Heureka: Kapitel Informatik

Ulrike von Luxburg

18. November 2009

1 Einleitung

Informatik ist eine weit gefächerte Disziplin, unter deren Dach sich viele verschiedene Teilgebiete versammeln. Die Methoden und Herangehensweisen dieser Teilgebiete unterscheiden sich sehr stark voneinander. Während die Methoden der theoretischen Informatik ähnlich zu denen der Mathematik sind, ist das Design von Hardware-Komponenten eher mit der Physik oder den Ingenieurwissenschaften verwandt. Das Herzstück der Informatik, mit dessen Hilfe sich die Informatik als eigenständige wissenschaftliche Disziplin etabliert hat, ist die praktische Informatik. Sie untersucht alle Aspekte der Frage, wie sich gegebene Probleme mit Hilfe eines Computer lösen lassen: von der Entwicklung von Betriebssystemen und Programmiersprachen über das Design von Algorithmen und Datenstrukturen bis zur Frage, wie Software gut geplant, implementiert und gewartet werden kann.

Heureka-Erlebnisse in der Informatik können sehr unterschiedlich ausfallen. Es verschafft enorme Befriedigung, ein Softwaresystem so implementiert zu haben, dass stabil ist und genau das tut, was es soll. Dafür ist aber oft nicht ein einziger Gedankenblitz die Ursache, sondern handwerkliches Geschick, genügend Ausdauer und eine glückliche Kombination vieler kleiner Ideen, die alle für sich genommen kein Heureka darstellen. Unter Heureka-Erlebnissen verstehen wir eher einzelne Ideen, Gedankenblitze, die alles Hergebrachte auf den Kopf stellen. In der Forschung treten solche Heureka-Erlebnisse immer wieder auf: eine Idee, wie man ein Verfahren so verbessern kann, dass es viel schneller wird. Oder eine elegante Lösung für ein neues Problem, das so noch nicht betrachtet wurde. Oder aber der Beweis, dass eine bestimmtes Problem keine effiziente Lösung besitzen kann und und mit fundamental neuen Methoden gelöst werden muss. Im Folgenden schauen wir uns an, wie Heureka-Erlebnisse in der praktischen Informatik zustande kommen. Exemplarisch wollen wir uns dabei auf das Gebiet des Algorithmendesigns konzentrieren.

Ein Algorithmus ist eine Rechenvorschrift, mit deren Hilfe man einem Computer Schritt für Schritt mitteilen kann, wie er ein bestimmtes Problem lösen soll. Betrachten wir zum Beispiel das folgende klassische Problem, genannt das Problem des Handlungsreisenden.

Problem des Handlungsreisenden: Gegeben ist eine Liste von verschiedenen Städten. Gesucht ist eine möglichst kurze Reiseroute, in der jede Stadt genau ein mal besucht wird und man dann zum Ausgangspunkt zurückkehrt.

Um dieses Problem mit einem Computer lösen zu können, müssen wir uns eine „Lösungsstrategie“ ausdenken: einen Algorithmus. Hier sind zwei Beispiele:

Erster Algorithmus zur Lösung des Handlungsreisenden-Problems:

1. Starte in der ersten Stadt auf der Liste.
2. Solange wir noch nicht alle Städte auf der Liste besucht haben, führen wir die folgenden Schritte aus:
   (a) Unter allen noch nicht besuchten Städten bestimmen wir diejenige, die am nächsten zu unserem jetzigen Standpunkt ist.
   (b) Dann fahren wir in diese Stadt.
3. Am Schluss kehren wir zu unserem Ausgangspunkt zurück.

Zweiter Algorithmus zur Lösung des Handlungsreisenden-Problems:

1. Erstelle eine Liste aller möglichen Reiserouten, in denen die Städte abgefahren werden können.
2. Für jede Reiseroute in dieser Liste: berechne die Länge der Reiseroute.
3. Wähle diejenige Reiseroute, die am kürzesten ist.

Es gibt viele verschiedene Kriterien, mit denen man einen Algorithmus bewerten kann. Zum Beispiel sind beide obige Algorithmen „korrekt“ in dem Sinne, dass sie für eine beliebige Liste von Städten eine Reihenfolge bestimmen, in der die Städte besucht werden sollen. Sie produzieren also zu jeder Eingabe auch eine sinnvolle Ausgabe.

Sehr wichtig in der Praxis ist, wie „schnell“ ein Algorithmus seine Lösung berechnet. Die beiden Algorithmen von oben sind sehr unterschiedlich schnell: der erste Algorithmus braucht nicht mehr als n^2 Rechenschritte, um eine Lösung zu berechnen.[1] Der zweite Algorithmus benötigt jedoch ungefähr n^n Rechenschritte[2] — das ist viel, viel mehr! Während ein moderner Computer mit dem ersten Algorithmus auch für große Städtelisten nur Sekunden braucht, um seine Lösung zu berechnen, bräuchte der zweite Algorithmus schon für 50 Städte mehr Rechenzeit, als das Universum alt ist! Daran kann man sehen, dass es extrem wichtig ist, wie schnell ein bestimmter Algorithmus zu seiner Lösung kommt.

Neben der Schnelligkeit ist die Güte der berechnenten Lösung das Ausschlag gebende Kriterium zur Bewertung von Algorithmen. Im Beispiel von oben: da der zweite Algorithmus alle möglichen Reiserouten durchprobiert, findet er mit Sicherheit immer die kürzeste aller möglichen Reiserouten. Für den ersten Algorithmus ist das jedoch nicht der Fall, im Gegenteil. Es gibt Beispiele von Städtelisten, für die dieser Algorithmus eine extrem schlechte Reiseroute berechnet: um Beispiel eine Route, die 100 mal länger ist als die optimale Route.[3] In diesem Kriterium schneidet der zweite Algorithmus also deutlich besser ab als der erste.

---

[1] Für die besonders interessierten Leser: Wir starten in der ersten Stadt. Nun müssen wir berechnen, welche der $n-1$ verbliebenen Städte am nächsten zu unsere Stadt ist. Dazu müssen wir die $n-1$ Entfernungen betrachten und das Minimum bestimmen. Das benötigt $n-1$ Rechenschritte. Nun gehen wir in die nächste Stadt und müssen die Entfernungen zu den noch verbliebenen $n-2$ Städten betrachten, was $n-2$ Rechenschritte bedeutet. Und so weiter. Wenn wir fertig sind, haben wir also $(n-1)+(n-2)+(n-3)+\ldots = (n-1)n/2 \approx n^2$ Rechenschritte benötigt.

[2] Der intuitive Grund dafür: Es gibt circa $n! \approx n^n$ mögliche Reiserouten. Da wir für alle diese Routen die Länge berechnen müssen, brauchen wir also mindestens n^n Rechenschritte.

[3] Das Problem des ersten Algorithmus ist, dass er die Länge der Strecke von der letzten Stadt zurück zum Ausgangspunkt nicht in Betracht zieht. Es kann also passieren, dass man sich immer weiter vom Ausgangspunkt entfernt, so dass man am Schluss die weiteste aller möglichen Entfernungen zurück legen muss, um zum Ausgangspunkt zurückzukehren.

2 Werden in der Informatik Thesen aufgestellt?

Ergebnisse in der Informatik werden hauptsächlich in Form von Artikeln in Konferenzbänden oder Fachzeitschriften publik gemacht. Eine These wird in solchen Artikeln meist nicht explizit aufgestellt. Implizit findet sich in vielen Artikeln eine These, die aber eher als „Hauptaussage“ oder „Hauptergebnis“ umschrieben wird. Der Artikel dient dann dazu, diese Hauptaussage zu untermauern. Die Hauptaussage eines Artikels kann sehr unterschiedlich ausfallen:

- Manchmal löst ein Artikel eine offene Frage, zum Beispiel: „Kann es einen effizienten Algorithmus geben, der für das Problem des Handlungsreisenden immer die optimale Lösung liefert?“ Falls es eine verbreitete Vermutung gibt, wie die Antwort aussehen könnte, dient diese als These. In anderen Fällen wird einfach die offene Frage klar zu formuliert und dann die gefundene Antwort begründet.

- Sehr viele Artikel beschreiben neue Algorithmen, die in irgendeinem Sinne „besser“ sind als schon bekannte Algorithmen. Thesen werden in diesem Fall zwar meist nicht explizit formuliert. Es gibt aber eine Reihe von Standardthesen, die in solchen Artikeln normalerweise gezeigt werden. Auf diese werden wir weiter unten genauer eingehen.

  Es ist zu beachten, dass üblicherweise nur positiv formulierte Thesen untersucht und dann bestätigt werden („Mein Algorithmus ist schneller“). Manchmal ist das Ergebnis zwar etwas vielschichtiger („Mein Algorithmus ist zwar langsamer, aber robuster“). Es kommt jedoch nur selten vor, dass die Hauptaussage eines Artikels über einen neuen Algorithmus negativ ist. Das liegt daran, dass es schwierig ist, objektiv zu beschreiben, warum ein spezieller Ansatz nicht funktioniert. Vor allem muss ausgeschlossen werden können, dass ein Verfahren nur deshalb nicht funktioniert, weil man es ungeschickt angewendet hat. Negative Ergebnisse sind nur dann von Interesse, wenn daraus eine Lehre für die Zukunft gezogen werden kann.

- Manche Artikel analysieren schon bestehende Algorithmen und versuchen, deren Verhalten besser zu verstehen und vorauszusagen. Man könnte zum Beispiel charakterisieren, welche Eigenschaften eine Städteliste haben muss, damit der oben beschriebene Algorithmus zur Lösung des Handlungsreisenden-Problems eine optimale Lösung erzeugt. In solchen Artikeln wird meist eine explizite These formuliert (auch wenn sie nicht so genannt wird). Das ist auch nötig, denn für diesen Fall gibt es keine wohletablierten „Standardthesen“, die implizit zugrunde gelegt werden können.

- Wenige, meist aber einflussreiche Artikel beschreiben ein neuartiges Problem. Sie erläutern, warum dieses Problem interessant ist und was für Vorteile es hätte, wenn man dieses Problem lösen könnte. In solchen Artikeln ist keine These vorgesehen.

Wie oben schon erwähnt gibt es einen Satz von Standardthesen, die in vielen Artikeln untersucht werden. Das geschieht dann in mehr oder weniger expliziter Form. Solche Standardthesen sind zum Beispiel:

- *Im Vergleich zu schon bestehenden Algorithmen ist der neue Algorithmus schneller.* Zum Beispiel braucht er für die gleiche Städteliste weniger Rechenschritte.

- *Im Vergleich zu schon bestehenden Algorithmen erzielt der neue Algorithmus qualitativ bessere Ergebnisse.* Zum Beispiel liefert er für die gleiche Städteliste kürzere Reiserouten.

- *Im Vergleich zu schon bestehenden Algorithmen ist der neue Algorithmus einfacher zu implementieren.* Bevor ein Algorithmus von einem Computer ausgeführt werden kann, muss er in einer Programmiersprache implementiert werden. Das führt zu einem Computerprogramm, das von einem Computer interpretiert und ausgeführt werden kann. Manche Algorithmen sind sehr einfach zu implementieren. Andere Algorithmen sind jedoch höchst komplex, und es ist nicht offensichtlich, wie sie am besten in eine Programmiersprache übersetzt werden. Oft unterscheiden sich dann die Ergebnisse verschiedener Implementierungen des gleichen Algorithmus voneinander. In diesem Falle ist es schwierig, den Algorithmus zu evaluieren, was ein großer Nachteil ist. Es ist also wünschenswert, dass ein Verfahren einfach zu implementieren ist. Auf diesen Punkt werden wir auch im dritten Kapitel noch zu sprechen kommen.

- *Im Vergleich zu schon bestehenden Algorithmen ist der neue Algorithmus einfacher zu bedienen.* Manche Algorithmen haben viele Stellschrauben (Parameter), mit denen man das Verhalten des Algorithmus verändern kann und die man richtig einstellen muss. Je mehr Stellschrauben ein Algorithmus hat, desto schwieriger ist es zu wissen, welche Einstellungen zu guten Ergebnissen führen.

- *Im Vergleich zu schon bestehenden Algorithmen ist der neue Algorithmus zuverlässiger und robuster.* Manchmal treffen Algorithmen zufällige Entscheidungen (zum Beispiel, sie wählen den Standort der Städtereise zufällig aus der Liste aus). Oder ihre Ergebnisse hängen von an und für sich unwichtigen Details der Eingabedaten ab (zum Beispiel von der Reihenfolge, in der die Städteliste eingegeben wird). Je weniger das Ergebnis eines Algorithmus unter solchen Variationen der Eingabe schwankt, desto besser.

Es ist etablierte Praxis, dass für einen neuen Algorithmus mindestens eine oder mehrere dieser oder verwandter Thesen untersucht werden.

3 Wie wird in der Informatik eine These oder Behauptung gestützt?

Will man in der Informatik jemanden davon überzeugen, dass man einen guten neuen Algorithmus gefunden hat, muss man sowohl theoretische Analysen als auch praktische Evaluierungen vorweisen können.

Mit Hilfe *theoretischer Ergebnisse* werden grundlegende Eigenschaften eines Algorithmus bewiesen. Theoretische Aussagen werden wie in der Mathematik in Form von Sätzen formuliert und mit Hilfe von Beweisen begründet (siehe Kapitel Mathematik). Beispiele für theoretische Aussagen sind:

> Für eine beliebige Liste von n Städten, deren Abstände mindestens einen Kilometer voneinander betragen, findet ein bestimmter Algorithmus immer eine Reiseroute, deren Länge nicht mehr als $n/2$ Kilometer von der optimalen Route abweicht.

Theoretische Aussagen sind eine wichtige Grundlage, auf der Algorithmen bewertet werden. Bestimmte theoretische Aussagen sind für jeden neuen Algorithmus von Interesse und werden standardmäßig hergeleitet. Ein Beispiel ist die Anzahl der Rechenschritte, die ein Algorithmus braucht, um bei einer bestimmten Eingabe seine Ausgabe zu berechnen.

> Der erste Algorithmus braucht nie mehr als n^2 Rechenschritte, um für eine Eingabe von n Städten eine Reiseroute zu berechnen.

Theoretische Aussagen über Algorithmen können unterschiedlicher Natur sein. Es gibt theoretische Aussagen, die keiner weiteren experimentellen Überprüfung bedürfen und komplett für sich alleine stehen:

> Für alle möglichen Listen von Städten berechnet der zweite Algorithmus immer die allerkürzeste Reiseroute.

Wenn diese theoretische Aussage korrekt bewiesen ist, bedarf es keiner weiteren Evidenz, um ihre Richtigkeit zu etablieren. Insbesondere gewinnt man keinen Mehrwert an Erkenntnis, wenn man diese Aussage nun mit Hilfe von Simulationen in der Praxis verifiziert (außer man glaubt den Beweis nicht). Oft gelangt man durch theoretische Überlegungen jedoch zu Aussagen, die man nicht eins zu eins in die Praxis übertragen kann. Das heißt nicht, dass die theoretischen Ergebnisse falsch sind (sind sie nicht, sie wurden ja formal bewiesen). Aber da theoretische Aussagen manchmal sehr schwer zu beweisen sind, werden für die Theorie vereinfachende Annahmen gemacht, die in der Praxis nicht unbedingt gelten; oder man beweist ein bischen „am Ziel vorbei“. Ein Beispiel hierfür sind „worst case Laufzeiten“. Hier wird analysiert, wie viele Rechenschritte ein Algorithmus höchstens braucht, um seine Ausgabe zu berechnen. Eine Aussage zu worst case Laufzeiten haben wir schon oben gesehen: „Der erste Algorithmus braucht nie mehr als n^2 Rechenschritte, um für eine Eingabe von n Städten eine

Reiseroute zu berechnen.“ Selbst wenn diese Aussage korrekt bewiesen wurde, sagt sie nichts darüber aus, wie sich der Algorithmus nun in der Praxis verhält. Braucht er tatsächlich meistens n^2 Schritte, oder braucht er im Durchschnitt deutlich weniger Schritte?

Solch eine Frage wird zunächst mit Hilfe von *Simulationen* analysiert. Simulationen sind Experimente, mit deren Hilfe man das Verhalten eines Algorithmus unter kontrollierten Bedingungen austestet. Zunächst werden künstliche Eingabedaten generiert, für die die gewünschte Ausgabe des Algorithmus wohlbekannt ist. Im Beispiel des Handlungsreisenden-Problems stellen wir uns zum Beispiel n Städte vor, die regelmäßig auf einem Kreis angeordnet sind. Für diese Eingabe ist die optimale Reiseroute bekannt: wir müssen die Städte einfach entlang des Kreises besuchen. Nun können wir ausprobieren, ob unser Algorithmus für diese Eingabe die optimale Lösung findet oder wie weit die Ausgabe unseres Algorithmus von der optimalen Lösung abweicht. In Simulationen werden dann bestimmte „Parameter“ der Eingabedaten kontrolliert verändert. Anschließend beobachtet man, wie der Algorithmus darauf reagiert. Wir können zum Beispiel die Städte des Kreises dem Algorithmus in verschiedenen Reihenfolgen eingeben. Oder wir variieren den Abstand zwischen zweien dieser Städte ein bischen und probieren aus, wie sich der Algorithmus dann verhält. Oder wir wenden den Algorithmus für verschiedene Anzahl von Städten an.

Idealerweise helfen solche ersten Simulationen, das Verhalten des Algorithmus auf sehr einfachen Beispielen zu verstehen. Der nächste Schritt in Simulationen ist normalerweise, den Algorithmus auf künstlichen Daten auszuprobieren, die in gewissem Sinne „Prototypen“ für realistische Anwendungen sind. Wir könnten uns zum Beispiel vorstellen, zufällig n Städte auf einer Landkarte zu wählen und den Algorithmus für diese Liste laufen zu lassen. Im Beispiel des Handlungsreisenden-Problems könnten wir zum Beispiel beobachten, dass viele der Lösungen des ersten Algorithmus eine besonders lange Rückreise von der letzten in die erste Stadt aufweisen. Solche Simulationen haben der Vorteil, dass sie einerseits „realistisch“ in dem Sinne sind, dass sie auf „naturnahen“ Daten agieren. Andererseits hat man aber beliebig viele Datensätze zur Verfügung und kann die Datensätze per Hand manipulieren, wenn man ungewöhnliches Verhalten beobachtet.

Hat ein Algorithmus auch in diesen Simulationen das erwünschte Verhalten gezeigt, geht man dann zum Härtetest über: man evaluiert den Algorithmus auf „echten Anwendungsbeispielen“. Dieser Schritt wird auf Denglisch *„real world experiments“* genannt. Oft ergeben sich in diesen Praxistests allerhand Überraschungen. Eingabedaten aus Anwendungsbeispielen sind oft nicht so „regelmäßig“ wie künstlich erzeugte Simulationsdaten. Im Fall des Handlungsreisenden könnte es zum Beispiel sein, dass ein Geschäftsreisender mehrere Städte in Süddeutschland und mehrere Hauptstädte anderer Europäischer Länder besuchen muss. Die „Charakteristik“ dieser Liste von Städten ist nun eine andere als die der simulierten Städtelisten, in denen alle Städte recht gleichmässig verteilt

waren. Es kann sein, dass ein Algorithmus auf den simulierten Daten hervorragend abschneidet, aber auf „echten“ Daten eher schlechte Ergebnisse liefert.

Sowohl Simulationen als auch Praxistests werden oft als Experimente bezeichnet. Ein wichtiger Knackpunkt in Experimenten ist die Auswahl der Datensätze, auf denen verschiedene Algorithmen verglichen werden. Oft funktionieren Algorithmen auf manchen Datensätzen sehr gut, auf anderen dagegen nur mittelmäßig. Für einige Probleme kann man sogar theoretisch beweisen, dass es keinen Algorithmus geben kann, der auf allen möglichen Eingabedaten „besser“ ist als jeder andere Algorithmus („no free lunch theorem“). Für die Evaluation von Algorithmen stellt das natürlich ein Problem dar. Die Versuchung ist groß, in einem Artikel nur Ergebnisse auf solchen Datensätzen zu präsentieren, auf denen der eigene Algorithmus gut abschneidet. Um diesem Vorwurf zu begegnen, kann man verschiedene Schritte unternehmen. Es macht Sinn, möglichst viele verschiedene Datensätze zu benutzen, und die Auswahl der Datensätze auch zu begründen. Für manche Problem hat sich auch eine Art „Standard“ herausgebildet, auf welchen Datensätzen ein Algorithmus immer getestet werden sollte. Wenn ein Algorithmus auf bestimmten Beispielen nicht gut funktioniert, muss das nicht notwendigerweise schlecht sein. Ein wichtiger Teil der Analyse eines Algorithmus ist herauszufinden, auf welchen Datensätzen ein Algorithmus gut funktioniert und auf welchen nicht. Idealerweise sollte man daraus eine Empfehlung ableiten können: „Mein Algorithmus funktioniert besonders gut, wenn alle Städte ungefähr den gleichen Abstand voneinander haben. Wenn das nicht der Fall ist, sollte man lieber den anderen Algorithmus verwenden.“

Wie werden nun die Ergebnisse von Simulationen und Praxistests evaluiert und kommuniziert? Hier unterscheiden sich verschiedene Publikationen deutlich voneinander, und hier trennt sich auch oft die Spreu vom Weizen. In guten Artikeln wird folgendermaßen vorgegangen: es werden eine oder wenige Fragen formuliert, die mit Hilfe der Simulationen und Praxistests geklärt werden sollen. Zum Beispiel, ob der neue Algorithmus schneller ist als der alte, und ob er tendenziell kürzere Reiserouten berechnet. Dann wird der Aufbau der Experimente beschrieben. Insbesondere sollte hierbei erklärt werden, warum die Simulationen geeignet sind, Einsicht in die gestellte Frage zu geben. Danach werden die Ergebnisse der Experimente ausgewertet, und zwar in der Form von einigen aussagekräftigen Tabellen und Graphiken. So sieht die gute Praxis aus. Leider ist es aber nicht unüblich, dass Fragen nicht genau formuliert werden, eine große Anzahl von Experimenten durchgeführt wird, und eine noch größere Anzahl an Graphiken und Tabellen präsentiert wird. Dann wird es dem Leser überlassen, sich durch die Daten zu wühlen und sich eine Meinung zu bilden (die dann meist negativ, da genervt, ausfällt). Fast alle Informatiker sind sich einig, dass die letztere Vorgehensweise wenig zielführend ist. Dennoch wird sie immer wieder angetroffen.

Bisher haben wir gesehen, dass viele Ergebnisse in der praktischen Informatik durch drei Arten der Evidenz etabliert werden: (a) theoretische Aussagen

und deren Beweise, (b) künstliche Simulationen und (c) Praxistests. Es gibt jedoch noch einige weitere Kriterien, die die Aussagen eines Artikels stärken können. Das erste Kriterium hat mit der Nachprüfbarkeit der Ergebnisse zu tun. Ziel von Wissenschaft ist es, Aussagen zu treffen, deren „Wahrheitsgehalt“ von anderen nachgeprüft oder getestet werden kann. Im speziellen Falle der Informatik heißt das oft, dass Algorithmen von anderen Leuten benutzt werden können, die das gleiche Problem lösen wollen. Um Algorithmen zu benutzen, muss man sie allerdings in einer Programmiersprache implementieren. Dieser Schritt ist normalerweise nicht der eigentliche Gegenstand der Forschung, kann aber extrem zeitaufwändig und nervenaufreibend sein. Insbesondere müssen bei einer Implementierungen viele kleine Details bedacht werden, die in wissenschaftlichen Artikeln eigentlich nie beschrieben werden. Es wird implizit vorausgesetzt, dass der Leser in der Lage ist, diese fehlenden Details selbst auszuarbeiten. Oft unterscheiden sich die Ergebnisse verschiedener Implementierungen des gleichen Algorithmus voneinander. Dann ist es natürlich schwierig, einen in einem Artikel vorgeschlagen Algorithmus zu evaluieren. Man scheut den Aufwand, Tage-oder Wochenlang ein Verfahren zu implementieren, von dessen Nutzen man noch nicht überzeugt ist. Ein großer Pluspunkt ist es deshalb, wenn Wissenschaftler den Quellcode ihrer Implementierung öffentlich zugänglich machen. Dann können andere Forscher den gleichen Code benutzen, um die Ergebnisse des Artikels nachzuprüfen und ihre eigenen Experimente damit durchzuführen. Das stärkt die Glaubwürdigkeit eines Artikels enorm. Einen noch größeren Pluspunkt erhält, wessen Implementierung auch noch besonders sauber und nachvollziehbar vorgenommen wurde.

Das letzte Kriterium, das zur Bewertung von Forschungsergebnissen benutzt wird, ist ein sehr subjektives Kriterium. Auch in der Informatik gibt es eine Vorliebe für besonders „elegante“ und „einfache“ Lösungen. Es ist nicht leicht, für Außenstehende zu beschreiben, wann ein Algorithmus elegant ist. Oft enthalten elegante Algorithmen ein gewisses Überraschungsmoment. Anstatt auf eine offensichtliche, aber umständliche Lösung zurückzugreifen, wird ein gänzlich neuer Ansatz präsentiert. Dieser Ansatz ist im Nachhinein (mit ein bischen Nachdenken) leicht nachzuvollziehen; er stellt aber eine neue und ungewöhnliche Lösung dar, auf die man nicht einfach so kommt. Es steckt ein Geistesblitz dahinter.

Es gibt viele Forschungsarbeiten, von deren Korrektheit oder Nützlichkeit man sich als Informatiker gerne überzeugen lässt, indem die oben beschriebene Evidenzkette aufgebaut wird. Ein befriedigendes Heureka-Erlebnis stellt sich aber meistens dann ein, wenn die vorgeschlagene Lösung besonders „elegant“ ist. Selbst eine elegante Lösung für ein Problem zu finden, ist sicher die schönste Form von Heureka!

4 Wie unterscheiden sich Kommunikations- und Erzeugungsprozess?

Bisher haben wir beschrieben, welche Evidenzen man braucht, um neue Forschungsergebnisse in der Informatik zu etablieren. Im Forschungsalltag werden neue Ergebnisse oft nicht auf so systematischem Wege erreicht. Natürlich kommt es manchmal vor, dass eine Forschungsarbeit aussieht wie oben beschrieben: es gibt eine offene Frage, man hat eine Vermutung, wie eine Lösung aussehen könnte, und man verifiziert diese mit Hilfe von Simulationen, Praxistests und theoretischen Ergebnissen. In dieser Form findet Forschung aber meistens nicht statt. Einer der wichtigsten und kreativsten Schritte in der Forschung ist der Weg bis zur Formulierung einer These.

Forschungsfragen sind oft sehr viel diffuser und unkonkreter als eine These. Sie starten zum Beispiel mit einer Idee, wie man einen Algorithmus verbessern oder einen neuen entwerfen könnte. Diese Idee kann auf sehr unterschiedliche Weisen entstehen. Manche Probleme trägt man eine ganze Weile im Kopf herum, bis man auf einmal in recht unvermittelt über einen möglichen Lösungsansatz stolpert, oft in ganz anderem Kontext. Oder man ist unzufrieden mit den existierenden Lösungen und versucht durch „Herumspielen“ zu verstehen, warum sie nicht funktionieren. Manchmal gelangt man dabei zu neuen Einsichten, die in einer Verbesserungsidee münden. Sobald man eine Verbesserungsidee hat, probiert man sie auf kleinen simulierten Beispielen aus. Dabei stellt sich oft heraus, das die Idee so einfach nicht funktioniert. In einem iterativen Prozess verbessert man dann nach und nach den neuen Ansatz und analysiert seine theoretischen Eigenschaften. Nach vielem Hin und Her hat man vielleicht einen neuen Algorithmus gefunden. In Fachartikeln, in denen man den neuen Ansatz dann veröffentlicht, wird dieser iterative Prozess nicht weiter beschrieben. Man geht nur auf die Evidenz ein, wie wir es oben schon dargelegt haben.

Es ist gar nicht so einfach, gute Forschungsfragen zu stellen. Um eine gute Frage zu stellen, muss man die Literatur kennen und muss verstanden haben, warum bestehende Ansätze nicht gut sind, welche Art von Problemen damit vielleicht nicht gelöst werden können, oder wie sie grundsätzlich verbessert werden können. Außerdem ist eine gute Forschungsfrage dadurch charakterisiert, dass sie „schwierig aber nicht zu schwierig“ ist. Mit der Lösung einer einfachen Frage kann man niemanden beeindrucken. Andererseits macht es wenig Sinn, Fragen zu bearbeiten, die nahezu unlösbar sind. Der erste Schritt auf dem Weg zum selbständigen Wissenschaftler ist zu lernen, gute Fragen zu stellen. Je besser und interessanter die Frage, desto vielversprechender die zu erwartende Antwort.

Gänzlich neue Forschungsgebiete in der Informatik ergeben sich oft aus dem Bezug zu Anwendungen. Salopp gesagt: wenn man eine gute Idee für eine neue Anwendung von Computern hat, eröffnen sich Forschungsfragen und manchmal ganze Forschungsfelder. Am Anfang sind die Probleme meist nicht gut

umrissen, man versucht naive Lösungsmethoden und stellt fest, dass sie nicht funktionieren. Dann macht man sich daran, systematisch die Literatur nach Lösungsmöglichkeiten zu durchforsten und Kollegen um Rat und Ideen zu fragen. Wenn man Glück hat, kristallisiert sich im Laufe der Zeit dann eine etwas konkretere Forschungsfrage heraus, die man mit wissenschaftlichen Methoden angehen kann.

Wissenschaftliche Thesen, wie wir sie am Anfang diskutiert haben, stehen paradoxerweise fast ganz am Ende der Forschungskette. Erst wenn man sehr genau weiß, was man will, und eine konkrete Vermutung hat, wie man es erreichen könnte, arbeiten Informatiker mit so etwas wie Thesen. Wie in anderen Forschungsdisziplinen auch ist der spannendste und überraschendste Teil der Forschung jedoch der Prozess, mit dem man zu einer solchen These gelangt.

5 Exkurs: Automatische Inferenz

Zum Schluss möchten wir noch ein Teilgebiet der Informatik vorstellen, das versucht, den Prozess der Bildung von Hypothesen zu formalisieren und mit Hilfe von Computern zu automatisieren. Dieses Teilgebiet heißt „maschinelles Lernen“. Das Ziel des maschinellen Lernens ist, Algorithmen zu entwickeln, mit deren Hilfe ein Computer selbständig *lernen* kann, bestimmte Aufgaben zu erledigen. Ein Lernalgorithmus bekommt sogenannte Trainingsbeispiele als Eingabe. Das sind Beispiele von Ein-und Ausgabepaaren, für die sowohl Aufgabenstellung als auch gewünschtes Ergebnis bekannt sind. Ausgehend von diesen Beispielen soll der Computer nun Regeln ableiten, mit deren Hilfe er ähnliche Aufgaben in Zukunft selbständig lösen kann.

Ein populäres Beispiel ist die Konstruktion von Filtern, die unerwünschte Werbe-Emails, sogenannte Spam-Emails, ausfiltern. Aufgrund der schieren Flut von Werbe-Emails ist es nahezu unmöglich, dass Menschen eigenhändig Regeln aufstellen, nach denen Spam-Emails aussortiert werden können. Statt dessen verwendet man einen Lernalgorithmus. Als Trainingsbeispiele erhält der Algorithmus Beispiele von Werbe-Emails und Beispiele von normalen Emails. Der Algorithmus soll nun selbständig Regeln finden, mit deren Hilfe er zukünftige Emails als „Spam“ oder „Nicht Spam“ klassifizieren kann. Dabei soll er natürlich so wenig Fehler wie möglich machen. Für jede Menge an Trainings-Emails gibt es viele Regeln, mit denen man diese *schon bekannten* Emails gut klassifizieren kann. Die einfachste Regel ist „auswendig lernen“: man klassifiziert genau diejenigen Emails als Spam, die mit einer schon bekannten Spam-Email übereinstimmen. Diese Regel wird aber leider für *zukünftige* Emails nicht besonders erfolgreich sein. Sobald eine neue Spam-Email auftaucht, die wir bisher noch nicht kannten, wird sie nicht mehr als Spam erkannt. Unsere Regeln müssen also etwas abstrakter sein, sie müssen generalisieren können.

Von höherer Warte aus betrachtet ist die Aufgabe eines Lernalgorithmus,

basierend auf schon bestehenden Daten möglichst gute Hypothesen aufzustellen. Man spricht deshalb auch von automatischer Inferenz (oder von empirischer Inferenz, wenn man betonen will, dass diese Inferenz auf empirisch erhobenen Daten beruht). Jeder mögliche Regelsatz, den der Algorithmus von Trainingsbeispielen ableiten könnte, stellt eine mögliche Hypothese dar. Der Forschungsgegenstand des maschinellen Lernens ist die Frage, wie man aus einem großen Topf möglicher Hypothesen diejenige auswählen kann, die einerseits die bestehenden Daten gut erklärt (die die bisherigen Emails korrekt klassifizieren kann) und die andererseits das größte Potential hat, auf zukünftigen Daten gut zu passen (zukünftige Emails korrekt zu klassifizieren). Das maschinelle Lernen hat hierzu komplexe mathematische Theorien entwickelt, mit deren Hilfe verschiedene Hypothesen bewertet werden können. Man kann zum Beispiel beweisen, dass einfache Hypothesen mit hoher Wahrscheinlichkeit besser geeignet sind, zukünftige Daten zu erklären, als besonders komplexe Hypothesen. Was „einfach“ und „komplex“ heißt, wird in der Theorie des maschinellen Lernens genau definiert. Weiterhin wird im maschinellen Lernen untersucht, welche Annahmen man machen muss, damit Lernen überhaupt möglich ist. Man kann beweisen, dass es unmöglich ist, ohne Annahmen zu Hypothesen zu gelangen, die zukünftige Daten gut beschreiben können.

Das maschinelle Lernen ist in vielen Anwendungen (wie der Konstruktion von Spam-Filtern) nicht mehr wegzudenken. Wissenschaftstheoretisch besonders interessant sind jedoch die Fälle, wo andere wissenschaftliche Disziplinen sich des maschinellen Lernens bedienen, um eigene Hypothesen aufzustellen und dann zu testen. Zum Beispiel benutzt man maschinelle Lernverfahren in der Bioinformatik, um Regionen der DNA zu identifizieren, in denen mit hoher Wahrscheinlichkeit Gene kodiert sind. Oder in der Pharmakologie, wo man aus aus einer großen Anzahl chemischer Komponenten diejenigen heraus sucht, die ein besonderes Potential haben, als Wirkstoff für ein bestimmtes Medikament dienen zu können. Die Ergebnisse des maschinellen Lernens dienen dann als Ausgangspunkte (Hypothesen) für aufwändige Labor-Experimente. In diesen Experimenten wird dann mit klassischen Verfahren untersucht, ob die Region der DNA wirklich ein Gen enthält, oder ob die chemische Komponente tatsächlich die erwünschte Wirkung für das Medikament entfaltet. Das maschinelle Lernen übernimmt also eine Art Vorverarbeitungsschritt: da unmöglich für alle Teile der DNA im Labor verifiziert werden kann, ob es sich um Gene handelt oder nicht, versucht man, die vielversprechendsten Kandidaten per Computer zu ermitteln. Nur diese Kandidaten werden dann der Laboranalyse unterzogen.

Gibt es denn nun im Bereich des maschinellen Lernens auch Heureka-Erlebnisse? Das kommt darauf an. Die Hypothesen, die das maschinelle Lernen generiert, sorgen normalerweise nicht für ein Heureka-Erlebnis, ganz im Gegenteil. Die Gründe, warum der Computer eine bestimmte Region der DNA für ein Gen hält, sind meist nur schwer nachzuvollziehen. Ein solcher „black box“- Mechanismus zur Generierung von Hypothesen liefert dem Menschen zunächst keine Einsicht und ist deshalb denkbar ungeeignet für ein Heureka-Erlebnis. Ein sol-

ches stellt sich beim Anwender aber dann ein, wenn die Ergebnisse des Lernalgorithmus erfolgreich weiter verwendet werden können. Zum Beispiel, wenn der Spam-Filter gut funktioniert. Oder wenn Labor-Experimente bestätigt haben, dass die vorgeschlagene Region der DNA tatsächlich ein Gen enthält. Falls ein Lernalgorithmus besonders erfolgreich ist, gute Hypothesen für Anwendungen vorzuschlagen, sorgt das wiederum für ein Heureka beim Entwickler des Lernalgorithmus. Ein doppelt gemoppeltes Heureka, sozusagen!