dann beweisbare Leistungsgarantien für beliebige Eingaben. Dieser Ansatz führt in vielen Fällen zu eleganten, zeitlosen Lösungen, die sich an viele Anwendungen anpassen lassen. Die harten Leistungsgarantien ergeben zuverlässig eine hohe Effizienz und gute Lösungsqualität auch für zur Implementierungszeit unbekannte Typen von Eingaben. Auswahl und Implementation eines Algorithmus ist aus Sicht der Algorithmentheorie Teil der Anwendungsentwicklung. Nach allgemeiner Beobachtung ist diese Art des Ergebnistransfers aber ein sehr langsamer Prozess. Bei wachsenden Anforderungen an innovative Algorithmen ergeben sich daraus immer größere Lücken zwischen Theorie und Praxis: Reale Hardware entwickelt sich durch Parallelismus, Pipelining, Speicherhierarchien u.s.w. immer weiter weg von einfachen Maschinenmodellen. Anwendungen werden immer komplexer. Gleichzeitig entwickelt die Algorithmentheorie immer ausgeklügeltere Algorithmen, die zwar wichtige Ideen enthalten aber oft kaum implementierbar sind. Außerdem haben reale Eingaben oft wenig mit den worst-case Szenarien der theoretischen Analyse zu tun. Im Extremfall werden viel versprechende algorithmische Ansätze vernachlässigt, weil eine vollständige Analyse mathematisch zu schwierig wäre.



**Abb. 1** Algorithm Engineering als Kreislauf aus Entwurf, Analyse, Implementierung und experimenteller Analyse.

Algorithm Engineering ist eine Methodik für die Entwicklung und Erforschung von Algorithmen, die dazu beiträgt die oben beschriebenen Lücken zwischen Theorie und Praxis zu überwinden. Algorithm Engineering vereint

theoretische und experimentelle Ansätze und erlaubt eine enge Kopplung an Anwendungen. Kern des Algorithm Engineering ist ein Kreislauf aus Entwurf, Analyse, Implementierung und experimenteller Bewertung von Algorithmen. Hinzu kommt eine stärkere Einbindung von Anwendungen und realistischen Maschinenmodellen. Implementierungen können direkt oder indirekt (über wiederverwendbare Algorithmenbibliotheken) in Anwendungen einfließen. Umgekehrt liefern Anwendungen verfeinerte Modelle für den Algorithmenentwurf sowie Szenarien für realistische Experimente. Abbildung 1 zeigt ein Schema dieses Ansatzes. Die einzelnen Aspekte werden in den folgenden Abschnitten diskutiert.

## 2 Algorithm Engineering-Kreislauf

### 2.1 Realistische Modelle

Basis jedes systematischen Algorithmenentwurfs sind mathematische Modelle, die das zu lösende Problem und die ausführende Maschine beschreiben. Entscheidend ist hier die richtige Balance zwischen Einfachheit und Genauigkeit. Besonders deutlich wird das bei den Maschinenmodellen. Während die Algorithmentheorie weitgehend mit dem von Neumann-Modell arbeitet, bei dem ein einfach aufgebauter Prozessor wahlfreien Zugriff auf einen homogen strukturierten Speicher hat, bestehen moderne Computer aus vielen Prozessorkernen und Speichermodulen (Caches, Hauptspeicher, Platten,...), die ein komplexes hierarchisches Netzwerk bilden. Selbst die einzelnen Kerne haben durch verschiedene Formen von Befehlsparallelismus (Pipelining, Superskalarität, Vektorbefehle, spekulative Ausführung,...) ein sehr komplexes Verhalten. Eine genaue Modellierung der Ausführungszeit eines Programms ist deshalb kaum möglich und wäre auch nicht zielführend, weil das Ergebnis weder verständlich noch auf andere Architekturen übertragbar wäre. Das Algorithm Engineering setzt deshalb auf Abstraktionen, die einzelne besonders wichtige Aspekte des Maschinenmodells erfassen. Erfolgreich ist zum Beispiel das I/O-Modell, das sich auf zwei Speicherhierarchieebenen beschränkt – einen schnellen Speicher mit beschränkter Kapazität  $M$  und einen großen Speicher auf den in Blöcken der Größe  $B$  zugegriffen wird [6]. Ein nützliches Modell für Parallelverarbeitung betrachtet  $p$  gleichartige Rechnerknoten, die über ein Netzwerk verbunden sind. Knoten interagieren durch Nachrichtenaustausch. Jeder Prozessor kann zu einem Zeitpunkt nur eine Nachricht senden und/oder empfangen und der Zeitbedarf für eine Nachricht der Länge  $n$  ist  $\alpha n + \beta$  für geeignete Parameter  $\alpha$  und  $\beta$ . Dieses Modell passt sehr gut zu Cluster-Computern und führt auch bei Many-Core-Rechnern mit gemeinsamem (Non-Uniform Memory Access) Speicher zu effizienten Algorithmen.

Das Abstraktionsniveau für Anwendungsprobleme ist naturgemäß sehr problemabhängig. Bei einem relationalen Datenbanksystem können wir zum Beispiel sehr genau spezifizieren was das Ergebnis einer SQL-Anfrage sein soll. Bei einem Routenplaner für den Straßenverkehr ist das richtige Modell da-

gegen viel weniger klar. Eine einfache Abstraktion ist die Modellierung des Straßennetzes durch einen Graphen, bei dem Kanten Straßenabschnitte darstellen und Kantengewichte die Fahrzeit oder ein anderes Kostenmaß angeben [3]. Die genaue Modellierung fortgeschrittener Aspekte wie tageszeitabhängige Verkehrsdichte, Stauumfahrungen oder Ampelwartezeiten ist dagegen extrem schwierig und scheitert schon an der Verfügbarkeit geeigneter Daten. Trotzdem können wir hoffen durch geeignete Modelle verbesserte Ergebnisse zu erzielen. Zum Beispiel lassen sich schnelle Routenplanungstechniken für das einfache Modell auf ein zeitabhängiges Modell mit stückweise linearen Fahrzeitfunktionen übertragen oder zu auf komplexere Metriken als die Fahrzeit anpassen.

## 2.2 Entwurf

Beim Algorithmenentwurf sucht die Algorithmentheorie meistens nach asymptotisch effizienten Algorithmen für den schlechtesten Fall. Im Algorithm Engineering interessieren wir uns zusätzlich für Implementierbarkeit. Die Algorithmen sollten also einfach sein und wenn möglich bereits verfügbare Komponenten wiederverwenden. Bei der Effizienz und Lösungsqualität interessiert uns nicht nur der schlechtesten Fall, sondern mehr noch das Verhalten für “typische” Eingaben, das deutlich anders sein kann. Natürlich sind konstante Faktoren in der Ausführungszeit keineswegs zu vernachlässigen. Zum Beispiel ist unser Algorithmus zur Bestimmung minimaler Spannbäume im I/O-Modell theoretisch um einen logarithmischen Faktor schlechter als das beste bekannte Verfahren. Für realistische Hardware wird aus diesem Faktor aber eine Konstante, und verglichen zu den theoretisch besten Verfahren ergibt sich eine mindestens dreimal schnellere und sehr einfach zu implementierende Methode [4].

## 2.3 Analyse

Die Algorithmenanalyse bleibt im Algorithm Engineering wichtig, weil sie in kompakter Form Voraussagen über das Verhalten von Algorithmen in einer Vielzahl von Situationen erlaubt. Hier entstehen auch neue theoretische Herausforderungen, etwa bei der Entwicklung von Analysetechniken jenseits der worst-case Analyse. Smoothed Analysis [7] oder die Betrachtung parametrisierter Komplexität [5] liefern neue Leistungsgarantien von Algorithmen, die schon lange im praktischen Gebrauch sind, sich einer einfachen Analyse aber entziehen.

## 2.4 Implementierung

Die effiziente Implementierung von Algorithmen ist eine große Herausforderung, weil es große semantische Lücken zwischen der abstrakten Beschreibung

eines Algorithmus, höheren Programmiersprachen und der ausführenden Hardware gibt. Diese Lücken werden durch die immer komplexere Hardware eher größer als kleiner.

## 2.5 Experimente

Aussagekräftige Experimente sind der Schlüssel zum Schließen des Algorithm Engineering-Entwicklungszyklus und erfordern eine sorgfältige Planung, oft hohen Rechenaufwand und eine noch sorgfältigere Interpretation der Ergebnisse. Die experimentelle Methodik kann von Poppers wissenschaftlicher Methode lernen: Experimente sind getrieben von falsifizierbaren Hypothesen über das Verhalten der untersuchten Algorithmen. Die Hypothesen stammen aus Entwurf, Analyse, Implementierung oder aus vorhergehenden Experimenten. Die Ergebnisse bestätigen, widerlegen oder verfeinern die Hypothesen. Dies liefert dann Ideen für den Entwurf verbesserter Algorithmen, eine genauere Analyse oder effizientere Implementierung.

Wichtig ist die Reproduzierbarkeit von Experimenten. Das bedeutet, dass die involvierten Programme, Eingabedaten und sonstige Ablaufparameter sorgfältig dokumentiert werden. Diese Informationen müssen nach allgemein anerkannten Richtlinien mindestens 10 Jahre aufgehoben werden. Wissenschaftler, die Experimente überprüfen oder unter veränderten Bedingungen wiederholen möchten, sollten Zugang zu diesen Informationen bekommen. Eine exakte Wiederholbarkeit von Experimenten nach mehreren Jahren käme allerdings dem exakten Wiederaufbau einer Versuchsanlage in den Natur- oder Lebenswissenschaften gleich, und ist oft nicht möglich, da die benutzten Rechner nicht mehr existieren und alte Softwareversionen auf neuer Hardware vielfach nicht mehr laufen. Deshalb ist es ratsam nicht nur Laufzeiten zu messen sondern auch weniger hardwareabhängige Größen. Zum Beispiel kann man bei einem Algorithmus im I/O-Modell das I/O-Volumen dokumentieren.

## 2.6 Instanzen und Benchmarks

Sammlungen realistischer Eingaben für ein algorithmisches Problem sind entscheidend für Fortschritte beim Algorithm Engineering, weil sie einen einfachen und objektiven Vergleich verschiedener Ansätze erlauben. Zum Beispiel hat das Verfügbarwerden kontinentgroßer Straßengraphen um 2005 zu einer regelrechten Explosion der Leistung von Routenplanungsalgorithmen geführt [3]. Die Vereinbarung von Benchmark-Instanzen ist ein wichtiger Forschungsaspekt, zu dem die *DIMACS Implementation Challenges*<sup>1</sup> einen bedeutenden Beitrag leisten. So sind für die Graphpartitionierung und -clustering bei der *10th DIMACS Implementation Challenge - Graph Partitioning and Graph Clustering*<sup>2</sup> Standards für die experimentelle Evaluation gesetzt worden [1].

<sup>1</sup> <http://dimacs.rutgers.edu/Challenges/>

<sup>2</sup> <http://www.cc.gatech.edu/dimacs10/>

Obwohl synthetische Instanzen weniger realistisch sind, haben auch diese Vorteile. Da sie sich in beliebiger Größe erzeugen lassen, erleichtern sie Skalierbarkeitsexperimente und können als Stresstests dienen. Zudem bieten synthetische Instanzen bessere Handhabung im größeren Umfang und unter genauerer Kontrolle relevanter Parameter Experimente durchzuführen.

## 2.7 Algorithmenbibliotheken

Algorithmenbibliotheken sind die Schnittstelle zwischen Algorithm Engineering und Software Engineering. Die Bibliotheken sollten effizient, leicht benutzbar, portabel und gut dokumentiert sein. Algorithmenbibliotheken erleichtern den Transfer von Know-how aus dem Algorithm Engineering in Anwendungen. Innerhalb des Algorithm Engineering vereinfachen sie den Vergleich zwischen Algorithmen und die Konstruktion von darauf aufbauender Funktionalität (siehe auch das Beispiel in Abschnitt 2.2). Softwareengineering für Algorithmenbibliotheken ist eine besondere Herausforderung, weil es oft einen natürlichen Konflikt zwischen einfacher Benutzbarkeit, allgemeiner Einsetzbarkeit einerseits und Effizienz der Implementierungen andererseits gibt. Dazu kommt, dass zur Erstellungszeit der Bibliothek gar nicht alle Anwendungen bekannt sind. Besonders hoch sind auch die Anforderungen an die Zuverlässigkeit, weil ein Benutzer einer Algorithmenbibliothek schnell entnervt aufgibt wenn eine Bibliothek – wirklich oder scheinbar – nicht funktioniert (bekanntlich schreiben viele Programmierer/Arbeitsgruppen ihre Codes lieber selbst).

All diese Schwierigkeiten machen deutlich, dass eine gute Algorithmenbibliothek viel schwieriger zu erstellen ist als die prototypischen Implementierungen, die sonst den Algorithm Engineering-Zyklus vorantreiben. Nur ein Bruchteil der implementierten Verfahren wird also seinen Weg in Bibliotheken finden.

## 3 Zusammenfassung und Ausblick

Die Forschung im Algorithm Engineering hat in einigen Bereichen bereits erfolgreich dazu beigetragen die Kluft zwischen Theorie und Praxis zu überbrücken. Die rasanten Fortschritte, die durch neue Algorithmen für die Routenplanung oder die Suche in Texten erzielt wurden, sind Paradebeispiele für das Algorithm Engineering. Besonders faszinierend sind Forschungsergebnisse, die auf dem vollständigen Kreislauf des Algorithm Engineering basieren und aus Experimenten und Anwendungen gewonnene theoretische Erkenntnisse wieder in den Entwurf neuer Algorithmen einbeziehen [2].

Mit der Weiterentwicklung der Hardware oder der zunehmenden Verfügbarkeit riesiger Datensätze und der wachsenden Bedeutung von Anwendungen, die auf riesigen Datensätzen basieren, stellen sich immer wieder neue Herausforderungen an die Forschung im Algorithm Engineering. Aber auch in der Lehre an Universitäten sollte heutzutage Algorithm Engineering in der Ausbildung von Informatikern einen festen Platz haben.

## Literatur

1. D. A. Bader, H. Meyerhenke, P. Sanders, C. Schulz, A. Schumm, and D. Wagner. Benchmarks for Network Analysis. In: J. Rokne, R. Alhajj (eds.) *Encyclopedia of Social Network Analysis and Mining*, 2012. Submitted.
2. D. Delling, A. V. Goldberg and R. F. Werneck. Shortest Paths in Road Networks: From Practice to Theory and Back. In *it—Information Technology*, volume 53 on Algorithm Engineering, P. Sanders and D. Wagner (eds.), pages 294–301, Oldenburg-Verlag, 2011.
3. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, volume 5515 of *LNCS State-of-the-Art Survey*, pages 117–139. Springer, 2009.
4. R. Dementiev, P. Sanders, D. Schultes, and J. Sibeyn. Engineering an external memory minimum spanning tree algorithm. In *IFIP TCS*, pages 195–208, Toulouse, 2004.
5. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
6. U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS Tutorial*. Springer, 2003.
7. D. A. Spielman and S. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 296–305, 2001.