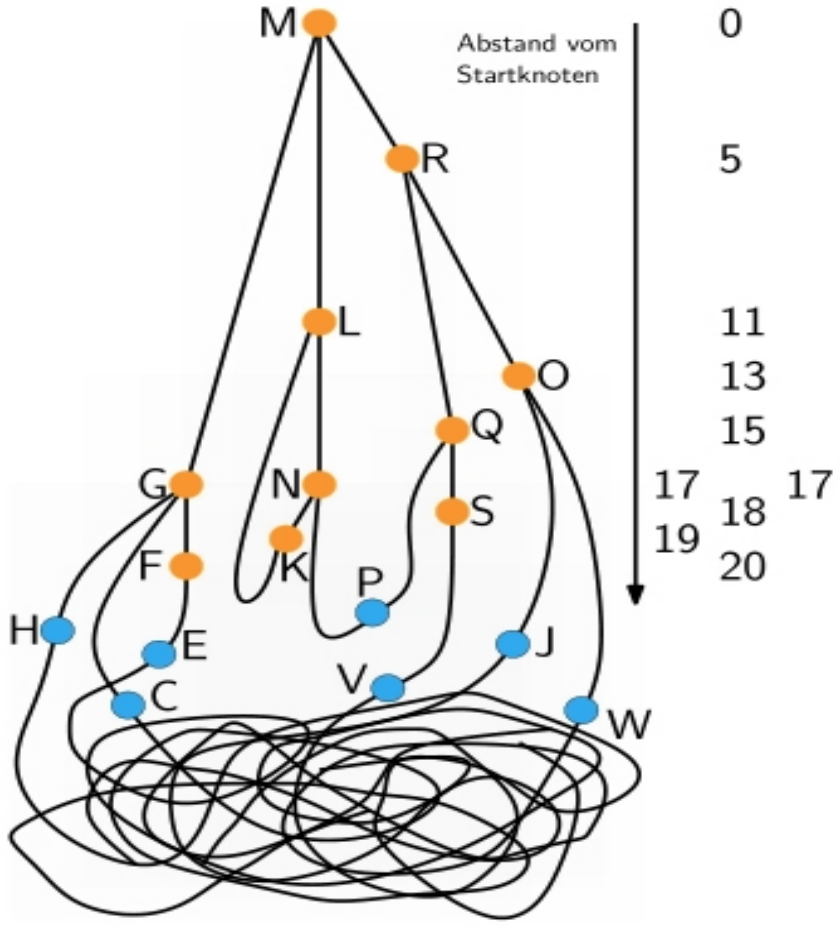
# Wegoptimierung mit dem Dijkstra-Algorithmus


*Grafiken aus dem "Taschenbuch der Algorithmen" S.345, S.346*

# Inhaltsverzeichnis

## *Allgemeines*

## Optimierungsprobleme

Immer, wenn die optimale Lösung für ein Problem gesucht wird, liegt ein Optimierungsproblem vor. Oft sind sehr viele mögliche Lösungen vorhanden und manchmal ist die beste erreichbare Lösung nur eine angenäherte.

Beispiele sind:

- Das Finden kürzester Wege von A nach B  mit
    - dem **Dijkstra-Algorithmus** / dem Bellman-Ford-Algorithmus / Algorithmus von Floyd und Warshall
    - Random Bounce
    - Simple Trace / Robust Trace
    - Breadth-First / Depth-First / IDDF / Best-First
    - A* / D*/ IDA*
- Das Finden minimaler aufspannender Bäume (Verbindungen zwischen verschiedenen Objekten mit minimalem Aufwand)
    - Algorithmus von Kruskal
    - Algorithmus von Borůvka
    - Algorithmus von Prim
- Planen mit maximalen Flüssen (Simulation von Straßenverkehr, Leitungsnetze für Datenverkehr, Wasser, Strom etc.)
    - Algorithmus von Goldberg und Tarjan
    - Flows in Networks, Ford und Fulkerson
- Zuordnungsprobleme (möglichst viele möglichst passende Objekte / Zustände / Personen sollen zueinander gebracht werden; z.B. bei Partnerschaftsvermittlung)
    - Verfahren von Hopcroft und Karp
    - Heiratssatz von Hall

## Wegoptimierung

Wegoptimierung ist das Finden des kürzesten Weges von A nach B über gegebene Wege. Bei der Wegfindung müssen sogar noch die Wege erstellt bzw. gefunden werden.

Computer können und sollten solche Probleme lösen, weil Optimierungen automatisierbar sind und weil Menschen nicht schnell und genau genug arbeiten können.

Alle Wegoptimierungsverfahren basieren auf einem sogenannten Graphen. Dieser Graph ist eine Abstraktion der Welt, die der Computer für Berechnungen braucht. Ein Graph ist ein Netzwerk aus Kanten und Knoten. Knoten sind die Endpunkte von Kanten und Kanten sind die Verbindungen zwischen den Knoten.

## *Einführung zum Dijkstra-Algorithmus*

## Über den Erfinder

Edsger Wybe Dijkstra [...] (* 11. Mai 1930 in Rotterdam; † 6. August 2002 in Nuenen, Niederlande) war ein niederländischer Informatiker. Er war der Wegbereiter strukturierter Programmierung. [...]

Unter seinen Beiträgen zur Informatik finden sich Dijkstras Algorithmus zur Berechnung des kürzesten Weges in einem Graphen, die erstmalige Einführung von Semaphoren zur Synchronisation zwischen Threads und das damit zusammenhängende Philosophenproblem sowie eine Abhandlung über den Goto-Befehl und warum er nicht benutzt werden sollte. Er führte den Begriff der strukturierten Programmierung in die Informatik ein.

Im Jahre 1972 erhielt Dijkstra den Turing Award. -Wikipedia

## Der Algorithmus von Dijkstra

Der Algorithmus von Dijkstra (nach seinem Erfinder Edsger W. Dijkstra) dient der Berechnung eines kürzesten Pfades zwischen einem Startknoten und einem beliebigen Knoten in einem kantengewichteten Graphen. -Wikipedia

In anderen Worten: Der Algorithmus von Dijkstra wird verwendet, um den kürzesten Weg von einem Startpunkt zu einem Zielpunkt in einem Netz aus Wegen und Kreuzungen. Grundlage für die Berechnung sind die Längen der Wege zwischen den Kreuzungen.

Und mit Worten der Informatik / Mathematik: Mit dem Algorithmus von Dijkstra berechnet man den kürzesten Weg zwischen 2 Knoten über Kanten und Zwischenknoten anhand der Kantengewichtung.

Die Kantengewichtung muss nicht unbedingt nur auf der Kantenlänge basieren. Hier kann auch die nötige Zeit für das Bewegen über die Kante berücksichtigt werden. So kann man z.B. bei einem Routenplaner-Programm Autobahnen und 30er-Zonen unterschiedlich bewerten. Außerdem kann man der Kantengewichtung einen Bonus bzw. Abzug geben, je nachdem ob der Weg über die Kante besser oder schlechter ist, als es nur nach der Länge oder Bewegungsdauer aussehen würde. Dabei darf aber keine Kantengewichtung negativ sein, da sonst der Algorithmus ein falsches Ergebnis liefern würde. Aber man kann leicht dafür sorgen, dass „Einbahnstraßen“ verwendet werden können.

## Warum der Dijkstra-Algorithmus wichtig ist

Der Algorithmus von Dijkstra als erster Wegoptimierungs-Algorithmus ist die Basis für die meisten Algorithmen, wie z.B. der Bellman-Ford-Algorithmus, Algorithmus von Floyd und Warshall und A* und dessen Erweiterung IDA*. Vor allem der IDA*-Algorithmus wird momentan für Künstliche Intelligenzen in der professionellen Spielentwicklung benutzt. Es werden auch ähnliche Verfahren für die KIs von Robotern verwendet. Und natürlich werden sie verwendet, wenn der Mensch den schnellsten Weg sucht, also in Routenplanern und in vielem mehr.

## Grenzen des Dijkstra-Algorithmus

Macht es Sinn, bei einer Route von Spanien nach China alle kleinen Feldwege zu beachten? Eher nicht. Selbst wenn hier nur Autobahnen berücksichtigt werden, dauern Berechnungen mit diesem Algorithmus durch die Massen an Knoten und Kanten ziemlich lange. Im Allgemeinen kann man

jedes Gebiet in Felder eingeteilt werden. So kommt es ziemlich schnell Gebieten zu einer großen Zahl von Kanten und Knoten. Man kann entweder den Graph so reduzieren, dass nur die nötigsten Elemente vorkommen, oder man betrachtet die Wege mit der höchsten Erfolgsaussicht zuerst, wie beim A*-Algorithmus. Dort werden als erstes die Wege mit der kleinsten Abweichung von der Start-Ziel-Richtung geprüft.

Wie Anfangs erwähnt dürfen Kantengewichtungen nicht negativ sein, wenn der Dijkstra-Algorithmus funktionieren soll.

Der Algorithmus von Dijkstra sowie die meisten anderen Wegoptimierungsalgorithmen können nicht flexibel auf einen veränderten Graphen reagieren, das heißt, der Algorithmus muss jedes mal neu ablaufen, wenn eine Kante, ein Knoten, Start- oder Zielpunkt geändert werden. Der D*-Algorithmus schafft hier Abhilfe, hat aber andere Probleme.

## *Wie und warum der Algorithmus von Dijkstra funktioniert*

### Grundidee

Im „Taschenbuch der Algorithmen“ wird sehr schön Dijkstras Algorithmus mit dem langsamen Aufheben eines Fadennetzes verglichen. Die Kurzfassung geht so: Man hat eine Karte, legt Fäden den Wegen entlang und verknotet kreuzende Schnüre (Knoten). Dann habt man den Knoten am Startpunkt immer weiter hoch. Im Algorithmus werden noch liegende Knoten als „wartend“ und angehobene als „angehoben“ bezeichnet (in manchen Texten sind die Bezeichnungen für den Status anders). Dabei lösen sich die Knoten nacheinander vom Boden. Wenn alle Knoten in der Luft hängen, sind die kürzesten Verbindungen erkennbar, denn der kürzeste Weg vom Startknoten zum Zielknoten kann nur über gespannte Kanten gehen.

### Aufbau

Der Algorithmus von Dijkstra besteht aus folgenden Schritten:

1. Man initialisiert alle Knoten mit der Distanz „unendlich“, was eine Zahl sein muss, die größer als die maximale Distanz zu allen Knoten ist
2. Der Startknoten wird auf die Warteliste gesetzt
3. Die Distanz des Startknotens ist 0
4. Wiederhole 4. und 7. bis es keine Knoten mehr auf der Warteliste gibt
5. Der Knoten auf der Warteliste mit der kleinsten Distanz wird angehoben
6. Die Distanzen über diesen aktuellen Knoten zu allen benachbarten Knoten berechnet nach: Distanz des Nachbarknoten = Distanz des aktuellen Knoten + Verbindung zwischen Nachbar- und aktuellem Knoten
7. Wenn eine so berechnete Distanz kleiner ist, als der vorhandene Wert, dann wird diese neue Distanz beibehalten. Dies nennt sich Update und lässt den Algorithmus „durch hängende Fäden“ aussortieren.
8. Alle benachbarten Knoten werden auf die Warteliste gesetzt, wenn sie noch nicht darauf sind

Nach dem Durchlauf hat man nun die kürzesten Distanzen vom Startknoten zu allen anderen Knoten. Man weiß also nicht über welche Kanten man gehen muss. Dies kann man relativ einfach beim Update in Schritt 6 speichern. Man merkt sich einfach welcher Knoten der aktuelle war, als es ein Update für die Distanz gab.

## *Implementierung*

### Allgemeines

Insgesamt hab ich mich hauptsächlich an die Hinweise und Erklärungen im „Taschenbuch der Algorithmen“ gehalten. Im Internet war erstaunlich wenig zum Dijkstra-Algorithmus zu finden. Es war kein Beispiel in VB zu finden und in C++ nur wenige, teils unverständliche mit Syntaxfehlern. Zum A*-Algorithmus gibt es allerdings ziemlich viel. Außerdem wurde nicht viel über die verwendeten Datenstrukturen gesagt.

### Warum Visual Basic

Abgesehen von meinem positiven persönlichen Bezug zu Visual Basic 6 gibt es einige Gründe für diese Entscheidung. VB wird zwar oft als Anfängersprache bezeichnet, hat aber tatsächlich alles, was für professionelle Programmierung nötig ist. Zwar gibt es kein so ausgeprägtes Pointersystem wie in C++, aber das ist in diesem Fall nicht besonders schlimm, da es sich nur um ein kleines Beispielprogramm handelt und Pointer bringen bei Longinteger-Variablen nicht viel Zeitersparnis. Der Fokus liegt ganz klar auf dem Algorithmus von Dijkstra, wovon C++ und viele andere Programmiersprachen mit ihren größeren Aufwand bei Grafischen Oberflächen, Datentypumwandlungen und ähnlichem ablenken würden.

### Datenstrukturen

Den Algorithmus als solches zu implementieren war nicht so schwierig. Er erfordert aber ziemlich umfangreiche Vorbereitungen, wie die Datenstrukturen. Nötig wurden 2 benutzerdefinierte Datentypen (Klassen in C++) für Kanten und Knoten. Kanten enthalten Informationen über ihren Start- und Endknoten und ihre Länge. Knoten enthalten Informationen über ihre Koordinaten und ihre Verbindungen. Diese Verbindungsinformationen sagen, mit welchen anderen Knoten sie verbunden sind und über welche Kante. Die Distanzen vom Startknoten werden in einem normalen Fließkommazahlen-Array gespeichert.

### Datenverwaltung

Das nächste Aufwändige bei der Implementierung war das Hinzufügen und Entfernen von Datensätzen. Man muss Kanten und Knoten schließlich erstellen können, wenn man den Graph ändern will muss man sie löschen können und wenn man alles neu machen will, müssen die Daten komplett entfernt werden können. Das Programm hängt neue Datensätze an das Ende der Arrays und alte werden durch Nachrutschen der hinteren Datensätze gelöscht. Dies hat sich als anfangs sehr fehleranfällig herausgestellt. Besonders kompliziert war, dass die Verbindungsinformationen der Knoten auf bestimmte Kanten verweisen und wenn diese Kanten bearbeitet werden, müssen die Informationen entsprechend angepasst werden.

### Funktionen des fertigen Programms

#### *Menü*

1. Datei
    1. Speichern – Speichert den aktuellen Graphen in einer Textdatei
    2. Laden – Lädt den gespeicherten Graphen aus einer Textdatei; der aktuelle Graph wird

vorher gelöscht

3. Beenden – Beendet das Programm

2. Ansicht
    1. Hintergrund
        1. Bild einfügen – Öffnet ein Dialogfenster und setzt das ausgewählte Bild als Hintergrund
        2. Bild entfernen – Entfernt das aktuelle Hintergrundbild
    2. Informationen
        1. Knotenindex – Zeigt den Index jedes Knotens an
        2. Kantenindex – Zeigt den Index jeder Kante an
        3. Distanzen – Zeigt die Distanzen vom Startknoten zu den anderen Knoten an
        4. Kantenlängen – Zeigt die Länge aller Kanten an

### *Aktionen*

1. Knoten erstellen – Bei einem Linksklick auf die Bildfläche (normalerweise der weiße umrandete Bereich) wird ein Knoten erstellt. Neue Knoten müssen mindestens 10 Pixel von allen anderen Knoten entfernt sein (Übersichtlichkeit). Knoten werden als schwarze Punkte dargestellt.
2. Kante erstellen – Der erste Linksklick (man muss nicht auf den Knoten klicken, nur in die Nähe) wählt den Startknoten einer Kante aus, der zweite den Zielknoten. Kanten werden als schwarze Striche dargestellt. Der erste Knoten wird mit einem blauen Ring markiert. Beide Knoten müssen unterschiedlich sein. Der Vorgang kann mit einem Rechtsklick abgebrochen werden. Die Länge der Kanten wird mit schwarzer Schrift angezeigt.
3. Knoten bewegen – Drücken der linken Maustaste in der Nähe des zu bewegenden Knotens und Bewegen der Maus verschiebt den Knoten (Drag & Drop). Die Lage der Kanten wird dabei aktuell gehalten.
4. Knoten löschen – Ein Linksklick in die Nähe eines Knotens macht, dass dieser Knoten und alle Kanten an ihm gelöscht werden. (Das war ziemlich aufwendig zu programmieren ;-)
5. Kante löschen – Linksklick auf den Mittelpunkt einer Kante löscht die Kante.
6. Kürzesten Weg berechnen – Linksklick auf Startknoten macht, dass die kürzesten Wege vom Startpunkt zu allen Knoten mit dem Dijkstra-Algorithmus berechnet werden. Die Distanzen werden mit roter Schrift rechts vom betreffenden Knoten angezeigt. Dieser Schritt muss gemacht werden, damit der nächste (7.) erfolgen kann. So lange die ausgerechneten Werte stimmen, wird der Startknoten mit einem grünen Ring markiert.
7. Weg abfahren – Ein Linksklick auf den Zielknoten lässt ein Auto den kürzesten Weg vom Start- zum Zielknoten abfahren. Der Weg wird mit dicken schwarzen Strichen markiert, das aktuelle Wegstück wird gelb hervorgehoben und der Zielknoten hat einen roten Ring. Nachdem das Auto das Ziel erreicht hat, kann ein neuer Zielknoten angefahren werden. Während der Fahrt sind keine Aktionen möglich.
8. Reset – Löscht den gesamten Graphen, das Hintergrundbild bleibt

Das Textanzeigefeld (Label) gibt Informationen zu den Aktionen.

### *Geschwindigkeit*

Wem die Fahrt zu lange dauert, der kann das Auto mit dem horizontalen Scrollbalken schneller machen und umgekehrt.

## Probleme und Besonderheiten

### *Warum Verbindungsinformationen abgespeichert werden*

In der Praxis ist es so, dass Programme mit hohen Hardwareanforderungen, wie z.B. Spiele, Filmbearbeitung, Bildbearbeitung und Simulationen den Prozessor und vor allem den Grafikprozessor sehr belasten, es im Arbeitsspeicher aber Reserven gibt. Das liegt daran, dass man kaum den RAM mit normalen Variablen füllen kann. Nicht einmal riesige mehrdimensionale String-Arrays füllen diesen Speicher. Außerdem werden große Daten wie z.B. 3D-Daten, Bilder etc. eher im Grafikkartenspeicher gelagert. Man sollte also größere Variablenmengen in kauf nehmen, um Rechenleistung zu sparen.

### *Vorteile von Longinteger als Indexvariablen*

Es stimmt, dass der gesamte Wertbereich eines Longintegers in diesem Programm wohl nie für einen Index verwendet wird. Es wird also Arbeitsspeicher verschwendet. Wie eben erwähnt, soll man lieber mehr Daten im Speicher ablegen, um die CPU zu schonen. Das ist hier auch der Fall, denn Longinteger brauchen zwar mehr Speicherplatz, können dafür aber schneller verarbeitet werden. Das hat etwas damit zu tun, das Longinteger-Variablen 32 Bits Speicher benötigen und die meisten Prozessoren mit 32 Bit-Technologie arbeiten. Auf 64-Bit-Systemen gibt es auch einen sog. Largeinteger, der mit seinen 64 Bits Speicherbedarf dort am besten passt.

## Ergebnisdarstellung

Der Algorithmus von Dijkstra war implementiert und konnte die Längen der Wege von Startknoten zu allen anderen Knoten berechnen. Aber was bringt es das Wissen, dass der kürzeste Weg x Einheiten lang ist, man aber nicht weiß, über welche Knoten dieser kürzeste Weg geht? Da für die Berechnung des kürzesten Weges nach jeder Inkrementierung der großen Do-While-Schleife durch die Zählvariablen klar ist, über welchen Knoten man muss, kann man dieses Wissen in einem Datenfeld abspeichern. Im Beispielprogramm sieht das so aus: toNode(u) = v und bedeutet, man muss über Knoten v, um zu Knoten u zu kommen. Also kann man die einzelnen Elemente dieses Arrays aneinander ketten und bekommt so die Reihenfolge der zu besuchenden Knoten vom Zielknoten über Zwischenknoten zum Startknoten. Also falsch herum. Das alleine bringt nicht viel, da man bei einer Reise beim Startpunkt beginnt. Die Liste der zu besuchenden Knoten muss also irgendwie umgedreht werden. Wenn man die noch nicht umgedrehte Reiseroute auflöst, lautet der letzte Eintrag so: Um zu Zwischenknoten x zu kommen, muss man vom Startknoten kommen. Oder anders gesagt, der nächste zu besuchende Knoten in der richtigen Reihenfolge ist der Knoten x. Wenn man sich dann zu diesem Knoten bewegt hat, könnte man den Dijkstra-Algorithmus nochmal ablaufen lassen für die aktuelle Position und wieder die Reiseroute rückwärts ablaufen bis zum nächsten Knoten, was aber ein unnötiger Rechenaufwand wäre. Oder man beendet das rückwärts Ablaufen der Reiseroute an der aktuellen Position. Man tut also einfach so, als ob die aktuelle Position der Startpunkt wäre. Und so geht man weiter bis man am Ziel ist.

# *Anhang*

## Begriffserklärung

**Optimierungsprobleme**

Immer, wenn die optimale Lösung für ein Problem gesucht wird liegt ein Optimierungsproblem vor.

Oft sind sehr viele mögliche Lösungen vorhanden und manchmal ist die beste erreichbare Lösung nur eine angenäherte.

Im mathematischen Sinne ist Optimierung, wenn man den Minimal- oder Maximalwert einer Funktion errechnet

**Wegoptimierung**

Wegoptimierung ist das finden des kürzesten Weges von A nach B über gegebene Wege.

**Wegfindung**

Wegfindung ist das finden des kürzesten Weges von A nach B um Hindernisse herum. Ein Wegfindungsalgorithmus muss also seine Wege selbst erstellen bzw. erkennen.

**Knoten**

An einem Knoten treffen sich mindestens 2 Wege. Lässt sich mit einer Straßenkreuzung vergleichen.

**Kante**

Eine Kante verbindet 2 Knoten. Lässt sich mit einer Straße vergleichen.

**Algorithmus**

Ein Algorithmus ist eine Folge von Anweisungen, die zusammen eine Lösung für ein Problem finden sollen.Er lässt sich mit einem Kochrezept vergleichen.

**Heuristik**

Heuristik (altgr. εὑρίσκω *heurísko* „ich finde“; *heuriskein*, „(auf-)finden“, „entdecken“) bezeichnet die Kunst, mit begrenztem Wissen und wenig Zeit zu guten Lösungen zu kommen. -Wikipedia

In der Informatik bedeutet dies, dass man mit Hilfe „intelligenter“ Verfahren versucht, einen guten Kompromiss zwischen Ergebnisqualität und Rechenzeit zu erzielen.

**Warteliste**

Auf dieser Liste sind alle Knoten, die für das Anheben in Frage kommen. D.h. es sind alle Knoten aufgelistet, die mit hängenden Knoten verbunden sind. Der Startknoten bildet eine Ausnahme, da er ohne eine solche Verbindung bei der Vorbereitung des Dijkstra-Algorithmus schon auf die Warteliste gesetzt wird. Implementiert wird die Warteliste als ein (Long-)Integer-Array.

## Quellen und Beispiele

- http://www.gamasutra.com
  - http://www.gamasutra.com/view/feature/3096/toward_more_realistic_pathfinding.php
  - http://www.gamasutra.com/view/feature/3569/game_ai_the_state_of_the_.php
  - http://www.gamasutra.com/view/feature/3317/smart_move_intelligent_.php

- http://de.Wikipedia.org
  - http://de.wikipedia.org/wiki/Dijkstra-Algorithmus
  - http://de.wikipedia.org/wiki/Bellman-Ford-Algorithmus
  - http://de.wikipedia.org/wiki/Algorithmus_von_Floyd_und_Warshall
  - http://de.wikipedia.org/wiki/A*-Algorithmus
- Bryan Stout hat auf http://www.programmersheaven.com/download/1001/download.aspx ein wirklich super Beispielprogramm mit fast allen oben erwähnten Wegfindungsalgorithmen und auf http://www.gamasutra.com einige interessante Artikel geschrieben.
- http://www.uweschmidt.org/dijkstravis, eine wunderschöne Visualisierung des Dijkstra-Algorithmus in Java, es werden sogar die einzelnen Schritte dargestellt und beschrieben
- http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo7.php, Der 7. Algorithmus der Woche ist wie im „Taschenbuch der Algorithmen“ beschrieben
- „Pathfinding: Search Space Representations“ von Dan Bader
- „3D-Pathfinding“ von Daniel Kastenholz
- „Edsger Wybe Dijkstra (1930-2002): A Portrait of a Genius“ von Krzysztof R. Apt
- „The Scholten/Dijkstra Pebble Game Played Straightliy, Distributedly, Online and Reversed“ von Wolfgang Reisig
- „The State of the Art in Game AI Standardisation“ von Billy Yue und Penny de-Byl
- „Artificial Intelligence – Dijkstra's Algorithm: Notes to Complement and Reinforce the Graduate Student Presentation“ von Marco Valtorta
- „Realistic Pathfinding“ (Beispielprogramm) von Marco Pinter
- „IntensePathfinding_v2.00“ Demo einer KI-Engine