

Bösartige Gegenspieler, gutartige Probleme

Zusammenfassung

Diese Arbeit möchte einerseits erläutern, wie *bösartige Gegenspieler* in der theoretischen Informatik als *advocatus diaboli* eingesetzt werden, um zu beweisen, dass bestimmte Probleme schwierig oder unlösbar sind. Andererseits wollen wir aber auch *gutartige Probleme* betrachten und begründen, warum in diesen Fällen die Annahme eines böserartigen Gegenspielers zu pessimistisch ist – und was man statt dessen als Bewertungsgrundlage nehmen sollte.

1 Bösartige Gegenspieler

Man betrachte als einführendes Beispiel das Problem des Handlungsreisenden, der ausgehend von seiner Heimatstadt sieben Orte hintereinander besuchen und anschließend in seine Heimatstadt zurückkehren möchte. Hierfür sucht er die kürzeste Route. Ein *Programm* zur Lösung dieses Problems besteht offensichtlich darin, *alle möglichen* Reihenfolgen „auszumessen“ und die kürzeste zu nehmen. Bei sieben Orten gibt es $7 \cdot 6 \cdot \dots \cdot 2 \cdot 1 = 5040$ verschiedene Reihenfolgen, die man mit jedem Computer schnell durchprobieren kann, um so die kürzeste zu identifizieren.

Schwierigkeiten können allerdings in diesem Beispiel unter anderem schnell auftreten, wenn die Anzahl der Orte wächst, weil dann nicht mehr alle möglichen Reihenfolgen getestet werden können – schon bei 20 Orten braucht *jeder* Rechner für die 2432902008176640000 Reihenfolgen mehr als ein Jahr. Jetzt wird die Geschichte in gewisser Weise erst interessant. Unser Programm ist nicht in der Lage, das ihm gestellte Problem in der uns zur Verfügung stehenden Zeit immer optimal zu lösen. Wir stellen uns die folgenden zwei Fragen:

- A. Gibt es vielleicht überhaupt keine besseren Programme?
- B. *Wie* schlecht ist unser Programm eigentlich?

Im Folgenden wollen wir zunächst auf Frage **A** eingehen. Wenn wir versuchen, die Frage zu bejahen, müssen wir also zeigen, dass „bessere“ Programme unmöglich sind. Zentrale Akteure werden hierfür „böserartige Gegenspieler“ sein. Unmöglichkeitsergebnisse dieser Art stoßen oft auf Befremden, weil sie durch und durch negativ geprägt zu sein scheinen. Tatsächlich sind sie aber sehr nützlich: Die böserartigen Gegenspieler sind hier, wenn auch nicht ganz im Faustschen Sinne, „ein Teil von jener Kraft, die stets das Böse will und stets das Gute schafft“, denn sie versuchen dadurch, dass sie jedes nur mögliche Programm von vorneherein diskreditieren, uns von dem Zwang zu befreien, noch weiter zu suchen.

Diese Widersacher sind klassischerweise mit soviel Macht ausgestattet, dass wir in der Tat Frage **A** in vielen Fällen bejahen können. Was ist dann aber zu tun, wenn man erwiesenermaßen nichts mehr tun kann? Als Ausweg aus dem Jammertal bietet sich dann Frage **B**, mit der wir uns im zweiten Teil beschäftigen wollen.

Absolute Unmöglichkeit

Wir wenden uns zunächst Frage **A** zu. Wie beweist man, dass es etwas überhaupt nicht geben kann? Nichts ist unmöglich? Doch! Solche Unmöglichkeitserweise sind selten, aber dafür umso schöner. Wir demonstrieren das an der folgenden Fragestellung.

Eine elementare Weise, sich einer Menge von Objekten zu nähern, besteht darin, sie *abzuzählen*. Damit ist hier gemeint: Kann man die Objekte so in einer Reihenfolge anordnen, dass jedes Objekt nach endlich vielen Schritten mindestens einmal „an die Reihe kommt“? Interessant ist das natürlich nur bei unendlich großen Mengen. Versuchen wir es zunächst mit den *ganzen* Zahlen:

$$0, +1, +2, +3, \dots, -1, -2, -3, \dots$$

ist *keine* zulässige Reihenfolge, weil z.B. die Zahl -1 (und mit ihr alle negativen Zahlen) erst nach unendlich vielen Schritten erreicht werden. Aber so

$$0, +1, -1, +2, -2, +3, -3, +4, -4, \dots$$

geht es natürlich, jede Zahl x wird nach endlich vielen Schritten erreicht. Bei den *rationalen* Zahlen (das sind die „Brüche“) geht es auch noch. Wir wollen uns hier der Einfachheit halber auf die positiven beschränken. Zunächst erfassen wir die Zahlen in einer Tabelle und geben die Reihenfolge dann durch die Pfeile an.

	1	2	3	4	...			
1	$\frac{1}{1}$	→	$\frac{2}{1}$	→	$\frac{3}{1}$	→	$\frac{4}{1}$...
2	$\frac{1}{2}$	↙	$\frac{2}{2}$	↘	$\frac{3}{2}$	↘	$\frac{4}{2}$	↘
3	$\frac{1}{3}$	↘	$\frac{2}{3}$	↙	$\frac{3}{3}$	↘	$\frac{4}{3}$	
4	$\frac{1}{4}$	↙	$\frac{2}{4}$	↘	$\frac{3}{4}$	↘	$\frac{4}{4}$	
⋮	⋮	↘		↘		↘		⋮

Jetzt zu dem versprochenen Unmöglichkeitbeweis: Die *reellen Zahlen* sind nicht abzählbar. Zur Erinnerung: Reelle Zahlen sind genau diejenigen, aus denen der Zahlenstrahl besteht, und wir wollen uns hier auf jene beschränken, die zwischen 0 und 1 liegen. Warum sollte man sie nicht abzählen können? Angenommen, das ginge, und wir könnten sie in der entsprechenden Reihenfolge in ihrer Dezimaldarstellung auflisten, jede Zahl in eine neue Zeile, zum Beispiel:

0, **1**356
 0, 7**7**7777...
 0, 5 **■**
 0, 935**9**35935...
 0, 6504**2**42
 ⋮

Dadurch erhalten wir eine Tabelle mit unendlich vielen Zeilen (von denen einige auch unendlich lang sind).

Unser erster böser Gegenspieler betritt nun die Bühne und will uns eine Zahl präsentieren, die wir „vergessen haben“, also nicht nach endlich vielen Zeilen erreichen. Dazu nimmt er als Nachkommastellen die Ziffern aus der Diagonale (oben grau unterlegt). Wenn er in einer Zeile keinen Diagonaleintrag findet (weil die Dezimaldarstellung zu kurz war), interpretiert er das

als 0. In unserem Beispiel ergibt sich dadurch beispielsweise 0,17092... Anschließend erhöht er jede Ziffer um 1 (aus der 9 wird die 0) und erhält beispielsweise 0,28103... Der böse Gegenspieler fragt nun, ob diese neue Zahl denn auch in unserer Tabelle aufgelistet worden sei. Im Prinzip muss das so sein, denn schließlich handelt es sich um eine reelle Zahl, die zwischen 0 und 1 liegt. Wir nehmen also an, dass sie beispielsweise in der 173. Zeile steht. Das aber kann wiederum nicht sein, denn die 173. Nachkommastelle der neuen Zahl ist um eins höher als die 173. Nachkommastelle der 173. Zeile. Also steht dort nicht die neue Zahl.

Der böse Gegenspieler hat gewonnen, denn er verwickelt uns in jedem Fall in einen Widerspruch, und das bedeutet dass jeder Versuch, die reellen Zahlen so in eine lineare Anordnung zu bringen, dass jede irgendwann einmal vorkommt, scheitern muss. Wir haben also ein Beispiel für ein Problem gesehen, für das es gar kein Programm geben kann. Dieses Beispiel ist insofern besonders niederschmetternd (und dadurch in seiner Grausamkeit schon wieder schön), als dass es zeigt, dass es *gar keine Lösung geben kann, egal wer was wann wie und wie lange tut*. Es mag paradox klingen, aber in gewisserweise sind das – was die Eindeutigkeit der Antwort auf Frage **A** angeht – ähnlich paradiesische Zustände wie in dem Fall, wo alle mit der gefundenen Lösung zufrieden sind.

Reduktionen: Relative Unmöglichkeit

Kehren wir also zu unserem ersten Beispiel zurück, dem Problem des Handlungsreisenden, das wir im Folgenden kurz mit TSP (engl. für *Travelling Salesman Problem*) bezeichnen wollen. Wir nennen ein Programm *exakt*, wenn es immer die kürzeste Reihenfolge findet. Das naive Programm, das alle Reihenfolgen durchprobiert, ist beispielsweise exakt. Einzig, es dauert zu lange. Gibt es vielleicht ein schnelleres exaktes Programm? Oder kann man, ähnlich wie im zweiten Abschnitt, beweisen, dass jedes exakte Programm alle Möglichkeiten durchprobieren muss? Die enttäuschenden Antworten lauten, in umgekehrter Reihenfolge: *Nein*, man kann es nicht beweisen, denn: *Ja*, es gibt tatsächlich exakte Programme, die nie *wirklich alles* ausprobieren müssen – aber da sie im Prinzip immer noch *fast alles* ausprobieren müssen, sind auch diese Verfahren noch weit davon entfernt, als *effizient* bezeichnet werden zu können.

Bevor wir weiter diskutieren, ob es schnellere Verfahren geben kann oder nicht, müssen wir aber erst einmal festlegen, was wir mit *effizient* meinen. Es liegt auf der Hand, dass die Laufzeit unseres Programms von der „Größe“ der Eingabe abhängt – also beispielsweise der Anzahl der zu besuchenden Orte. (Mit „Eingabe“ ist im allgemeinen immer die individuelle Problemausprägung gemeint, hier also die konkreten Orte mit den dazwischen liegenden Abständen.) Wenn ein neuer Ort dazu kommt, dann würden wir sicher auch in Kauf nehmen, dass sich die Laufzeit um einen relativ geringen Betrag verlängert. Ein derartiges Verfahren bezeichnen wir als effizient. Wenn sich stattdessen die Laufzeit jedoch beispielsweise verdoppelt, dann können wir nicht mehr von Effizienz sprechen. Das populärste Anschauungsobjekt für derartiges, exponentielles Wachstum ist vielleicht das Schachbrett, auf dessen Feldern erst ein Reiskorn liegt, dann zwei, dann vier, dann acht; und das folglich auf dem letzten Feld mehr als alle Reiskörner der Erde versammelt. Das Laufzeitschicksal aller bekannten exakten Programme für das TSP fällt genau in diese Wachstumskategorie.

Betrachten wir also einen anderen, im wahrsten Sinne des Wortes *naheliegenden* Programmansatz. Der Handlungsreisende steuert zunächst den Ort an, der seiner Heimatstadt am nächsten liegt, von dort aus wiederum den nächstgelegenen unter den noch nicht besuchten, und immer so weiter. Wir konstatieren: Dies ist ein effizientes Programm, denn mit jedem dazukommenden Ort muss ja lediglich einmal mehr der nächstgelegene Ort ermittelt werden.

Aber: Das Programm findet nicht immer die kürzeste Route, wie das folgende Beispiel zeigt (Abbildung 1). Ernüchterung tritt ein und wir stellen uns abermals Frage **A**: Gibt es über-



Abbildung 1: Der Heimatort ist durch den weißen Kreis gekennzeichnet. Links die Route, die immer den nächstgelegenen Ort ansteuert, rechts eine kürzere (und optimale) Reihenfolge.

haupt effiziente exakte Programme für unser Problem, oder ist das unmöglich? Es ist jetzt an der Zeit, eine bislang stillschweigend gemachte Grundannahme thematisieren. Natürlich *gibt es* Problemausprägungen, bei denen das zuletzt genannte Programm die kürzeste Reihenfolge findet – etwa wenn die Orte alle in gleichem Abstand auf einem Kreis liegen. Und natürlich *gibt es* auch Problemausprägungen, bei denen bestimmte exakte Verfahren, die im allgemeinen nicht effizient sind, plötzlich sehr schnell zu einem optimalen Ergebnis kommen. Aber *wir beurteilen unsere Programme danach, wie sie im schlechtesten Fall abschneiden*, und das bedeutet: Wenn es für ein Programm Eingaben gibt, bei denen das Laufzeitverhalten nicht mehr effizient wächst (merke: das können wir nicht an *einer einzelnen* Eingabe feststellen), dann bezeichnen wir das Programm als ineffizient. Wenn es für ein Programm Eingaben gibt, bei denen es nicht mehr die beste Lösung findet, dann nennen wir das Programm nicht exakt.

Wir werden dieses Paradigma erst im zweiten Teil in Frage stellen (wenn wir uns Frage **B** widmen) und dann dort Alternativen diskutieren. Jetzt wollen wir es für unsere Zwecke benutzen, denn unser Ziel ist es ja gerade die Unmöglichkeit von exakten effizienten Programmen für unser Problem zu zeigen. In gewisser Weise sieht sich also jedes erdenkliche Programm einem bösen Gegenspieler ausgesetzt, der versucht, es auf eine der beiden skizzierten Weisen zu diskreditieren: zu lang (die Route), oder zu langsam (das Programm).

Für das Abzählbarkeitsproblem der reellen Zahlen konnten wir einen bösen Gegenspieler angeben, der jeden Lösungsversuch in einen Widerspruch verwickelt hat. Für das Problem des Handlungsreisenden ist das aus zwei Gründen schwieriger. Erstens: Die Menge der Programme bestand dort ganz konkret aus allen möglichen Reihenfolgen der reellen Zahlen, hier ist sie wesentlich diffuser und enthält alle „Rezepte“, zu einer Reihenfolge der Orte zu kommen. Zweitens war das Ziel dort klarer: Zeige, dass es überhaupt kein Programm geben kann. Hier wissen wir, dass es exakte Programme gibt, und der Gegenspieler soll zeigen, dass sie nicht effizient sein können.

Böse Gegenspieler dieses Kalibers hat man bis heute nicht (er)finden können. Gleichwohl ist nachwievor auch kein effizientes exaktes Programm für das Problem des Handlungsreisenden bekannt. Die Frage **A** ist für dieses Problem also noch offen. Was können wir tun, um einen eventuellen Auftraggeber davon zu überzeugen, dass es nicht unserer Dummheit, sondern der Natur des Problems zuzuschreiben ist, dass wir noch kein Programm gefunden haben? Tatsächlich ist die Frage **A** für hunderte andere Probleme ebenfalls noch ungeklärt, trotz intensiver Forschung sind für sie keine effizienten exakten Algorithmen bekannt. Überraschenderweise lässt sich dieser Umstand für unsere Zwecke verwenden. Wenn es uns gelänge, zu zeigen, dass

ein effizientes exaktes Programm für unser Problem zu einem effizienten exakten Programm für die anderen Probleme führen würde,

dann hätten wir ein Argument für die Schwierigkeit unseres Problems gefunden – denn schließlich haben viele Experten lange versucht, diese anderen Probleme zu lösen, und das ohne Erfolg.

Wir möchten also gerne Probleme in diesem Sinne miteinander vergleichen. Hier kommen wieder die bösen Gegenspieler ins Spiel. Angenommen, wir haben zwei Probleme X und Y . Deren Eingaben, also die individuellen Problemausprägungen, wollen wir mit I_X beziehungsweise I_Y bezeichnen. Dann sagen wir, dass sich X auf Y reduzieren lässt, wenn jede Eingabe I_X von X so in eine Eingabe I_Y von Y umgewandelt werden kann, dass sich aus der besten Lösung für I_Y effizient auch eine beste für I_X ableiten lässt.

Um das Konzept der Reduktion durch ein Beispiel mit Leben zu füllen, brauchen wir zunächst ein weiteres Problem und führen hierzu einige neue Begriffe ein. Ein *Netzwerk* besteht aus einer Menge von Punkten, von denen einige paarweise verbunden sind, andere nicht (siehe Abbildung 2). In einem solchen Netzwerk suchen wir nun einen in sich geschlossenen Weg, der entlang der Verbindungslinien *jeden Punkt genau einmal* besucht (die Verbindungslinien müssen dabei nicht alle genutzt werden). Ein solcher Kreis wird *Hamilton Kreis* genannt – zu Ehren von Sir William Rowan Hamilton, der 1857 ein Spiel erfand, bei dem es darum ging, die Ecken eines Dodekaeders so abzulaufen, dass jede Ecke genau einmal besucht wird. Natürlich hat nicht jedes Netzwerk einen Hamilton Kreis. Wir stellen uns also das folgende, kurz mit HK bezeichnete Problem: Zu einem gegebenen Netzwerk müssen wir entscheiden, ob es einen Hamilton Kreis besitzt, und wenn ja, ihn finden.

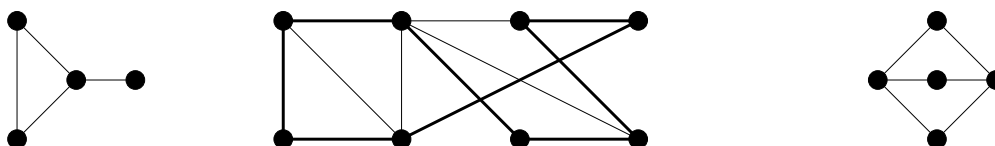


Abbildung 2: Links: ein Netzwerk. Mitte: ein Netzwerk mit Hamilton Kreis. Rechts: ein Netzwerk ohne Hamilton Kreis.

Wir reduzieren nun dieses Problem (HK) auf das Problem des Handlungsreisenden (TSP). Dazu machen wir aus einer Eingabe I_{HK} , die aus einem Netzwerk besteht, eine Eingabe I_{TSP} , indem die Punkte zu Orten werden. Jetzt müssen wir noch die Abstände zwischen den Orten festlegen: Zwei Orte, die als Punkte in dem Netzwerk verbunden waren, bekommen Abstand 1; zwei Orte, die nicht verbunden waren, Abstand 2. In I_{TSP} ist also (wie für die Eingaben des TSP erforderlich) jeder Ort mit jedem verbunden, und lediglich an dem Abstand kann man erkennen, ob das auch in I_{HK} der Fall war.



Abbildung 3: links: I_{HK} , rechts: I_{TSP} mit den jeweiligen Abständen.

So weit, so gut. Wie erhält man nun aber aus der optimalen Lösung für I_{TSP} eine optimale für I_{HK} ? Jede Route für I_{TSP} besteht aus genau so vielen Teilstrecken, wie wir Orte haben. Wenn also die Länge der kürzesten Route für I_{TSP} gleich der Anzahl der Orte ist, dann haben in ihr alle Teilstrecken Länge 1 und daher ist die Route immer nur zwischen Orten verlaufen,

die im Netzwerk auch verbunden waren. Die kürzeste Tour in I_{TSP} ist also genau ein Hamilton Kreis in I_{HK} . Und andersherum: Wenn die kürzeste Route für I_{TSP} länger als n ist, dann kann I_{HK} keinen Hamilton Kreis haben, denn der hätte in I_{TSP} ja eine Tour der Länge n induziert.

Wir haben es hier, mit anderen Worten ausgedrückt, mit einer Verschwörung des Bösen zu tun. Auch wenn wir für das Problem des Handlungsreisenden keinen bösen Gegenspieler gefunden haben, der uns von der absoluten Unmöglichkeit eines effizienten exakten Programms hat überzeugen können, so haben wir dennoch gezeigt: Gäbe es ein Programm, das mit jedem TSP-Gegenspieler fertig wird, dann könnten wir jeden HK-Gegenspieler in einen TSP-Gegenspieler überführen, diesen besiegen, und die Lösung in eine für das HK-Problem zurückübertragen. Wir haben also ein starkes Indiz dafür, dass jedes TSP-Programm einen unbesiegbaren bösen Gegenspieler hat (ohne dass wir letzteren wirklich präsentieren könnten).

Angenommen, wir haben eine ansehnliche Sammlung von Problemen, für die alle keine effizienten exakten Programme bekannt sind, und die sich alle paarweise aufeinander reduzieren lassen. Dann bedeutet das nicht weniger, als dass die (effiziente exakte) Lösbarkeit eines dieser Probleme die (effiziente exakte) Lösbarkeit aller Probleme zur Folge hätte. Nehmen wir nun weiter an, dass ein neues Problem Z auftaucht, das noch nicht klassifiziert ist, für das wir aber den gleichen Schwierigkeitsgrad zeigen wollen. Müssen wir es dann mit allen Problemen in unserer Sammlung vergleichen? Nein, zum Glück nicht, denn wenn *eines* der gesammelten Probleme Y sich auf Z reduzieren lassen, wir aber bereits wissen, dass sich *jedes* Problem X der Sammlung auf Y reduzieren lässt, dann ist es nicht schwer einzusehen, dass sich *jedes* Problem auch auf Z reduzieren lässt. Auf diese Weise kann man mit beschränktem Aufwand die Sammlung der Probleme Stück für Stück erweitern. Die Frage, ob nicht doch für alle diese Probleme effiziente exakte Programme existieren, wird somit immer unwahrscheinlicher. Sie ist – wenn auch hier etwas verkürzt dargestellt – unter dem formelhaftem Kürzel $P = NP?$ bekannt und zählt zu den sieben offenen Millennium-Problemen, für deren Beantwortung das Clay Mathematics Institute im Jahr 2000 jeweils eine Million Dollar ausgelobt hat.

2 Gutartige Probleme

Im Prinzip haben wir im ersten Teil dieser Arbeit die Grundzüge der Komplexitätstheorie dargestellt, wie sie sich seit den 70er Jahren etabliert hat. Sie hat eine große Vielzahl von Problemen klassifiziert, für die es (gemäß dieser Klassifikation) vermutlich keine exakten effizienten Verfahren geben kann. Diese Aussage steht aber in vielen Fällen diametral entgegengesetzt zu den Erfahrungen, die man in der Praxis mit diesen schwierigen Problemen macht. Bei vielen dieser Probleme wird nämlich beobachtet, dass man sie eigentlich „recht gut“ und „recht schnell“ lösen kann. So kann man beispielsweise das Problem des Handlungsreisenden für reale Eingaben mit weitüber 10.000 Städten nachweisbar optimale Lösungen finden.

Die klassische Komplexitätstheorie bietet hierfür weder Erklärungen, noch stellt sie Ansatzpunkte zur Verfügung, um dieses gutartige Verhalten der Probleme beweisbar zu machen. Sie kann das auch nicht können, da sie (wie wir bereits feststellten) Programme danach bewertet, wie sie sich verhalten, wenn sie auf den für sie schlimmsten Gegenspieler treffen. Dieser worst-case Pessimismus ist aber in vielen Fällen für die Praxis nicht repräsentativ, weil er den bösen Widersachern zu viel Macht einräumt. In realen Anwendungen sind Eingaben nicht sorgfältig konstruiert, sondern breit gestreut oder oft auch insofern *gutartig*, als dass sie mit natürlichen strukturellen Eigenschaften ausgestattet sind.

Was kann nun also die Theorie leisten, um so vage Beobachtungen wie „dieses Programm klappt eigentlich ganz gut“ zu untermauern? Im zweiten Teil unserer Betrachtungen wollen wir uns jetzt mit der Frage **B** auseinandersetzen, und zu einer realistischeren Einschätzung unserer Programme kommen. Der zentrale Ansatzpunkt wird sein, Verfahren nach ihrem *durchschnittlichen* Verhalten zu beurteilen.

Die naheliegende Interpretation dieses Ansatzes ist der folgende. Wir haben ein bestimmtes Problem, betrachten die Menge aller individuellen Problemausprägungen (Eingaben) und ein bestimmtes Programm. Wir nehmen an, dass alle Eingaben gleichwahrscheinlich sind, und mitteln die Güte oder die Laufzeit des Programms über alle diese Eingaben.

Hier sehen wir uns nun vor die Schwierigkeit gestellt, dass wir die konkrete Eingabe, also den genauen Gegenspieler, mit dem das Programm fertig werden soll, gar nicht vorliegen haben, sondern mit einem etwas unscharfen durchschnittlichen Widersacher konfrontiert sind. Was wir also benötigen, sind Aussagen darüber, welche Eigenschaften wir *typischerweise* von einer Eingabe erwarten dürfen. Hier liegt natürlich auch genau unsere Chance. Gelingt es uns, einige gutartige Struktureigenschaften zu finden, die bei der überwältigenden Mehrheit der Eingaben gegeben sind, dann können wir bei der Entwicklung der Programme natürlich darauf achten, dass letztere erstere schonungslos ausnutzen. Bei einer worst-case Betrachtung kämen wir damit nicht durch, denn natürlich wird es Eingaben geben, die diese Eigenschaften nicht erfüllen, aber durchschnittlich betrachtet ist das selten der Fall und fällt daher nicht signifikant ins Gewicht.

Während üblicherweise mathematische Sätze besagen, dass alle Elemente X der Menge Y die Eigenschaft Z haben, brauchen wir also Aussagen der Form *fast alle Elemente X der Menge Y haben die Eigenschaft Z* . Hiermit ist nicht etwa gemeint, dass zum Beispiel 99 Hundertstel der Elemente in Y die Eigenschaft Z haben, sondern, viel stärker, dass der Anteil derer mit Eigenschaft Z immer weiter wächst und gegen Eins strebt, wenn die Grundgesamtheit der Menge X wächst. Umgekehrt bedeutet das, dass der relative Anteil der Minderheit, die Z nicht erfüllen, immer geringer wird. Wir sagen, dass ein *zufällig ausgewähltes* Element X aus Y *fast sicher* die Eigenschaft Z hat, oder dass Z eine *typische* Eigenschaft für Y ist.

Fast alle, fast sicher

Nach dieser recht abstrakten Einführung in den zweiten Teil ist es Zeit für ein Beispiel. Wir betrachten wieder das Problem HK, bei dem man entscheiden muss, ob ein gegebenes Netzwerk einen Hamiltonkreis besitzt (und ihn gegebenenfalls auch finden soll). Wie sieht nun ein *durchschnittliches* Netzwerk aus? Wir machen dazu folgendes Gedankenexperiment. Die Punkte liegen dabei zunächst unverbunden vor. Für jedes Punkte-Paar x, y werfen wir nun eine Münze, und verbinden die beiden Punkte genau dann, wenn Kopf oben liegt.

In diesem Zufallsmodell ist tatsächlich jedes Netzwerk gleichwahrscheinlich, denn damit es durch den Zufallsprozess erzeugt wird, muss die Münze für jedes Paar, ob verbunden oder nicht, die „richtige“ Entscheidung treffen. Was haben wir durch dieses Modell nun gewonnen? Man kann damit gut rechnen. Fixieren wir einen beliebigen Punkt, dann ist jeder weitere mit Wahrscheinlichkeit $1/2$ mit ihm verbunden. Durchschnittlich wird er also zu ungefähr der Hälfte der anderen Punkte verbunden sein. Analog dazu haben zwei Punkte ungefähr ein Viertel der übrigbleibenden Punkte als *gemeinsame* Nachbarn, und für drei Punkte erwarten wir ein Achtel. Es ist eine leichte Übung (wenn man mit dem nötigen Handwerkszeug ausgestattet ist) zu beweisen, dass diese drei Eigenschaften in fast allen Netzwerken in allen Punkten erfüllt sind.

Hierbei handelt es sich in der Tat um elementare Eigenschaften, aber überraschenderweise garantieren sie (wenn sie erfüllt sind), dass das Netzwerk einen Hamilton Kreis besitzt, und dass wir diesen auch effizient finden können. Dies beruht im Wesentlichen darauf, dass wir zunächst mit einem kleinen Kreis beginnen können, und ihn dank der erwähnten Eigenschaften Schritt für Schritt erweitern können, bis wir schließlich jeden Punkt genau einmal besuchen. Wir wollen an dieser Stelle auf eine detaillierte Beschreibung verzichten und verweisen auf Abbildung 4, in der exemplarisch zwei der Erweiterungsschritte gezeigt werden. Wir haben also ein Programm, das das als schwierig bekannte Problem, einen Hamiltonkreis zu finden, für *fast alle* Netzwerke effizient löst.

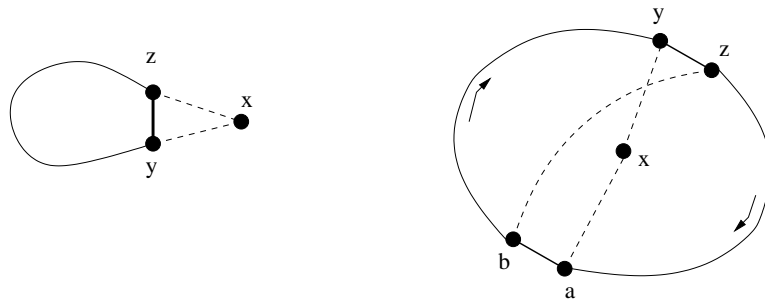


Abbildung 4: Erweiterungsschritte eines Verfahrens, um einen Hamilton Kreis zu finden. Links und rechts wird jeweils der Punkt x neu zum Kreis hinzugefügt.

Qualitätssicherung

Was passiert in den Fällen, in denen das Programm scheitert? Hierbei muss es sich also um Eingaben handeln, die die besagten Eigenschaften nicht erfüllen – und wir können ausrechnen (und das unterscheidet unser Ergebnis von einer vagen Beobachtung der Art „meistens geht’s gut“), wie groß die Minderheit derjenigen Netzwerke höchstens ist, bei denen dies passiert. Falls es passiert, dann wissen wir zunächst nicht, ob das Netzwerk einen Hamilton Kreis besitzt (und wir ihn nur nicht gefunden haben), oder nicht. Aber diese Fälle sind so unwahrscheinlich, dass wir es uns *dann* leisten können, noch ein exaktes, nicht-effizientes Programm zu bemühen, das ähnlich wie beim Problem des Handlungsreisenden alle Möglichkeiten durchprobiert und uns so Gewissheit verschafft. Gemittelt über alle Eingaben ergibt das immer noch eine effiziente durchschnittliche Laufzeit.

Zunächst hatten wir also ein Programm, das *immer* effizient und *fast immer* zu einer richtigen Antwort gelangt. Wir haben es mit einem einfachen Trick in ein Programm verwandelt, das *immer* die richtige Antwort ermittelt und *im Durchschnitt* eine gute Laufzeit hat – indem wir in den unwahrscheinlichen Fällen, die von der Norm abweichen, die „Notbremse“ gezogen haben und auf ein langsames, exaktes Verfahren zurückgegriffen haben. Je unwahrscheinlicher der Notfall, desto teurer darf die Notbremse sein.

Die Umwandlung war unter anderem deshalb so einfach, weil das Problem nur zwei Antworten zulässt (es gibt einen Hamilton Kreis, oder es gibt keinen). Bei Problemen, die eine größere Bandbreite haben, ist das oft nicht mehr der Fall, und hier treten kurioser Weise Schwierigkeiten genau dann auf, wenn die Problemeingabe *bösartig gut* ist. Wir stellen uns dazu ein Problem vor, bei dem beispielsweise eine bestimmte Größe X minimiert werden soll – so, wie bei dem Problem des Handlungsreisenden die Länge der Tour. Angenommen wir

wissen, dass bei einer typischen Eingabe der optimale Wert für X ungefähr ℓ beträgt. Nehmen wir weiter an, dass wir ein Programm entwickelt haben, das fast sicher einen Wert ℓ' für X erreicht, der nur unwesentlich größer als ℓ ist. Dieses Programm errechnet also mit hoher Wahrscheinlichkeit eine nahezu optimale Lösung. Was können wir tun, um das zu *garantieren*?

Die eine unsichere Komponente betrifft zunächst unser Programm. Falls es wider Erwarten den prognostizierten X -Wert von ℓ' nicht erreicht, dann merken wir das und können, da dies genügend unwahrscheinlich ist, wie bei dem Hamilton Kreis Problem wieder ein teures exaktes Lösungsverfahren bemühen und damit den optimalen X -Wert bestimmen.

Nehmen wir also an, dass unser Programm tatsächlich den X -Wert ℓ' erreicht. Die andere unsichere Komponente besteht in dem Verhalten des optimalen X -Werts – er könnte ja untypischerweise weit besser (also kleiner) als ℓ sein, und damit wäre unsere berechnete Lösung ℓ' nicht nahe genug am Optimum. Zugegeben, auch der Fall ist unwahrscheinlich, und wenn wir wüßten, dass er eingetreten ist, könnten wir in bewährter Manier die Notbremse ziehen. Nur: Woher wissen wir, ob der optimale X -Wert der Eingabe stark von dem erwarteten Wert ℓ abgewichen ist? Oder um mit den Friedrich Christoph Oetinger zugeschriebenen Worten zu sprechen:

„Herr, gib mir die Kraft zu ändern, was ich ändern kann,
die Gelassenheit hinzunehmen, was ich nicht ändern kann,
und die Weisheit, zwischen beidem zu unterscheiden.“

Wir müssen also möglichst schnell eine gewisse Vorahnung dafür entwickeln, ob eine Eingabe böswartigerweise von der Norm abweicht, denn wir können uns ja nicht *jedes Mal* einen teuren Test leisten, sondern dürfen das nur tun, wenn wir Anzeichen dafür haben, dass eine starke (und damit unwahrscheinliche) Ausnahmesituation vorliegt. Genau das versuchen Programme zu tun. Sie suchen in den Eingaben nach sogenannten Zertifikaten, die effizient überprüfbar sind und die garantieren, dass sich ein optimaler X -Wert nicht weit vom erwarteten Wert ℓ entfernen kann. Im Falle des Handlungsreisenden-Problems könnten diese zum Beispiel aus kleineren Kollektionen von paarweise weit auseinander liegenden Orten bestehen. Finden sie solche Garantien, dann lehnen sie sich zufrieden zurück. Bleiben sie aus, dann ist das ein erstes Anzeichen für eine starke Abweichung vom typischen Verhalten, und das rechtfertigt dann eine genauere (und teurere) Untersuchung – beispielsweise, indem weitere, aufwändigere Zertifikate ermittelt werden. Dies kann mehrmals hintereinander geschaltet werden, bis am Ende die Ausnahmen so selten sind, dass man den optimalen X -Wert wieder durch vollständiges Durchprobieren ermitteln kann.

Evolution und Phasenübergänge

Bislang haben wir uns auf die Grundannahme gestützt, dass alle Eingaben *gleichwahrscheinlich* sind. Auf dieser Hypothese aufbauend haben wir dann den Blick für das Wesentliche geschärft, indem wir von typischen Eigenschaften der Eingaben gesprochen haben und damit solche gemeint haben, die von der überwältigenden Mehrheit erfüllt wurden. In vielen Situationen kann diese Grundannahme natürlich völlig verkehrt sein – vielleicht will es das Unglück, dass ausgerechnet in der uns interessierenden Anwendung die Eingaben fortwährend aus genau jener Minderheit stammen, für die unser Programm nicht gut funktioniert.

Wie kann man also die Landschaft der typischen Eigenschaften relativieren, variieren und so den Bedürfnissen genauer anpassen? Rufen wir uns als Beispiel das Gedankenexperiment

zur Generierung eines zufälligen Netzwerks in Erinnerung, in dem jedes Punkte-Paar mit Wahrscheinlichkeit $1/2$ verbunden wurde. Das führt natürlich dazu, dass durchschnittlich die Hälfte aller möglichen Verbindungen existieren. Wir vergleichen das nun mit einem anderen Zufallsmodell. Hierbei sucht sich jedes Punkte-Paar zunächst eine zufällige Zahl zwischen 0 und 1 als „Geburtstermin“ aus. Anschließend werden die Verbindungen in der Reihenfolge ihrer Geburtstermine geboren, und wir beobachten das dabei entstehende Netzwerk N . Offensichtlich besitzt N am Ende des Prozesses alle möglichen Verbindungen, aber wenn wir es in dem Moment anhalten, wo die Hälfte aller möglichen Verbindungen existiert, dann hat N fast sicher genau die gleichen Eigenschaften wie ein zufälliges Netzwerk aus dem ersten Modell, beispielsweise fast sicher einen Hamilton Kreis.

Tatsächlich könnten wir aber den Zufallsprozess genauso gut schon früher anhalten und uns fragen, ob denn der erste Hamilton Kreis in N vielleicht schon viel früher auftaucht – und wenn ja, *wann*? Faszinierenderweise gibt es dafür nämlich wirklich einen scharfen Schwellenwert. Angenommen, wir haben 100 Punkte, dann erscheint der erste Hamilton Kreis, wenn ungefähr 6% aller möglichen Verbindungen existieren, bei 1000 Punkten reichen bereits 0,9%, und allgemein gilt, dass der Schwellenwert für einen Hamilton Kreis für n Punkte bei einem Bruchteil von $(\ln n + \ln \ln n)/n$ aller möglichen Verbindungen liegt.

Auf den ersten Blick möchte man hierauf einwenden, dass der Erscheinungspunkt doch wohl von der Reihenfolge abhängen muss, in der die Verbindungen auftauchen, aber genau das ist (fast sicher) nicht der Fall: Für wachsendes n strebt die Wahrscheinlichkeit dafür, dass bei zufälliger Wahl der Geburtstermine der erste Hamilton Kreis bei dem oben genannten Schwellenwert auftritt, gegen Eins. Wir können also vorhersagen, wann bei fast allen Reihenfolgen ein Hamilton Kreis das erste Mal auftaucht. Analoge Untersuchungen lassen sich natürlich auch für andere Eigenschaften durchführen. Der Schwellenwert dafür, dass jeder Punkt jeden anderen über eine Reihe von Verbindungen erreichen kann, ist beispielsweise identisch mit dem Zeitpunkt, ab dem jeder Punkt zu mindestens einem weiteren Punkt direkt verbunden ist. Und nur kurze Zeit später, nämlich genau dann, wenn jeder Punkt zu mindestens *zwei* Punkten direkt verbunden ist, bildet sich der erste Hamilton Kreis.

Gewinnen wir wieder ein wenig Abstand zu den technischen Details. Was wir hier gesehen haben, ist ein Beispiel für die Evolution eines Netzwerkes. Zu jedem Zeitpunkt können wir nach seinen typischen Eigenschaften fragen und erhalten auf diese Weise ein sich laufend veränderndes Landschaftsbild: einige Eigenschaften verschwinden, andere tauchen auf – frappierend ist dabei insbesondere die Geschwindigkeit, mit der sich solche Phasenübergänge vollziehen. Die Evolutionsgeschichte der Menge der Eingaben eines Problems führt so zu einem tieferen Verständnis der typischen Strukturen der Eingaben.

Resümee

Im ersten Teil dieser Arbeit haben wir Probleme kennen gelernt, für die keine zufriedenstellenden Programme bekannt sind. Wir haben uns der Frage **A** gestellt, ob diese Probleme vielleicht unlösbar sind, und versucht diese Vermutung mit Hilfe bössartiger Gegenspieler zu belegen. Wir haben kurz ein unlösbares Problem betrachtet und anschließend verstanden, wie sich aus der Erkenntnis, dass sich eine Vielzahl von Problemen aufeinander zurückführen lassen, weitere Indizien für ihren hohen Schwierigkeitsgrad ergeben. Unsere Unzufriedenheit mit den existierenden Lösungsverfahren lag allerdings im Wesentlichen darin begründet, dass wir ihre Leistung danach beurteilen, wie sie *bei der für sie bössartigsten Eingabe* abschneiden.

Im zweiten Teil haben wir genau diese Grundannahme fallen gelassen, und gemäß Fra-

ge **B** überlegt, ob es vielleicht Verfahren gibt, die bei *fast allen Eingaben* gut funktionieren. Wir haben ein Beispiel für ein schwieriges, aber gutartiges Problem kennen gelernt, bei dem die überwiegende Mehrheit der Eingaben hilfreiche Struktureigenschaften besitzt, so dass das Problem in fast allen Fällen optimal gelöst werden kann. Wir haben uns anschließend Gedanken darüber gemacht, wie man in den wenigen Ausnahmefällen trotzdem mit durchschnittlich effizientem Aufwand optimale Ergebnisse erzielen kann – und wie man überhaupt erkennt, dass es sich um solche Ausnahmen handelt. Der letzte Abschnitt hat sich dann damit befasst, wie sich typische Strukturen verändern können.

Untersucht man das durchschnittliche Verhalten von Programmen, dann sind Aussagen über die *typischen* Eigenschaften der Eingaben essenziell. Man beachte, dass mit dieser Einstellung der wissenschaftliche Reduktionismus noch um eine wesentliche Nuance verschärft wird. Ursprünglich beschränkt er sich ja darauf, die unterschiedlichen Elemente einer Menge als von ein-und-der-selben-Sorte zu betrachten, wenn sie alle eine bestimmte Eigenschaft Z erfüllen. Jetzt tun sie das noch nicht einmal *alle*, sondern nur *fast alle*, und trotzdem wird zunächst der ganzen Familie der Stempel Z aufgedrückt. Das entspricht in etwa dem Übergang von klassischer Mechanik zu statistischer Physik, oder dem Konflikt zwischen Konsensdemokratie und Mehrheitsentscheidungen. Der erste Teil dieser Arbeit hat uns gelehrt, dass wir, wenn wir jeder Eingabe ein Veto einräumen, unsere böartigen Gegenspieler so mächtig machen, dass effiziente Lösungsverfahren aller Voraussicht nach unmöglich sind. Der zweite Teil zeigt uns auf, welche Möglichkeiten sich uns bieten, wenn wir bereit sind, die Vollkasko-Mentalität abzulegen.

Literatur

Der Beweis der Nichtabzählbarkeit der reellen Zahlen mit Hilfe des Diagonalisierungsarguments geht auf Cantor (1873) zurück. Eine der ersten grundlegenden Arbeiten zum Thema Reduzierbarkeit ist

- Karp, Richard: Reducability among combinational problems. In: Miller, R.E.; Thatcher, J.W. (Hg.): *Complexity of Computer Computations*. New York 1972, 85–103.

Als Standardwerk gilt außerdem

- Garey, Michael R.; Johnson, David S. *Computers and Intractability. A Guide to the Theory of NP-completeness*. New York 1979.

Als Referenz für den Stand der Forschung für das Problem des Handlungsreisenden sei hier verwiesen auf die *Traveling Salesman Problem homepage*:

- <http://www.math.princeton.edu/tsp/index.html>

Die sieben Millenium-Probleme finden sich unter

- http://www.claymath.org/Millennium_Prize_Problems/

Die folgende Monographie bietet einen exzellenten Überblick über die Evolution zufälliger Netzwerke:

- Bollobás, Béla: *Random Graphs*. Cambridge 2001.