

Klaus Bovermann

17. Dezember 2019

Rucksack-Problem

Der Leistungskurs Informatik beschäftigt sich mit der Entwicklung *iterativer und rekursiver Algorithmen unter Nutzung der Strategien „Modularisierung“, „Teilen und Herrschen“ und „Backtracking“* (siehe Kernlehrplan Informatik, Inhaltsfeld 2). Die Schülerinnen und Schüler *implementieren iterative und rekursive Algorithmen auch unter Verwendung von dynamischen Datenstrukturen, sie testen Programme systematisch anhand von Beispielen und mit Hilfe von Testanwendungen*. Weiterhin beurteilen sie *die Effizienz von Algorithmen unter Berücksichtigung des Speicherbedarfs und der Zahl der Operationen*.

Im Hinblick auch auf Aspekte der theoretischen Informatik *untersuchen und beurteilen sie die Grenzen des Problemlösens mit Informatiksystemen* (siehe Kernlehrplan Informatik, Inhaltsfeld 5).

In diesem Zusammenhang bietet sich ein, auch für Schüler, interessantes Beispiel an, mit dem Themen wie

- Komplexität von Algorithmen
- NP-Vollständigkeit
- genetische Algorithmen

behandelt werden können. Es handelt sich um ein Standardproblem der Informatik, das sog. Rucksack-Problem (engl. *knapsack problem*). Z.B. möchte man für den Ausflug sinnvolle Gegenstände in einen Rucksack packen. Jeder dieser Gegenstände hat ein bestimmtes Gewicht und einen für den Ausflug bestimmten Nutzwert. Leider kann man den Rucksack nur mit einem begrenzten Gewicht beladen.

Ein anderes Szenarium¹ bietet die NASA, die anbot, zusätzliche Last (maximal 645 kg) in der Rakete zur ISS mitzunehmen. Diverse Forschungsstationen machten Angebote, in denen stand, wieviel die Ausrüstung für ihr Experiment wiegt und was sie dafür bereit waren, zu zahlen.

Auch im Transportwesen sind solche Probleme denkbar, wobei es darum geht, LKWs, Container, Eisenbahnwaggons usw. optimal zu beladen.

Jetzt stellt sich die Frage, welche der Gegenstände sollten eingepackt werden, um einen maximalen Gesamtwert zu erzielen, ohne das zulässige Gesamtgewicht zu überschreiten.

¹ Siehe dazu *Taschenbuch der Algorithmen*; Vöcking et.al.; Springer; Seite 405 ff

Eine formale Beschreibung des Problems:

Aus einer Menge von Objekten, die jeweils ein Gewicht und einen Wert haben, sollen einige der Objekte ausgewählt werden.

1. Das Gesamtgewicht der ausgewählten Objekte darf einen vorgegebenen Wert nicht überschreiten.
2. Unter dieser Bedingung soll der Gesamtwert der ausgewählten Objekte maximiert werden.

Und eine mathematische Problemformulierung:

Gegeben eine Menge $M = \{ (g_1, w_1), (g_2, w_2), (g_3, w_3), \dots, (g_n, w_n) \}$ von Zahlenpaaren $(g_i, w_i) \in \mathbb{N} \times \mathbb{N}$ sowie eine Zahl $k \in \mathbb{N}$.

Gesucht ist eine 0-1-Folge $(b_1, b_2, b_3, \dots, b_n)$, so dass

1. $\sum_{i=1}^n (b_i \cdot g_i) \leq k$
2. $\sum_{i=1}^n (b_i \cdot w_i)$ maximal

Beispiel

Gegeben ist ein Rucksack mit einer maximalen Tragfähigkeit von 15 kg. Die Attribute (Gewicht, Wert) der 10 möglichen Gegenstände:

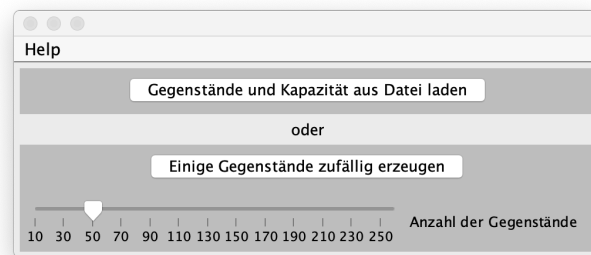
	Gewicht in g	Wert in €
1	3500	375
2	2500	300
3	2000	100
4	3000	225
5	1000	50
6	1750	125
7	750	75
8	3000	275
9	2500	150
10	2250	50

Packt man z.B. die Gegenstände 1, 4 6, 8 und 9 in den Rucksack, so hat man 13,750 kg mit einem Gesamtwert von 1150 € eingeladen. Ist das optimal?

Die beigefügte Software kann uns helfen, Einblicke in die Lösung des Problems zu bekommen.

Die Software

Das Programm startet mit einem kurzen Dialog:



Der Startbildschirm

Sie können hier ein Rucksack-Problem zufällig erzeugen oder die relevanten Daten über einen File-Input-Dialog aus einer Textdatei (der sog. Konfigurationsdatei) laden. Die Textdatei für das obige Beispiel hat den folgenden Inhalt:

```
KAP : 15000

; Name : Gewicht : Wert

Gegenstand 1 : 3500 : 375
Gegenstand 2 : 2500 : 300
Gegenstand 3 : 2000 : 100
Gegenstand 4 : 3000 : 225
Gegenstand 5 : 1000 : 50
Gegenstand 6 : 1750 : 125
Gegenstand 7 : 750 : 75
Gegenstand 8 : 3000 : 275
Gegenstand 9 : 2500 : 150
Gegenstand 10: 2250 : 50
```

Eine Zeile², die mit dem Schlüsselwort KAP beginnt, legt (nach dem Doppelpunkt : als Trenner) die Kapazität des Rucksacks fest. Die restlichen Zeilen beschreiben die möglichen Gegenstände in Form von

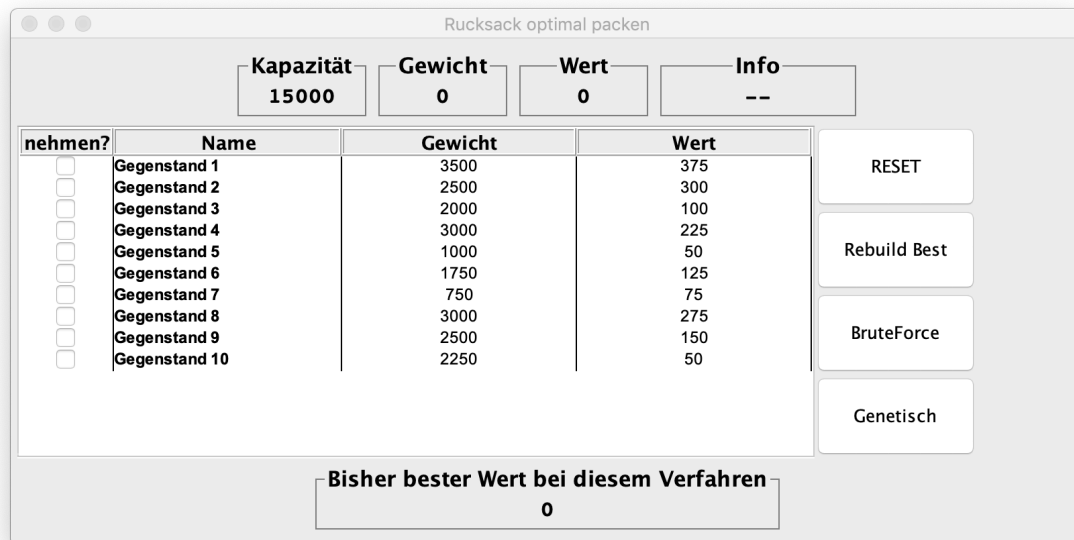
```
Name:Gewicht:Wert
```

mit zwei Doppelpunkten als Trenner.

Leerzeilen und Zeilen, die mit einem Semikolon ; beginnen, werden beim Einlesen ignoriert.

Nach Auswahl der gewünschten Rucksack-Datei öffnet sich das Hauptfenster der Anwendung. Für das obige Beispiel stellt sich das Fenster so dar:

² Fehlt diese Zeile, ist als Kapazität der Wert 100 voreingestellt.



nehmen?	Name	Gewicht	Wert
<input type="checkbox"/>	Gegenstand 1	3500	375
<input type="checkbox"/>	Gegenstand 2	2500	300
<input type="checkbox"/>	Gegenstand 3	2000	100
<input type="checkbox"/>	Gegenstand 4	3000	225
<input type="checkbox"/>	Gegenstand 5	1000	50
<input type="checkbox"/>	Gegenstand 6	1750	125
<input type="checkbox"/>	Gegenstand 7	750	75
<input type="checkbox"/>	Gegenstand 8	3000	275
<input type="checkbox"/>	Gegenstand 9	2500	150
<input type="checkbox"/>	Gegenstand 10	2250	50

Kapazität: 15000 Gewicht: 0 Wert: 0 Info: --

RESET

Rebuild Best

BruteForce

Genetisch

Bisher bester Wert bei diesem Verfahren: 0

Der Bildschirm zum Erforschen der optimalen Belegung

Wie findet man eine optimale Belegung?

Eine erste Idee: Probieren

Sie haben hier jetzt die Möglichkeit, Gegenstände in den Rucksack zu packen, indem Sie in der ersten Spalte die Check-Boxen benutzen. Dabei sehen Sie in den Feldern oberhalb der Tabelle das Gesamtgewicht und den Gesamtwert der aktuell benutzten Gegenstände. Übersteigt das Gesamtgewicht die Kapazität des Rucksacks, werden Sie über eine Anzeige in der Info-Box benachrichtigt.

In dem Feld unterhalb der Tabelle sehen Sie den bisher besten Gesamtwert, den Sie durch Ihr Probieren erzielt haben. Diesen Wert und die dazugehörige Auswahl können Sie jederzeit durch den Button

Rebuild Best

rekonstruieren.

Aufgabe:

Probieren Sie einmal, ob Sie die optimale Belegung finden (oder gibt es mehrere gleichwertige?). Testen Sie auch andere Rucksack-Probleme, indem Sie entsprechende Textdateien erzeugen.

Zweite Idee: Brute Force

Wir könnten ja einfach alle Möglichkeiten ausprobieren und würden dabei diejenige finden, die unter den zulässigen den größten Wert garantiert.

Welche Möglichkeiten gibt es denn?

Man kann jeden Gegenstand nehmen und einpacken — oder auch nicht. Eine Idee besteht also darin, eine mögliche Rucksack-Packung zu beschreiben mit Hilfe einer 0-1-Folge. In dem vorliegenden Beispiel (es gibt 10 Gegenstände) ist z.B. die Folge 1001010110 ein Modell dafür, die Gegenstände 1, 4, 6, 8 und 9 einzupacken. Sie repräsentiert damit eine zulässige Bepackung (13,750 kg ist unterhalb der Kapazität von 15 kg) und hat einen Wert von 1150 €.

Die Brute-Force-Methode probiert jetzt alle möglichen 0-1-Folgen der Länge 10 aus und sucht (und findet) diejenige, die zulässig ist und den besten Wert liefert.

Wie viele solcher Folgen gibt es?

Kombinatorische Überlegungen sagen uns, dass es insgesamt $2^{10} = 1024$ solcher Folgen gibt.

Aufgabe:

Starten Sie mit Hilfe der Software für das Beispiel den Brute-Force-Algorithmus mit dem entsprechenden Button und interpretieren Sie die Protokolldatei, die sich nach Ende des Vorgangs öffnet.

Das klingt überzeugend, denn diese Methode findet sicher die beste Lösung. Doch:

Aufgabe:

Stellen Sie sich vor, es gäbe nicht nur 10, sondern 100 Gegenstände, die Sie zur Auswahl haben.

- a) *Wie viele Möglichkeiten gibt es dann?*
- b) *Wie lange wird der Algorithmus benötigen, wenn pro Sekunde 1 Million Möglichkeiten untersucht werden können?*

Lösung:

- a) *Es gibt $2^{100} \approx 10^{33}$ Möglichkeiten.*
- b) *Der Algorithmus benötigt also ca. 10^{30} Sekunden. Eine sehr großzügige Abschätzung ergibt eine Zeit von ca. 10^{19} Jahren. Das Universum existiert angeblich erst seit ca. 14 Milliarden Jahre.*

Der Aufwand für den Brute-Force-Algorithmus ist offenbar proportional zu 2^n . Wenn also die Anzahl der vorhandenen Gegenstände um 1 vergrößert wird, verdoppelt sich die Zeit, die zum Finden der optimalen Auswahl nötig ist.

Dritte Idee: Der relative Wert (Nutzen)

Die beiden Gegenstände Nr. 2 und Nr. 9 haben zwar ein gleiches Gewicht (2500), jedoch hat der Gegenstand 2 einen doppelt so hohen Wert. Man wird also sicher den Gegenstand 2 bevorzugen.

Das führt uns zu der Idee, die Gegenstände nach ihrem Nutzen zu beurteilen:

Wir verstehen unter dem Nutzen eines Gegenstandes den relativen Wert, also den Wert pro Gewichtseinheit.

Für einen Gegenstand G ist also:

$$\text{Nutzen}_G = \text{Wert}_G / \text{Gewicht}_G$$

Wenn man jetzt also den Nutzen eines jeden Gegenstandes berechnet, ergibt sich die folgende Tabelle:

Nutzen der Gegenstände			
Gegenstand	Gewicht	Wert	Nutzen
1	3500	375	0,1071
2	2500	300	0,1200
3	2000	100	0,0500
4	3000	225	0,0750
5	1000	50	0,0500
6	1750	125	0,0714
7	750	75	0,1000
8	3000	275	0,0917
9	2500	150	0,0600
10	2250	50	0,0222

Der Gegenstand mit dem größten Nutzen ist Gegenstand 2. Man wird ihn also als Erstes einpacken.

Hinweis zur Software:

Schließen Sie das Simulationsfenster, ändern Sie in der Datei **RUCKSACK.ini** den dort eingetragenen Wert von **no** auf **yes** und laden Sie die Rucksack-Konfigurationsdatei erneut.

Sie sehen jetzt in dem Simulationsfenster eine weitere Spalte.

nehmen?	Name	Gewicht	Wert	Nutzen
<input type="checkbox"/>	Gegenstand 1	3500	375	0,1071
<input type="checkbox"/>	Gegenstand 2	2500	300	0,12
<input type="checkbox"/>	Gegenstand 3	2000	100	0,05
<input type="checkbox"/>	Gegenstand 4	3000	225	0,075
<input type="checkbox"/>	Gegenstand 5	1000	50	0,05
<input type="checkbox"/>	Gegenstand 6	1750	125	0,0714
<input type="checkbox"/>	Gegenstand 7	750	75	0,1
<input type="checkbox"/>	Gegenstand 8	3000	275	0,0917
<input type="checkbox"/>	Gegenstand 9	2500	150	0,06
<input type="checkbox"/>	Gegenstand 10	2250	50	0,0222

Kapazität: 15000, Gewicht: 0, Wert: 0, Info: --

Bisher bester Wert bei diesem Verfahren: 0

Der Bildschirm zum Erforschen der optimalen Belegung mit Hilfe der Information über den Nutzen

Die Idee kann jetzt umgesetzt werden:

Füge dem Rucksack der Reihe nach denjenigen Gegenstand hinzu, der (unter den noch nicht hinzugefügten Gegenständen) den größten Nutzen hat, und zwar so lange, wie die Kapazität nicht überschritten wird.

Das bedeutet für unser Beispiel:

- Packe Gegenstand 2 ein; Gesamtwert ist jetzt 300, Gesamtgewicht 2500
- Packe Gegenstand 1 ein; Gesamtwert ist jetzt 675, Gesamtgewicht 6000
- Packe Gegenstand 7 ein; Gesamtwert ist jetzt 750, Gesamtgewicht 6750
- Packe Gegenstand 8 ein; Gesamtwert ist jetzt 1025, Gesamtgewicht 9750
- Packe Gegenstand 4 ein; Gesamtwert ist jetzt 1250, Gesamtgewicht 12750
- Packe Gegenstand 6 ein; Gesamtwert ist jetzt 1375, Gesamtgewicht 14500
- Jetzt müsste man Gegenstand 9 einpacken. Doch damit wäre die Kapazität überschritten. Das Verfahren bricht also ab.

In der vorliegenden Software wird dieses Verfahren aktiviert mit

Greedy

Aufgabe:

Starten Sie mit Hilfe der Software für das Beispiel den Greedy-Algorithmus mit dem entsprechenden Button und interpretieren Sie die Protokolldatei, die sich nach Ende des Vorgangs öffnet.

Untersuchen Sie ebenfalls andere Rucksäcke.

Der gierige Algorithmus liefert hier tatsächlich die optimale Belegung, die wir per Brute-Force bereits ermittelt haben; und das mit wesentlich geringerem Aufwand.

Doch leider ist das nicht immer so!

Dazu betrachten wir ein anderes Beispiel (Kapazität = 17):

Gegenstand	Gewicht	Wert	Nutzen
a	3	3	1,0000
b	4	5	1,2500
c	4	5	1,2500
d	6	10	1,6667
e	6	10	1,6667
f	8	11	1,3750
g	8	11	1,3750
h	9	13	1,4444

Die optimale Belegung empfiehlt uns, die Gegenstände c, d und e einzupacken; das ergibt einen Wert von 25. Leider erzeugt der gierige Algorithmus die Lösung (d, e) mit einem Wert von nur 20. Den Grund erkennt man sofort; denn nachdem der Gegenstand e eingepackt wurde, wählt das Greedy-Verfahren jetzt den Gegenstand h. Er ist nämlich der Gegenstand unter den noch nicht benutzten Gegenständen mit dem größten Nutzen. Doch leider ist dann die Kapazität überschritten.

Eine Verbesserung ergibt sich, wenn man (immer noch gierig) die Liste der unbenutzten Gegenstände — im Hinblick auf den Nutzen in absteigender Reihenfolge — weiter untersucht. Man probiert also die Gegenstände f und g (leider auch vergebens), um dann endlich Gegenstand c einzupacken. Die Gegenstände b und a sind dann nicht mehr möglich, so dass das Verfahren endet und die optimale Lösung (d, e, c mit Wert 25) liefert.

Dieses intelligentere Greedy-Verfahren können Sie ebenfalls mit der Software aktivieren:

Greedy (intelligenter)

Doch auch dieses Verfahren liefert nicht immer die optimale Belegung:

Gegenstand	Gewicht	Wert	Nutzen
a1	5	8	1,6000
a2	5	8	1,6000
a3	6	6	1,0000
a4	8	5	0,6250
a5	10	10	1,0000
a6	11	5	0,4545
a7	12	10	0,8333
a8	15	17	1,1333
a9	15	20	1,3333
a10	30	20	0,6667

Kapazität = 30

Nachdem der Brute-Force-Algorithmus uns verraten hat, dass die optimale Belegung einen Wert von 38 hat (a2, a5, a9), liefern beide Greedy-Algorithmen den Wert von 36 (a1, a2, a9).

Ein weiteres Beispiel (zugegeben sehr überspitzt konstruiert) zeigt uns, dass das Greedy-Verfahren eine Belegung liefert, die sehr weit von der optimalen Lösung entfernt ist:

Gegenstand Nummer	Gewicht	Wert	Nutzen
0	1	1	1
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	1
10	100	99	0,99

Wenn der Rucksack eine Kapazität von 100 hat, sehen Sie sicherlich schnell, dass man nur den 10-ten Gegenstand einpacken sollte; denn dann hat man bei einem Gewicht von 100 einen Wert von 99 erreicht. Der Greedy-Algorithmus nimmt jedoch alle anderen Gegenstände und packt damit Gegenstände im Gesamtwert von nur 9 ein.

Fazit:

Im Allgemeinen findet das Greedy-Verfahren leider nicht garantiert eine optimale Lösung.

Es kann nicht einmal garantiert werden, dass die vom Greedy-Verfahren gelieferte Belegung der optimalen Lösung nahe kommt.

Weiterhin können wir die Brute-Force-Methode nicht benutzen, da sie bei praxisnahen Problemen mit vielen Gegenständen einen nicht-akzeptablen Zeitaufwand benötigt.

Leider gibt es bis heute kein Verfahren, das in akzeptabler Zeit eine optimale Lösung liefert, so dass wir uns auf die Suche nach einem Verfahren machen müssen, das in vertretbarer Zeit eine Belegung findet, die *nicht allzu weit* von der optimalen Belegung entfernt ist.

Neue Idee: Wir beobachten die Natur

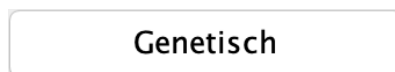
Die Lebewesen auf der Erde haben sich im Laufe der Erdgeschichte entwickelt und sich dabei an Umweltbedingungen (mehr oder weniger) gut angepasst. Die Wissenschaft spricht hier von der **Evolution**.

Ohne hier auf die sehr komplexen biologischen Details näher einzugehen, kann man das Prinzip (sicherlich sehr verkürzt) mit Hilfe von biologischen Begriffen beschreiben:

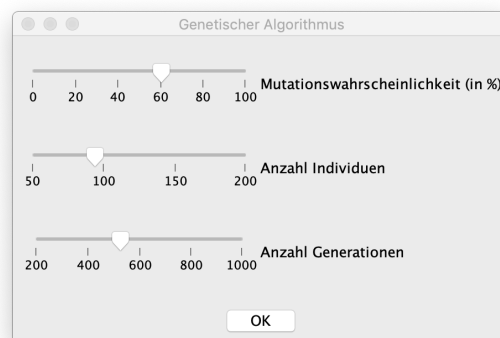
- Ein **Individuum** ist im Wesentlichen charakterisiert durch einen sogenannten **Gencode**.
 - ▶ Ein **Gen** ist ein Element aus einer Menge von zulässigen Symbolen. Wir verwenden hier die Symbolmenge $\{0, 1\}$.
 - ▶ Ein **Gencode** definiert ein sog. Individuum und ist eine Kette (ein n -Tupel) von Genen. Diese Tupel können als 0-1-Bitstring der Länge n , also z.B. (10010110001), definiert werden. Dabei ist n die Anzahl der Gegenstände, die in den Rucksack gepackt werden können. Eine 1 an der i -ten Stelle des Bitstrings bedeutet, dass der i -te Gegenstand eingepackt wird, eine 0 an dieser Stelle zeigt an, dass der i -te Gegenstand nicht eingepackt wird.
 - ▶ Ein Individuum ist ggf. nicht **lebensfähig** (d.h. zulässig).
 - ▶ Im Rucksack-Beispiel ist ein Individuum nur dann zulässig, wenn das Gesamtgewicht der eingepackten Gegenstände die Rucksack-Kapazität nicht übersteigt.
- Eine **Population** ist eine Menge von Individuen.
- Im Hinblick auf eine für das Überleben in der Umwelt notwendige Eigenschaft hat jedes Individuum eine sogenannte **Fitness**.
 - ▶ In unserem Rucksack-Beispiel könnte diese Fitness sinnvollerweise der Gesamtwert aller mit einer 1 codierten Gegenstände sein.
- Zwei Individuen sind in der Lage, neue Individuen zu generieren. Dabei werden die jeweiligen Gencodes nach bestimmten Regeln zu einem neuen Gencode kombiniert.
 - ▶ Eine einfache Regel besteht darin, dass aus den beiden Gencodes durch Kreuzung zwei neue Gencodes entstehen. Beispielsweise könnten zwei solcher Gencodes nach der Position 4 gekreuzt werden, so dass aus den beiden Ketten (10010110001) und (011101011010) die neuen Ketten (100101011010) und (01110110001) erzeugt werden.
- Eine **Mutation** eines Gencodes verändert (mindestens) ein Gen innerhalb des Bitstrings.
 - ▶ Mit einer gewissen Wahrscheinlichkeit wird ein Symbol an einer zufällig gewählten Stelle ersetzt durch ein anderes Symbol. So kann z.B. die Mutation bewirken, dass aus dem Individuum (1001010**1**1010) das Individuum (1001010**0**1010) wird (hier wurde zufällig das 8. Gen mutiert).

- Ausgehend einer Population P mit n Individuen besteht ein Evolutionsschritt jetzt aus den folgenden Teilschritten:
 1. Erzeuge eine neue leere Population Q
 2. Solange in Q noch nicht n Individuen enthalten sind, wiederhole:
 - 2.1. Wähle aus P zwei Individuen $I1$ und $I2$ aus.
 - 2.2. Erzeuge durch Kreuzung zwei neue Individuen $N1$ und $N2$.
 - 2.3. Mutiere $N1$ und $N2$ mit einer gewissen Wahrscheinlichkeit.
 - 2.4. Füge $N1$ und $N2$ (ggf. auch $I1$ und $I2$) zu Q hinzu, falls sie zulässig sind.
 3. Mache Q zur neuen Population.
- Ein solcher Evolutionsschritt wird jetzt mehrfach (sehr oft, vielleicht endlos) vollzogen. Dabei ist nach jedem Schritt das Individuum mit der besten Fitness ein Maß für die Güte der aktuellen Population.

Die Software aktiviert einen solchen Prozess mit:



Es öffnet sich ein weiterer Dialog, in dem Sie gewisse Parameter setzen können, mit denen die Evolution zu beeinflussen ist:



- Mutationswahrscheinlichkeit:
 - Dieser Wert steuert die Häufigkeit, mit der ein neu-generiertes Individuum mutiert.
- Anzahl Individuen:
 - Jede Population umfasst genau diese Anzahl an Individuen.
- Anzahl Generationen:
 - Ein Evolutionsschritt wird so oft wie angegeben ausgeführt.

Nach dem letzten Schritt wird als Ergebnis das fitteste Individuum als Lösung gesetzt.

WIE KANN MAN EINEN RUCKSACK OPTIMAL BELADEN?