ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Departement Bau,
Umwelt und Geomatik

Vertiefungsblock Wintersemester 2002/03

Integration von GIS-Funktionen in einen interaktiven Wanderroutenplaner

Stefan Spirig
Marcel Wegmann
Hans-Martin Zogg

Institut für Kartographie
Prof. Dr. L. Hurni

Betreuung:
B. Jenny, Dr. A. Terribilini, H. Freimark

Vorwort

Die vorliegende Arbeit entstand im Wintersemester 2002/2003 im Rahmen des Vertiefungsblockes „Kartographie“ im Studiengang Geomatikingenieurwissenschaften der ETH Zürich. Die Idee, einen interaktiven Wanderroutenplaner zu entwickeln, stammt ursprünglich von den Schweizer Wanderwegen (SAW) und wurde vom Institut für Kartographie als Vertiefungsblock ausgeschrieben.

An dieser Stelle möchten wir uns bei unseren Betreuern unter der Leitung von Herrn Prof. Dr. Lorenz Hurni für die ermöglichte Arbeit bedanken. Ein besonders herzliches Dankeschön sprechen wir Bernhard Jenny sowie Dr. Andrea Terribilini aus, die unsere Arbeit hervorragend betreuten und uns bei Programmierproblemen geduldig unterstützten, sowie Helen Freimark, die sich intensiv mit der Höhenberechnung auseinandersetzte. Ebenfalls für die Zusammenarbeit bedanken wir uns bei Thomas Gloor von den Schweizer Wanderwegen, der uns die benötigten Daten bereitstellte.

Der Schlussbericht beschreibt schwergewichtig Vorgehen und Resultate, auf detaillierte Erklärungen der einzelnen Programmteile wird bewusst verzichtet. Der gesamte Quellcode der Applikation und die Javadocs sind auf der zugehörigen CD-ROM vorhanden.

Wir können sagen, dass uns die Entwicklung des Wanderroutenplaners viel Freude bereitet hat. Dank dem grossen Wissen unserer Betreuer konnten wir sehr viel lernen und uns mit interessanten Techniken auseinandersetzen. Eine zusätzlich Motiviation für uns war, dass das vorgegebene Thema aus der Praxis stammt und eine Weiterentwicklung der Applikation durchaus erwünscht ist.

Zürich, 7. Februar 2003

Stefan Spirig

spririgs@student.ethz.ch

Marcel Wegmann

mwegmann@student.ethz.ch

Hans-Martin Zogg

zoggh@student.ethz.ch

Zusammenfassung

Wandern ist eine der beliebtesten Freizeitbeschäftigung in der Schweiz. Es besteht ein sehr dichtes Wanderwegnetz, das eine Vielzahl von Alternativen bietet, um von einem Ort zum anderen zu gelangen. Es existieren ganz bestimmte empfohlene Wanderrouten, die jedoch nicht kombinierbar sind, da sie immer einer kompletten Wanderung entsprechen. Mit der digitalen Erfassung der Wanderwege durch die Schweizer Wanderwege (SAW) lassen sich zukünftig Wanderrouten individuell planen, zum Beispiel über eine Internet-Applikation.

Im Rahmen dieser Arbeit wird ein Prototyp eines Wanderroutenplaners implementiert, der multikriterielle Routenoptimierungsfunktionen in ein benutzerfreundliches, interaktives Informationssystem integriert. Dabei wird berücksichtigt, dass der Wanderer neben der interaktiven Wahl von Start und Ziel auch spezifische Präferenzen angeben kann: Bevorzugung von schöner Aussicht, asphaltierten oder historischen Wegen. Das Resultat ist eine nach seinen Kriterien optimierte Route inklusive Darstellung des Höhenprofils.

Für die Entwicklung des Wanderroutenplaners als interaktive Internet-Applikation, kann auf der „GEOWARN"-Software, die am Institut für Kartographie entstanden ist, aufgebaut werden. Diese Software ist internetbasiert und verwendet eine Client-/Server-Architektur. Die Programmierung zusätzlicher Komponenten erfolgt in Java.

Aufbauend auf den bestehenden Daten der Schweizer Wanderwege, wird eine Datenbank erstellt, welche das Wanderwegnetz als Knoten-/Kanten-Struktur beschreibt. Jeder Kante werden die Attribute „Aussicht", „historischer Wert" und „Belag" zugewiesen. Die Visualisierung der Daten erfolgt im SVG-Format.

Die Berechnung des optimalen Weges vom Start- zum Zielpunkt erfolgt nach dem Algorithmus von Dijkstra. Anstatt einer Länge erhält jede Kante einen Widerstandswert, der aus den Werten und der Gewichtung der Attribute sowie der benötigten Wanderzeit berechnet wird. Die Gewichtung wird dabei durch den Benutzer definiert. Nun wird der in Bezug auf die Widerstandswerte kürzeste Weg berechnet.

Der Wanderroutenplaner besteht aus der graphischen Benutzeroberfläche sowie auf der Seite des Servers aus der Datenbank und den Tools zur Berechnung des Widerstandsfaktors jeder einzelnen Kante, der Netzanalyse mit der Berechnung des optimalen Weges und zur Höhenprofilberechnung.

Die Resultate der Berechnungen sind plausibel und sinnvoll. Im Hinblick auf eine allfällige Weiterentwicklung zeigt sich, dass der Erfassung der Attribute noch grössere Aufmerksamkeit geschenkt werden muss. Gerade die echten Panoramawege werden zu wenig bevorzugt, was an der Definition des Attributes „Aussicht" liegt, welche nur die Waldbedeckung, nicht aber Höhenlage und Exposition berücksichtigt. Die Sensitivität der optimalen Route auf Veränderungen in den präferenzierten Wegeigenschaften ist variabel.

Die Möglichkeit, am Internet Wanderrouten zu planen, wird sich sicherlich realisieren lassen. Um herauszufinden, ob überhaupt ein Bedürfnis danach besteht, könnte der Prototyp bei der Bergstation Stanserhorn auf einem Computer installiert werden, wobei noch einige Ergänzungen notwendig wären.

In erster Linie fehlen noch wichtige Informationen wie öV-Haltestellen und Restaurants in der Datenbank, welche den Informationsgehalt der Karte sowie des Höhenprofils erheblich verbessern würden.

Inhaltsverzeichnis

1. Problemstellung

Das schweizerische Wanderwegnetz, das rund 60'000 km markierte Wege umfasst, ist sehr dicht und bietet dem Wanderer eine Vielzahl von Möglichkeiten, um von einem Ort zum anderen zu gelangen. Aus diesem Grund existieren ganz bestimmte Wanderrouten, die den Touristen durch die ortsansässigen Wanderwegsektionen oder den Kurvereinen empfohlen werden. Es bestehen jedoch keine Hilfsmittel, um sich selber eine Tour zusammenzustellen, da die Routen immer einer kompletten Wanderung entsprechen und nicht kombinierbar sind.

Zur Zeit erfassen die Schweizer Wanderwege (SAW) das gesamte Wanderwegnetz digital. Mit diesen Daten lassen sich zukünftig Wanderrouten individuell planen, wie dies mit Routenplanern für den Strassenverkehr möglich ist. Allerdings geht es bei Wanderrouten nicht unbedingt darum, den schnellsten oder kürzesten Weg zwischen zwei Ortschaften zu berechnen, der Wanderer will vielmehr seine spezifischen Bedürfnisse berücksichtigen, sei das möglichst viel schöne Aussicht zu geniessen, entlang von asphaltierten Wegen zu gehen oder historische Sehenswürdigkeiten zu erreichen.

Ein interaktiver Wanderroutenplaner könnte neben dem optimalen Weg eine Reihe interessanter Zusatzinformation enthalten. Ein eigentlicher Ausflugplaner mit Informationen über die An- und Abreise (Bahn, Bus, Bergbahnen, Parkplätze), Verpflegungsmöglichkeiten (Standorte, Menüplan), Öffnungszeiten von Museen und Ticketpreisen wäre als Maximallösung denkbar.

Im Rahmen des Vertiefungsblocks geht es darum, multikriterielle Routenoptimierungsfunktionen in ein benutzerfreundliches, interaktives Informationssystem zu integrieren. Das System soll über Internet verfügbar und von jedermann benutzbar sein. Spezielle Fachkenntnisse, wie sie Abfragen in geographischen Informationssystemen benötigen, werden nicht vorausgesetzt. Als Testgebiet wird die Region Stanserhorn gewählt, welche sich topographisch sehr gut abgrenzen lässt.

Die zentralen Komponenten, die der Prototyp beinhalten soll, sind:

- interaktive Wahl von Start und Ziel
- Angabe von Präferenzen bezüglich Aussicht, asphaltierten und historischen Wegen
- Darstellung des Höhenprofils

Es wird darauf geachtet, dass ein späterer Ausbau des Systems möglich ist, indem auf bekannten Standards aufgebaut wird.

2. Grundlagen

2.1 Bestehende Wanderrouten

Die offiziellen Wanderrouten sind als Auswahl besonders schöner Wanderwege konzipiert. Es geht weniger darum, alle vorhandenen Wege zu verwalten, als vielmehr dem Wanderer eine vorgegebene Gesamtroute zu empfehlen.

In *Abb. 1* ist das Konzept der Wanderrouten klar zu erkennen. Von der Riederalp führen fünf Routen (341, 252, 265, 267 und 302) auf dem gleichen Wegabschnitt zum Stanserhorn. Es existieren aber auch Wanderwege, die von keiner Route berücksichtigt werden (gestrichelte Linien).

Abb. 1: Aufbau der Wanderrouten. Mehrere Routen führen über die selben Wege.

2.2 Verfügbare Rohdaten

Die Datengrundlage ist eine umfassende Geodatenbank, die von den Schweizer Wanderwegen zur Verfügung gestellt wird. Diese Datenbank kann nicht direkt als Grundlage für den Wanderroutenplaner übernommen werden, da die benötigten Daten nicht in einem einheitlichen Modell gespeichert sind. Ein Teil der vorhandenen Information wird zwar für die Implementierung des Wanderroutenplaners nicht benötigt werden, ist aber für dessen Konzeption interessant.

2.2.1 Knoten- und Kanten-Struktur

Die einzelnen Wegelemente mit den zugehörigen Knoten sind in den beiden Shapefiles „baseline.shp“ und „node.shp“ vorhanden. Die Features besitzen aber lediglich eine ID, die topologischen Informationen sind in der Geodatenbank „GeowalkNW“ zu finden. Die Tabelle „BG1_GAT“ weist jeder Kante einen Ausgangs- und Endpunkt zu (FNODE, TNODE).

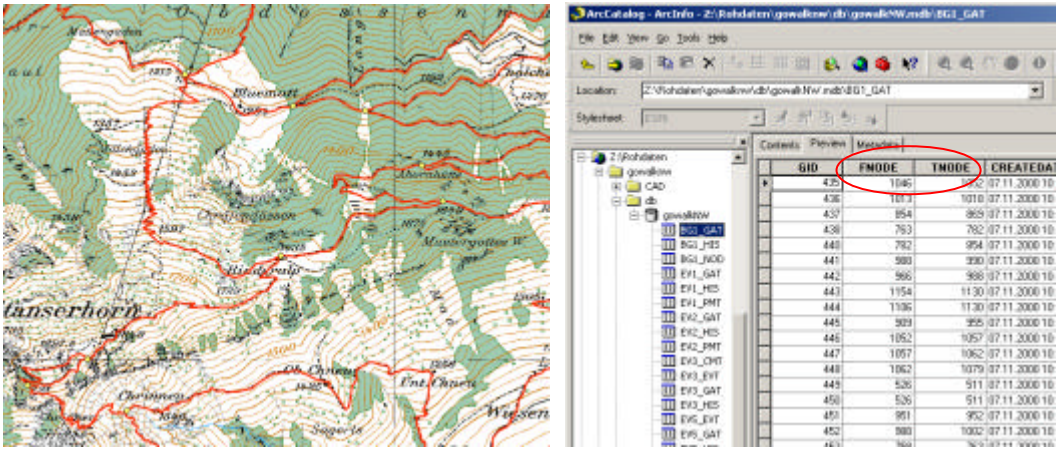
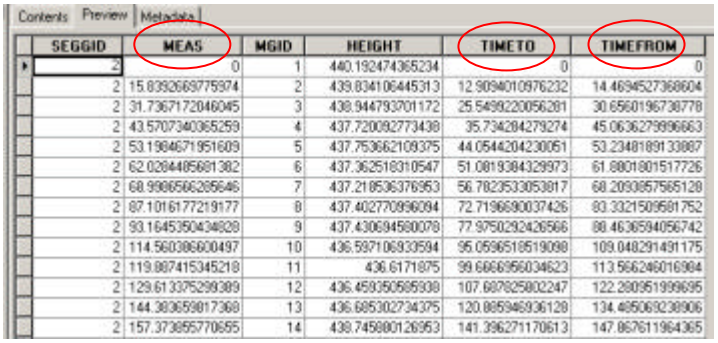


Abb. 2: Baselines und Nodes. Die Zuweisung von Knoten geschieht in der Geodatabase.

2.2.2 Thematische Attribute

Zeitangaben

Zur Berechnung der Wanderzeit existiert eine eigene Tabelle „EV1_PMT“ (Abb.3). Jeder Abschnitt (SEGGID entspricht GID der Kante) wird in kleine Stücke aufgeteilt. Vom Startpunkt aus werden Länge (MEAS [m]) und Wanderzeit in beiden Richtungen (TIMETO, TIMEFROM [m/s]) aggregiert. Benötigt wird je eine Zeit pro Abschnitt und Richtung.



SEGGID	MEAS	MGID	HEIGHT	TIMETO	TIMEFROM
2	0	1	440.192474365234	0	0
2	15.8392669775974	2	439.834106445313	12.9094010976232	14.4694527368604
2	31.7367172046045	3	438.944793701172	25.5499220056281	30.6560196738778
2	43.5707340365259	4	437.720092773438	35.734284279274	45.0636279996663
2	53.1984671951609	5	437.753662109375	44.0544204230051	53.2348189133887
2	62.0284485691382	6	437.362518310547	51.0819384329973	61.8801801517726
2	68.9986566285646	7	437.218536376953	56.7823533053817	68.2939857565128
2	87.1016177219177	8	437.402770996094	72.7196690037426	83.3321509581752
2	93.1645350434828	9	437.430694580078	77.9750292426566	88.4636594056742
2	114.560386600497	10	436.597106933594	95.0596518519098	109.048291491175
2	119.887415345218	11	436.6171875	99.6666956034623	113.566246016984
2	129.613375299389	12	436.459350585938	107.687825802247	122.280951999695
2	144.383659817368	13	436.685302734375	120.885946936128	134.485069238906
2	157.373855770655	14	438.745880126953	141.396271170613	147.867611964365

Abb. 3: EV1_PMT. Die einzelnen Elemente werden zur Berechnung der Wanderzeit stark segmentiert.

Strassenbelag

Die Information über die Art des Strassenbelages ist dem File „belagm.shp“ zu entnehmen. Dabei werden folgende Typen unterschieden:

- Hartbelag_ausserhalb_Siedlung
- Hartbelag_innerhalb_Siedlung
- Naturbelag_ausserhalb_Siedlung
- Naturbelag_innerhalb_Siedlung

Insgesamt ist von 1112 der 1784 Abschnitte die Art des Belags bekannt, rund ums Stanserhorn praktisch zu 100%. Bei fehlender Information wird ein Defaultwert eingesetzt. Es existieren auch Abschnitte, die wiederum in Teilstücke mit verschiedenen Belagarten unterteilt worden sind.

Aussichtsreiche und historische Wege

Im Shapefile „Aussicht_merge.shp“ sind die Attribute „Aussicht“ und „historischer Wert“ (gemäss Inventar historischer Verkehrswege der Schweiz - IVS) erfasst. Dabei werden den Teilstücken folgende Werte zugewiesen:

IVS:	Wanderweg zu ca. 75-100% auf IVS	=	100
	Wanderweg zwischen 25-75% auf IVS	=	50
	Wanderweg zwischen 0-25% auf IVS	=	0
Aussicht:	Wanderweg auf einer Krete	=	100
	Wanderwege (Restmenge)	=	50
	Wanderwege mehrheitlich im Wald	=	30
	Wanderwege vollständig im Wald	=	0

2.2.3 Basiskarte

Die Basiskarte wird so einfach wie möglich gestaltet, da die Informationen im Internet zugänglich sein sollen. Trotzdem werden sehr hohe Ansprüche an Genauigkeit und Qualität gestellt, denn der Benutzer soll sich einerseits einen Überblick über das ganze Gebiet verschaffen können, um Start/Ziel zu wählen, andererseits sind auch detaillierte Auskünfte bezüglich der gewählten Route erwünscht.

Der Kartenausschnitt hat die Koordinateneckpunkte: 660‘000 / 190‘000
674‘000 / 190‘000
674‘000 / 204‘000
660‘000 / 204‘000

Die Basiskarte wird als Rasterbild dargestellt, das Wanderwegnetz wird mit Vektoren aufgebaut.

Im Wanderroutenplaner kann zwischen zwei Basiskarten in unterschiedlichen Massstäben ausgewählt werden. Bei der Karte im Massstab 1:100‘000, welche beim Aufstarten des Wanderroutenplaners erscheint, handelt es sich um eine eingescannte Landeskarte, welche noch nachbearbeitet und JPEG-komprimiert wurde.

Für nähere Ansichten kann auf die Detailkarte 1: 50‘000 umgeschaltet werden. Sie wurde aus dem digitalen Kartenprogramm „Swiss Map 50“ zusammengestellt. Für den benötigten Ausschnitt von 14x14 km müssen insgesamt 18 Screen Shots in Photoshop aneinandergereiht beziehungsweise übereinandergelegt werden. Die Grösse der JPEG-Datei, welche das Bild zu 95% komprimiert, beträgt 951 KB, eine für Internetanwendungen kritische Grösse.

Abb. 4: Basiskarten mit Wegnetz (links 1:100'000, rechts 1:50'000). Der Benutzer kann sich anhand einer gewohnten Darstellung orientieren.

Für eine kommerzielle Internetanwendung müsste man die Datenmenge reduzieren, sei es durch eine Kachelung oder die Verwendung von geeigneten Vektordaten. Sinnvoll wäre die Implementation von adaptivem Zooming.

Die bereits digital vorhandene Pixelkarte 1:25'000 erweist sich als unbrauchbar. Die allzu starke Detaillierung hat sowohl zu grosse Datenmengen als auch schlechte Lesbarkeit im Überblick zur Folge.

2.3 Grobkonzept

2.3.1 Graphische Benutzeroberfläche

Die angestrebte Funktionalität des Wanderroutenplaners wird in einem Grobkonzept konkretisiert. Dabei geht es darum, aufzuzeigen, wie die Benutzeroberfläche in etwa aussehen könnte und welche interaktiven Funktionen implementiert werden. Der Benutzer soll dabei intuitiv merken, wie er zur gewünschten Information kommt.

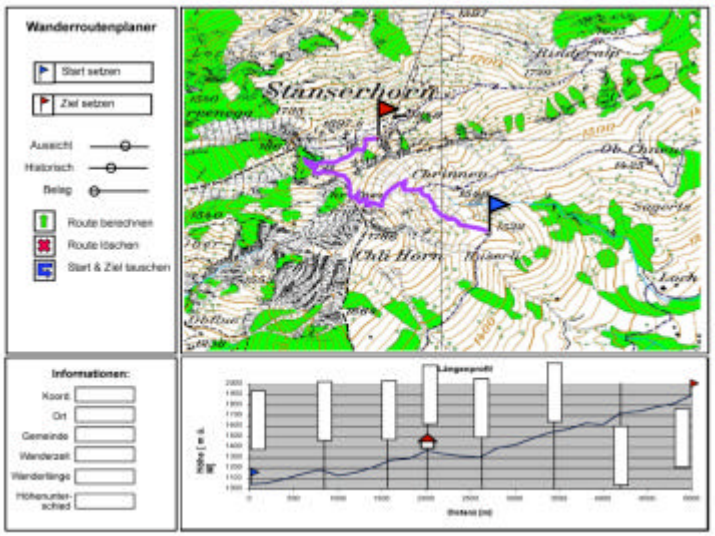


Abb. 5: Grobkonzept. Der Benutzer wählt Start und Ziel sowie die persönlichen Präferenzen.

Das GUI besteht aus vier Fenstern: der Karte oben rechts, den interaktiven Funktionen oben links, sowie den ergänzenden Informationen mit dem Längenprofil. Es sind folgende Interaktionen vorgesehen:

1. Wahl von Start/Ziel: Der Benutzer soll per Mausklick in der Karte den Anfangs- und Endpunkt seiner Tour bestimmen.
2. Eingabe der Präferenzen: Die Berechnung der optimalen Route zwischen A und B erfolgt aufgrund dreier Kriterien: schöne Aussicht, historische Wege und Güte des Belags. Die Gewichtung dieser Kriterien erfolgt durch den Benutzer (0% bis 100%)
3. Abfrage: Mit dem Start der Berechnung werden die eingegebenen Parameter gelesen und aufgrund der Netzeigenschaften der optimale Weg sowie das Höhenprofil berechnet.

2.3.2 Technisches Konzept

Die Architektur des Wanderroutenplaners ist dynamisch, was die Interaktion zwischen Client und Server ermöglicht. Der Client sendet eine Abfrage zum Server, welcher die entsprechende Antwort berechnet und diese zum Client zurückschickt. Der Wanderroutenplaner besteht einerseits aus dem clientseitigen GUI und andererseits auf der Seite des Servers aus der Datenbank und den Berechnungstools:

- Berechnung des Widerstandsfaktors jeder einzelnen Kante
- Analyse des Graphen mit der Berechnung des optimalen Weges
- Höhenprofilberechnung

Jede neue Wanderroute wird vom Server berechnet, da nur das eigentliche Resultat an den Client geschickt wird, die Daten bleiben auf dem Server.

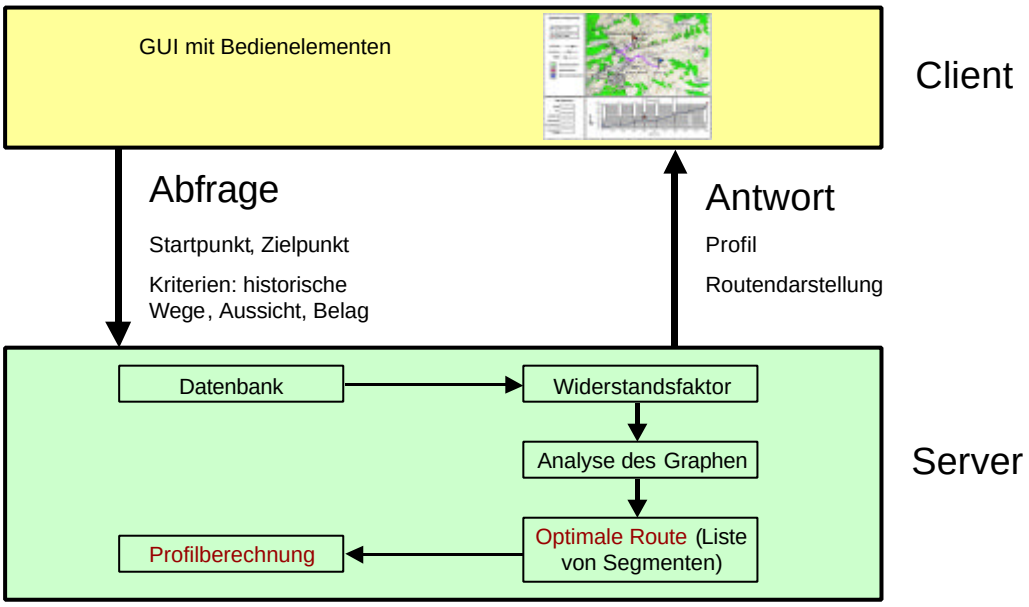


Abb. 6: Dynamische Client-Server Architektur. Der Server schickt nur das eigentliche Resultat an den Client zurück, die Daten bleiben auf dem Server.

2.3.3 Arbeitsaufteilung

Die Arbeiten können in drei unabhängige Teile, welche aufgrund des technischen Konzeptes hervorgehen, aufgeteilt werden. Es sind dies die Bearbeitung des GUI, die Analyse des Graphen zur Berechnung der optimalen Wanderroute und die Profilberechnung. Diese Aufteilung ermöglicht in der ersten Arbeitsphase ein voneinander unabhängiges Arbeiten, wodurch die zur Verfügung stehende Zeit optimal ausgenützt werden kann, da der einzelne nicht unmittelbar abhängig ist von den Arbeiten der anderen. Am Ende der Arbeiten müssen die drei Teile zu einem Ganzen zusammengefügt werden, bevor der Prototyp getestet und die Fehler oder Schwachstellen verbessert werden können.

2.4 Erstellung der Datengrundlagen

2.4.1 Aufbau der Datenbank

Da die vorhandenen Daten aus verschiedenen Quellen stammen, wird eine neue Access-Datenbank erstellt, in der alle benötigten Daten enthalten sind. Der Inhalt und die Herkunft der einzelnen Tabellen sind im folgenden kurz beschrieben:

Name	Quelle	Datenexport	Informationen
arcs	BG1_GAT	ArcCatalog: GeoDatabase to Table	Verknüpfung der Kanten mit Knoten
time_sort	EV1_PMT	ArcCatalog: GeoDatabase to Table	Wanderzeiten
belag_neu	belagm.shp	ArcMap: Export Data	Widerstandswerte des Attributs Belag
attribute	Aussicht_merge.shp	ArcView 3.2: Export Table	Widerstandswerte der Attribute Aussicht und historische Wege
output_3arc	Baseline.shp	ArcToolbox: Shapefile to Coverage	Verknüpfung der GID mit der GeomID
	output3 arc	ArcMap: Export Data	
geometrie_neu	output3 arc	ArcToolbox: Coverage to Generate	Geometrie der Kanten

Die beiden Tabellen „A_1_all“ und „A_geometrie“ enthalten die gesamte Information, einerseits die thematischen Attributen und andererseits die Geometrie. Es handelt sich dabei um Abfragen auf die oben beschriebenen Tabellen.

Auf diese beiden Tabellen wird vom Server zugegriffen: „A_1_all“ enthält die Informationen zur Berechnung der Widerstandswerte, sowie die Topologie des Netzes. Die Geometrie wird zur Berechnung des Höhenprofils benötigt.

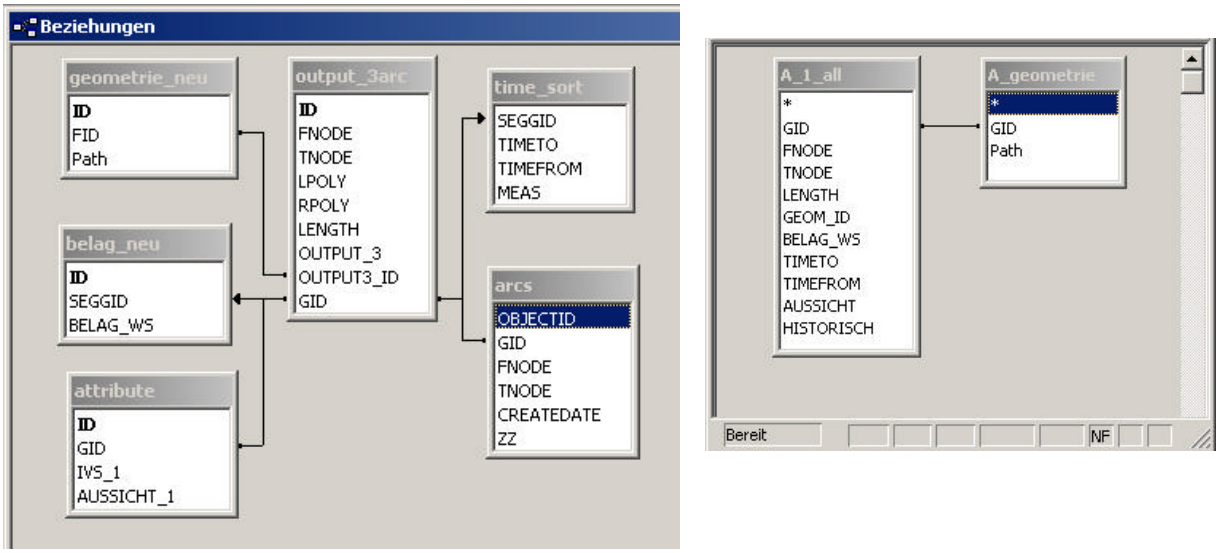


Abb. 7: Grundlagen der Access-DB. Die Attribute werden über die GID, die Geometrie über die GeomID verknüpft.

2.4.2 Wanderrouten als SVG-Elemente

Die Darstellung der Wanderrouten sowie der Knoten erfolgt in SVG, wobei die vorhandenen Daten in Adobe Illustrator importiert und dort als SVG abgespeichert werden. In Illustrator können die sehr genauen Rohdaten (grosse Anzahl an Punkten) noch bereinigt werden. Diejenigen Wege, die sich ausserhalb des Untersuchungsgebietes befinden, werden gelöscht, die übrig bleibenden Linien werden vereinfacht. Mit dem Simplify-Tool kann die Anzahl Punkte drastisch reduziert werden (siehe *Abb. 8*).

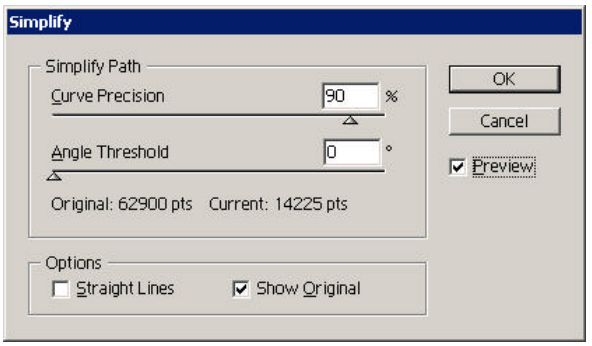


Abb. 8: Simplify in Adobe Illustrator. Mit hoher Präzision können bereits 80% der Punkte reduziert werden.

Da in Illustrator die ID's beim Export verändert werden, ist eine Nachbearbeitung der Daten nötig, was mittels Makros in UltraEdit geschieht (siehe *Anhang 3*).

3. Netzanalyse

3.1 Berechnung des kürzesten Weges

3.1.1 Allgemeine Bemerkungen zu Netzen und Graphen

Ein Netz bestehend aus Kanten und Knoten wird als Graph bezeichnet. Eine Erweiterung von normalen Graphen sind gewichtete Graphen, bei denen die Kanten Gewichte enthalten, welche beispielsweise die für die entsprechende Wegstrecke benötigte Zeit wiedergeben. Ein Graph kann auch gerichtet sein, das heisst, die Gewichtung von Punkt A zu Punkt B unterscheidet sich von der Gewichtung der Kante von Punkt B zu Punkt A, die entsprechende Richtung spielt also eine Rolle. Es ist auch möglich, dass eine Kante nur in eine Richtung benutzt werden kann, was einer Einbahnstrasse entspricht.

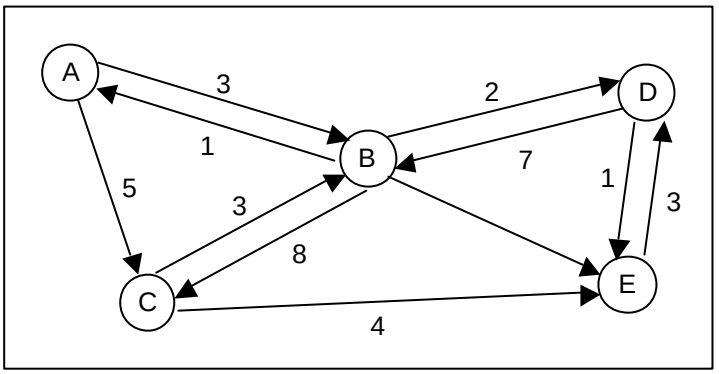


Abb. 9: Gerichteter und gewichteter Graph. Ein Graph besteht aus Knoten und Kanten, die durch unterschiedliche Gewichte spezifiziert werden.

3.1.2 Algorithmus von Dijkstra

Es gibt verschiedene Ansätze, um den kürzesten Weg in einem gewichteten und gerichteten Graphen zu bestimmen. Eine Lösung ist der sogenannte Dijkstra-Algorithmus. Die Grundüberlegung dabei ist, dass die Knoten die Gewichte von den Kanten übernehmen: Jede Kante besitzt ein definiertes Gewicht, das je nach Anfangs- und Zielpunkt den Knoten zugeteilt wird. Der Pfad mit dem kleinsten Gewicht auf dem Zielpunkt entspricht dem optimalen beziehungsweise kürzesten Weg.

Zu Beginn wird der Ausgangsknoten mit dem Gewicht 0 versehen, alle anderen Knoten haben ein noch unbestimmtes Gewicht. Als nächstes erhalten alle Knoten, welche mit dem Ausgangspunkt durch eine Kante verbunden sind, das Gewicht derjenigen Kante. Dieser Vorgang wird nun beim Knoten mit geringstem Gewicht fortgesetzt. Bei Knoten, wo mehrere Pfade zusammentreffen, wird der Pfad mit kleinstem Gewicht ausgewählt. Diese Pfadsuche wird bis zum Endpunkt fortgesetzt: Derjenige Pfad, der im Zielpunkt das kleinste Gewicht aufweist, entspricht der gesuchten Lösung. Bei einer Neuberechnung des kürzesten Weges erhalten wiederum alle Knoten ein unbestimmtes Gewicht.

3.2 Multikriterielle Bewertung

3.2.1 Voraussetzungen der optimalen Wanderroute

Die Berechnung der optimalen Wanderroute wird aufgrund mehrerer Kriterien durchgeführt, welche in den Entscheidungsprozess einbezogen werden. Die Attribute „Aussicht“, „Belag“ und „historischer Wert“ der Wege, welche bei der Wegberechnung berücksichtigt werden müssen, werden zu einem Widerstandswert pro Kante aggregiert.

Das Wanderwegnetz entspricht einem gerichteten und gewichteten Graphen mit Kanten und Knoten, wobei die Kanten den Wanderwegen und die Knoten den Wegverzweigungen entsprechen. Die Gewichtung der Kanten erfolgt aus der Berechnung der Widerstandswerte, einer Funktion der Weglänge und der Attribute. Der gesuchte Pfad zwischen Ausgangs- und Zielpunkt entspricht dem Pfad mit geringstem Gewicht, welcher mit dem Dijkstra-Algorithmus ermittelt wird.

3.2.2 Definition des Widerstandswertes

Der Widerstandswert beschreibt die Gewichtung einer Kante zwischen zwei Knoten. Dieser Wert wird gebildet aus der Distanz bzw. Marschzeit zwischen zwei Knoten und den Eigenschaften „Aussicht“, „historischer Wert“ und „Belag“.

Folgende Formel beschreibt den Widerstandswert für die Kante i in Richtung k:

$$w_{ik} = t_{ik} \cdot (p_A \cdot w_{i,A} + p_H \cdot w_{i,H} + p_B \cdot w_{i,B} + 0.1)$$

w_{ik} = Widerstandswert der Kante i in Richtung k (aggregiert)
t_{ik} = Marschzeit der Kante i in Richtung k
w_{iA} = Widerstandswert des Attributs „Aussicht“ der Kante i [0-1]
w_{iH} = Widerstandswert des Attributs „historischer Wert“ der Kante i [0-1]
w_{iB} = Widerstandswert des Attributs „asphaltierter Belag“ der Kante i [0-1]
$p_{A,H,B}$ = Gewichtung der entsprechenden Attribute (Präferenz der Benutzer) [0-1]

Voraussetzung für die Bildung der aggregierten Widerstandswerte ist die Beschreibung der Attribute einer Kante als Widerstandswerte. Die Eigenschaften eines Attributes werden in Klassen unterteilt, die einem Wert zwischen 0 und 1 entsprechen (0 = sehr gut, 1 = sehr schlecht).

Die Präferenzen des Anwenders für ein oder mehrere Attribute werden ebenfalls in die Berechnung der Widerstandswerte einbezogen. Diese Gewichtung pro Attribut entspricht ebenfalls einem Wert zwischen 0 und 1, der durch den Benutzer verändert werden kann. Der Wert 1 bedeutet, dass das Interesse am entsprechenden Attribut sehr gross ist, wird der Wert 0 angegeben, fliessen die Eigenschaften des Attributs nicht in die Berechnung mit ein.

Die Definition der Widerstandswerte für einzelne Attributklassen beeinflusst die Routenberechnung sehr stark. Es wird davon ausgegangen, dass der Benutzer einen Wegabschnitt A, der seine Präferenzen erfüllt, einem Wegabschnitt B, der die Präferenzen nicht erfüllt, vorzieht, falls der Weg A nicht mehr als doppelt so lang ist als Weg B (Details siehe *Anhang 4*).

4. Programm-Architektur

Der Wanderroutenplaner wird als interaktive Internet-Applikation implementiert. Dabei kann auf einer Software aufgebaut werden, die am Institut für Kartographie für das Projekt „GEOWARN“ entwickelt worden ist. Es handelt sich dabei um eine komplette Eigenentwicklung eines Atlas-Informationssystems.

Die GEOWARN-Software ist internetbasiert und verwendet eine Client-/Server-Architektur. Die einzelnen Tools („Beans“) werden in Java programmiert und der Applikation als Jar-Archive zur Verfügung gestellt.

4.1 Client-/Server-Architektur

Die Interaktion zwischen Client und Server spielt beim Wanderroutenplaner eine wichtige Rolle, da immer wieder neue Routen berechnet werden müssen. Jede Routenberechnung hat eine erneute Abfrage beim Server zur Folge, dem Client werden immer nur die unmittelbar benötigten Daten übermittelt, das heisst die Ausgangsdaten bleiben in der Datenbank gespeichert. Die Datenbank wird nicht verändert. Bei einer entsprechenden Anfrage des Clients wird die Berechnung auf dem Server ausgeführt und die Antwort zurückgeschickt. Diese Client-/Server-Architektur wird als dynamische Architektur bezeichnet.

Der Client, die Web-Komponente, der Application-Server und die Datenbank können miteinander kommunizieren. Befehle werden vom Client ausgelöst und an die Web-Komponente des Servers per HTTP-Protokoll übermittelt. Häufig werden Abfragen in URL verpackt und an den Server geschickt. Je nach Inhalt des URL wird ein bestimmtes Servlet angesteuert, das die Anfrage decodiert und die Parameter an die entsprechende Applikation weiterleitet, welche sich auf dem Application-Server befindet. Die Applikationen führen bestimmte Funktionen aus und senden das Resultat an das Servlet zurück. Dieses erzeugt aus den Resultaten der Applikationen eine für den Client lesbare Antwort und schickt sie diesem zurück.

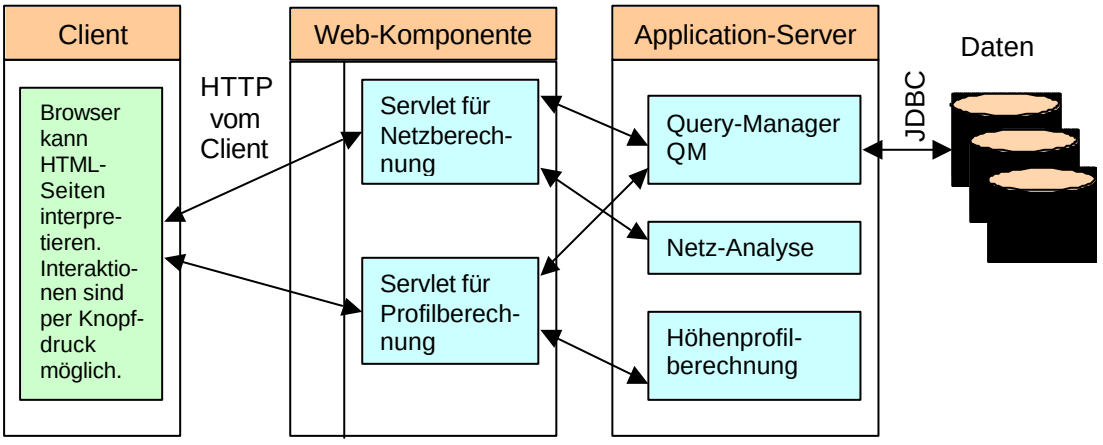


Abb. 10: Dynamische Architektur mit Interaktionen zwischen Client und Server. Die Tools auf dem Application-Server werden von Servlets angesteuert.

Zusammensetzung eines URL:

http://129.132.127.196:8081/servlet/TestServlet?start=5&ende=9&aussicht=0.6&belag=0.3&historisch=0.3

Adresse des Servers	Port-nummer	Beschreibt, welches Servlet angesprochen wird	Parameter, die vom Client an die Applikation übergeben werden (Decodierung durch Servlet)

Beim Wanderroutenplaner werden die Netzanalyse und die Höhenprofilberechnung mit je einem separaten Servlet angesteuert. Der Vorteil von mehreren Servlets liegt darin, dass bei gleichzeitigen Anfragen verschiedener Clients mögliche Datenstaus vermieden werden können. Der Vorgang einer Abfrage muss möglichst schnell geschehen, da sonst Wartezeiten für den Client entstehen.

4.2 Interaktion in der Karte

Die Interaktion im Webbrowser geschieht über die Toolbox auf der linken Seite, in der verschiedene Beans aktiviert werden können. Die Steuerung dieser Beans erfolgt im Bean Manager, welcher in „project.xml“ konfiguriert wird, wo neben den Beans auch Parameter definiert werden, die vom Bean Manager direkt gelesen werden können.

Die Kommunikation des aktiven Beans mit der Kartenoberfläche geschieht über den Map Controller (MC), welcher die JavaScript-Tools (zum Beispiel das Snap-Tool) aktivieren und steuern kann. Der Map Controller wiederum wird in „map.xml“ konfiguriert. In diesem File sind sämtliche Layer definiert. Will nun der Bean Manager auf die Parameter von „map.xml“ zugreifen, muss er dies über den Shared Context tun, da sich der Map Controller ausserhalb des Bean Managers befindet. Wird innerhalb der Karte ein Event (zum Beispiel ein Mausklick) ausgeführt, wird das im Snap-Tool registriert. Das Snap-Tool kann darauf reagieren, das heisst Funktionen, wie zum Beispiel das Setzen der Startflagge, ausführen.

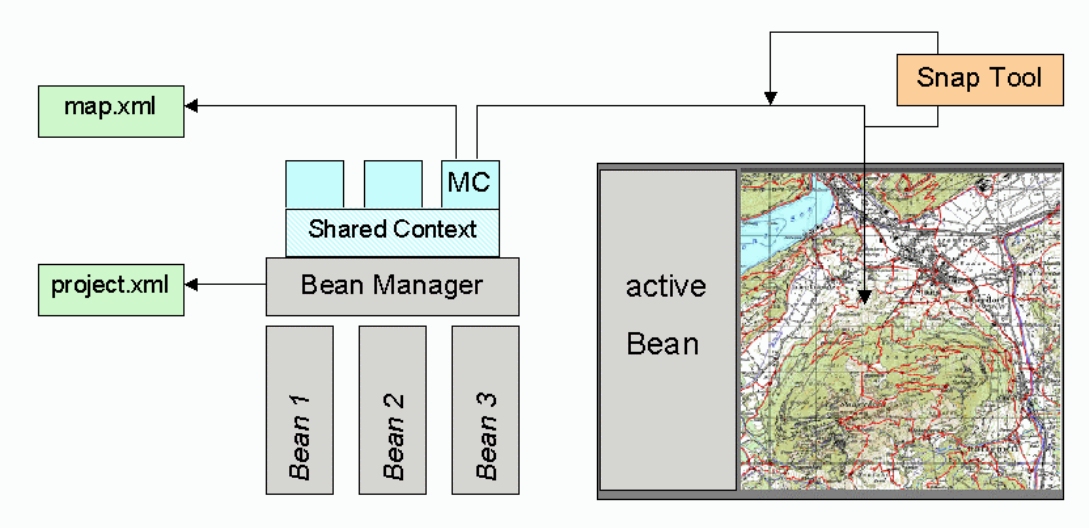


Abb. 11: Aufbau der Client-Applikation. Die Interaktion mit der Karte erfolgt mit JavaScript (in diesem Beispiel das Snap-Tool).

In jedem Fall wird dem Bean zurückgemeldet, welcher Event ausgeführt wurde (Event-Type und Event-Tool). Auf diese Weise können auch im Java-Code aufgrund eines Events Funktionen aufgerufen oder weitere JavaScript-Tools ausgeführt werden. Die Steuerung geschieht dann in der Funktion „mapChanged“ des aktiven Beans.

4.3 Zugriff auf die Datenbank

Bei der Berechnung von Wanderrouten werden Daten aus der Datenbank benötigt (zum Beispiel das Wanderwegnetz mit den Eigenschaften der einzelnen Wegabschnitte). Es wird also eine Datenbankabfrage durchgeführt, was mit Hilfe eines Query-Managers geschieht, der die Ausführung von Datenbankabfragen unterstützt und stark vereinfacht.

Die Abfragen, die aus SQL-Statements bestehen, sind in einem Konfigurationsfile (siehe *Anhang 5*) vordefiniert und werden bei jedem Aufruf ausgeführt. Dabei werden Name und Parameter der Query mit einem URL an den Query-Manager geschickt, welcher den URL entschlüsseln und die gewünschte Datenbankabfrage ausführen kann. Ein grosser Vorteil des Query-Managers besteht darin, dass der Benutzer nicht wissen muss, wo sich die Daten befinden und wie diese aussehen.

Die Verbindung Servlet – Datenbank ist nicht ständig online, das heisst, bei jeder neuen Netzanalyse oder Höhenprofilberechnung wird eine neue Verbindung zwischen dem Servlet und der Datenbank erstellt. Die gewünschten Daten können so ausgetauscht werden. Nach dem Austausch wird die Verbindung wieder geschlossen. Zu beachten gilt, dass das gesamte Wanderwegnetz nur beim ersten Start des Servlets von der Datenbank abgefragt wird. Es wird in einem Zwischenspeicher abgelegt und kann bei einer erneuten Netzanalyse direkt wiederverwendet werden, was die Zeit zur Wegberechnug stark verringert.

Der Query-Manager führt die Datenbankabfrage mittels JDBC aus. Die Antwort aus der Datenbank wird zurück an das Servlet geleitet. Als Resultat einer Datenbankabfrage wird ein XML-File generiert, welches von beliebigen Applikationen wie der Netzanalyse oder der Profilberechnung weiterverarbeitet werden kann. Das Resultat kann auch direkt dem Client zurückgeschickt werden.

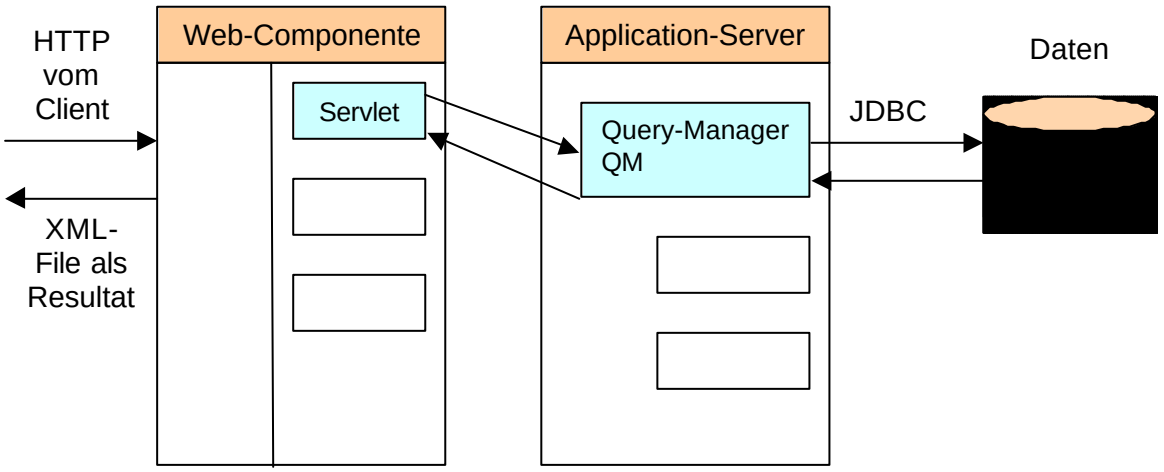


Abb. 12: Zwei-Schichten-Architektur. Der Query-Manager führt die vom Servlet decodierte Benutzerabfrage in der Datenbank aus und schickt die Antwort dem Servlet zurück.

5. Funktionalität des Prototyps

5.1 Aufbau

Der Wanderroutenplaner läuft als Java-Applet auf dem Internet Explorer. Das Browserfenster ist zweigeteilt: rechts die Karte und links die Menüs. Von der zur Verfügung stehenden Funktionalität sind allgemeine Teile wie der „Map Navigator“ oder der „Layer Editor“ vom Geowarn-Projekt übernommen worden. Neu sind die Tools „Wanderroutenplaner“ und „Infos“.

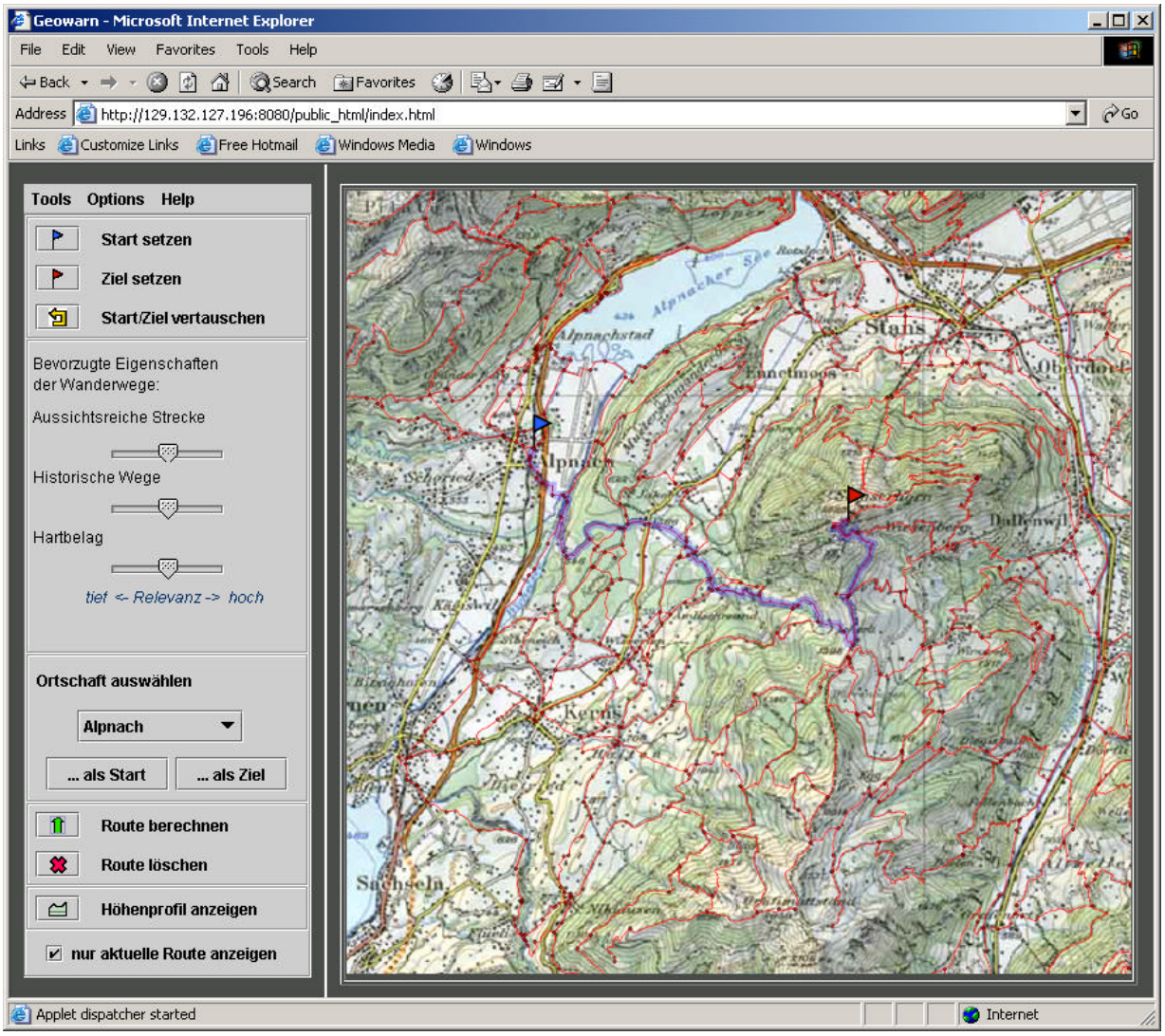


Abb. 13: Browserfenster des interaktiven Wanderroutenplaners. Zentrale Elemente des Programms sind die Tools “Wanderroutenplaner” und “Infos”.

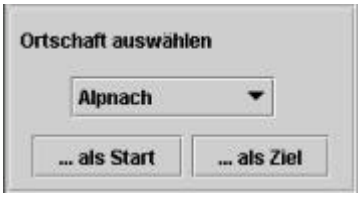
5.2 Wanderroutenplaner

Der Wanderroutenplaner bietet die Möglichkeit, einen Weg von A nach B aufgrund bestimmter Kriterien zu optimieren. Das Tool ist möglichst einfach und selbsterklärend aufgebaut, damit auch Laien damit umgehen können. Trotzdem sind eine Reihe nützlicher Features vorgesehen.

5.2.1 Wahl von Start und Ziel

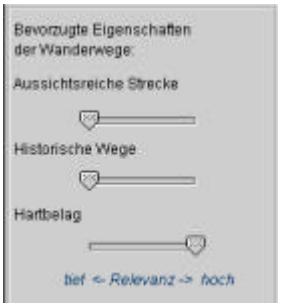
Es stehen zwei verschiedene Möglichkeiten zur Verfügung, um den Start und das Ziel zu setzen: durch Mausklick auf die Karte oder durch Auswahl einer Ortschaft aus einer vorgegebenen Liste. Soll der Start per Mausklick gesetzt werden, muss der Button „Start setzen“ angewählt werden, wodurch die Knoten aktiviert werden und nun ausgewählt werden können. Bei erfolgter Wahl wird der entsprechende Knoten mit der Startflagge markiert.

Zur Auswahl einer Ortschaft steht eine Liste der wichtigsten Ausgangs und Zielpunkte zur Verfügung. Die gewünschte Ortschaft wird in der Liste markiert und anschliessend als Start oder Ziel zugewiesen (Button „… als Start“ resp. „… als Ziel“). Diese Funktion ist vor allem für Benutzer sinnvoll, die sich im Gebiet noch nicht so gut auskennen. Eine weitere nützliche Funktion, die zur Verfügung steht, ist die Möglichkeit, Start und Ziel zu vertauschen (Button „Start/Ziel vertauschen“).



5.2.2 Berechnung von Routen

Sind Start und Ziel definiert, kann die optimale Route berechnet werden (Button „Route berechnen“). Die Optimierung erfolgt anhand der gewählten Präferenzen: aussichtsreiche Strecke, historische Wege und Hartbelag. Für jede Wegeigenschaft kann der Einfluss in die Berechnung angegeben werden (0% bis 100%). Werden alle Eigenschaften als unwichtig angesehen und auf 0% gesetzt, wird der kürzeste Weg berechnet.



Eine berechnete Route wird mit dem Button „Route löschen“ gelöscht. Es ist möglich, dass mehrere Routen miteinander in der Karte angezeigt werden, was der Fall ist, wenn die Option „nur aktuelle Route anzeigen“ ausgeschaltet ist. Ansonsten wird bei einer Neuberechnung die alte Route automatisch gelöscht.

5.2.3 Berechnung des Höhenprofils

Nachdem eine sinnvolle Route berechnet worden ist, besteht die Möglichkeit, das Höhenprofil zu erstellen. Der Button „Höhenprofil anzeigen“ startet die Berechnung und stellt das Höhenprofil in einem neuen Browserfenster dar. Diese Darstellung kann auf ein A4 Querformat gedruckt werden.

Das Höhenprofil wird als SVG-Graphik in einem HTML-File eingebettet. Im HTML-File werden mittels JavaScript die Buttons „Print“ und „Close“ erstellt, welche dem User ermöglichen, das Höhenprofil auszudrucken oder das Browserfenster wieder zu schliessen.

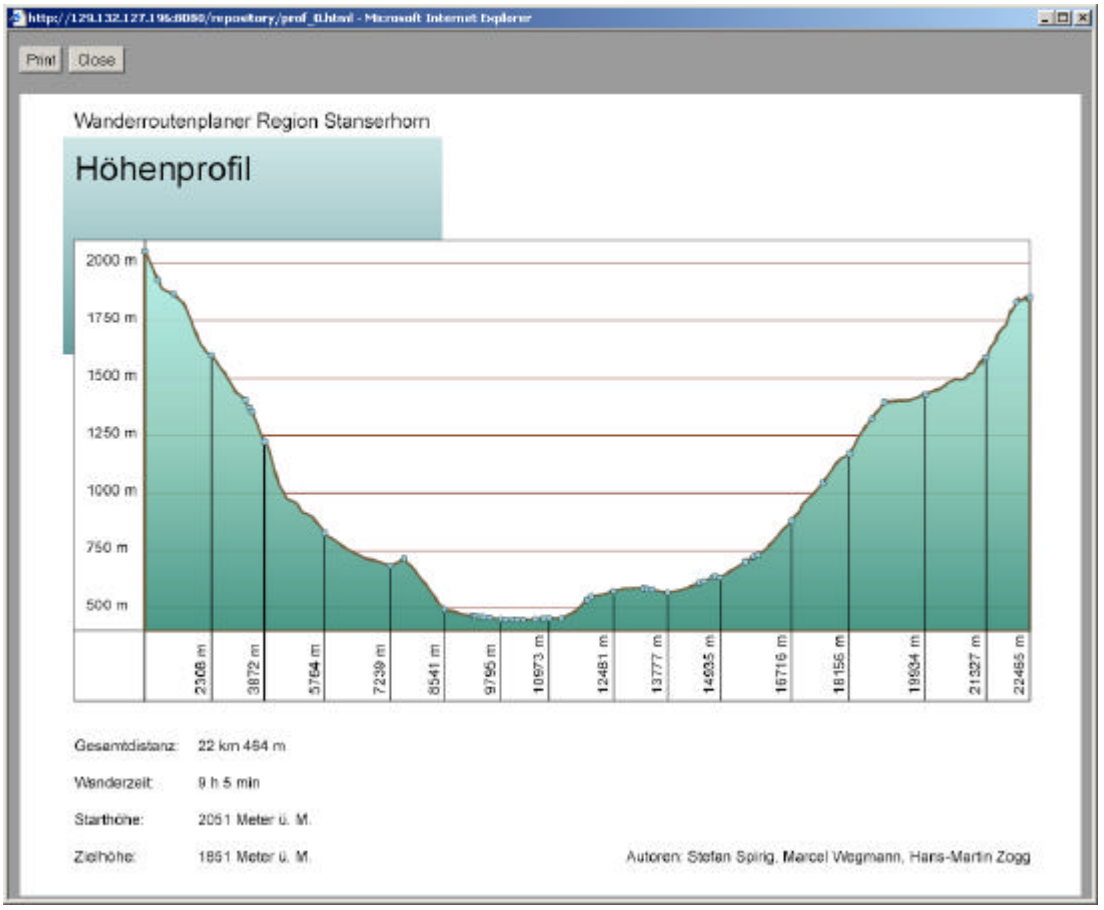


Abb. 14: Höhenprofil. Das Resultat der Höhenprofilberechnung wird als SVG-Graphik (eingebettet in einem HTML-File) dem Client zurückgeschickt.

5.3 Abfrage von zusätzlicher Information

Das Tool „Infos“ ermöglicht eine Orientierung im Gelände, indem Koordinaten, Kanton, Gemeinde Ort und Flurnamen der Knoten angezeigt werden.

Die ganzen Informationen sind dabei Attribute der Knoten und nur im map.svg, jedoch nicht in der Datenbank gespeichert, was sinnvoll wäre. Die Angabe der Koordinaten bezieht sich ebenfalls auf die Knoten bezieht. Besser wäre es, die aktuelle Lage der Maus in Landeskoordinaten anzuzeigen.

5.4 Verifikation der Resultate

Zur Überprüfung der Resultate können die berechneten Routen mit den Wandervorschlägen, die im Internet publiziert sind (www.stanserhorn.ch), verglichen werden. Es zeigt sich, dass der Wanderroutenplaner bei gleichem Start- und Zielort sowie maximaler Präferenz aller Attribute ähnliche Strecken auswählt, die Zeitangaben aber nicht immer übereinstimmen. Die publizierten Wanderzeiten sind stets etwas grösser.

Route	**Aufstieg**		**Abstieg**	
	IST	SOLL	IST	SOLL
Stans - Stanserhorn	4h 17min	4h 30min	2h 45min	3h 30min
Stanserhorn - Wirzweli	2h 47min	3h 30min	2h 10min	2h 30min
Stanserhorn - Gummen	3h 4min	3h 15min	2h 48min	3h
Stanserhorn - Ennetmoos			3h 17min	3h

Abb. 15: Vergleich der berechneten Resultate (IST) mit den gegebenen Wandervorschlägen (SOLL). Die berechneten Routen stimmen mit den Wandervorschlägen überein.

6. Implementation

6.1 Graphische Benutzeroberfläche

Die graphische Benutzeroberfläche baut auf der GEOWARN-Software auf. Die beiden neuen Tools „Wanderroutenplaner“ und „Infos“ sind Java-Beans, welche in einem Jar-Archiv zusammengefasst sind. Die Bestimmung von Start und Ziel, sowie das Anzeigen von Informationen geschieht interaktiv über die Kartenoberfläche. Zur Ausführung der Berechnungen werden Servlets gestartet.

6.1.1 Initialisierung der Beans und allgemeine Methoden

Bei der Initialisierung des Tools „Wanderroutenplaner“ werden die Methoden „init“ und „start“ aufgerufen, wo das Snap-Tool aktiviert wird. Die Aktivierung geschieht über den sharedContext, die Ortschaften der Auswahlliste werden aus dem „project.xml“ im privateContext gelesen (siehe *Kap. 4.2*). Die Kommunikation mit dem Snap-Tool erfolgt über das Objekt „MapProtocol“ des MapControllers.

init*(sharedContext, privateContext)*	
✍ MapController mc	sharedContext.get(„map.controller“)
✍ protocol	mc.getProtocol
✍ Laden der Ortschaften in Auswahlliste	privateContext.get(„ortschaften“)

start*()*	
✍ Aktivierung des Snap-Tools	MapProtocol.ACTIVATE_TOOL
✍ MapListener	mc.addMapListener(this)

Wird im Snap-Tool ein Event registriert und zurückgemeldet (zum Beispiel ein Mausklick), wird aktiven Bean die Funktion „mapChanged“ ausgeführt, wo mit GET_TOOL_DATA die ID des geklickten Features abgefragt und als „start-„ respektive „endId“ gespeichert wird.

mapChanged*(event)*	
✍ Abfrage der Tool Daten	MapProtocol.GET_TOOL_DATA
✍ Speicherung der start- oder endId	

Um weitere Funktionen des Snap-Tools auszuführen, wird die Funktion SET_TOOL_ATTRIBUTE verwendet, welche im Java-Code über die Methode setSnapAttribute aufgerufen werden kann (siehe *Kap. 6.1.3*).

setSnapAttribute*(name, value)*	
✍ Starten von Funktionen des Snap-Tools	MapProtocol.SET_TOOL_ATTRIBUTE

6.1.2 Implementation von Buttons

Der Start-Button aktiviert die Knoten (enableEvents) und meldet dem Snap-Tool, dass die Funktion „Start setzen“ aktiviert ist.

startButton *(resp. EndButton)*	
✍ Aktivierung der Knoten	mc. . „nodes“ .enableEvents(4)
✍ setSnapAttribute(kind, start)	

Der Invert-Button tauscht die IDs von Start und Ziel aus. Im Snap-Tool werden die entsprechenden Flaggen vertauscht.

invertButton
✍ Wechsel von start- und endId ✍ SetSnapAttribute(invert, 1)

Die Funktion „ort2Start“ liest die aktivierte Ortschaft in der Auswahlliste und setzt sie als Startpunkt. Danach wird im Snap-Tool die Flagge beim entsprechenden Punkt gezeichnet.

ort2StartButton *(resp. ort2EndButton)*
✍ Aktivierung der Knoten ✍ setSnapAttribute(kind, end) ✍ setSnapAttribute(setLoc, selectedItem)

Route/Profil berechnen: *calculateRouteButton/calculateProfileButton*

Um die Route oder das Profil zu berechnen, werden Servlets aufgerufen. Dabei wird die Antwort von der Routenberechnung im ByteArrayOutputStream „bufferStream“ zwischengespeichert und der Profilberechnung weitergeschickt.

Das Hervorheben der berechneten Route geschieht über die Funktion setSnapAttribute(draw, geomidStr), welche im Snap-Tool die Funktion drawPath(geomidStr) aufruft.

Soll ein Höhenprofil berechnet werden, muss zuerst der OutputStream der Routenberechnung (das Resultat der Netzanalyse) an das Servlet zurückgeschickt werden. Anschliessend wird das Ergebnis (das Höhenprofil) in einen InputStream gelesen und als HTML-Seite in einem neuen Fenster geöffnet.

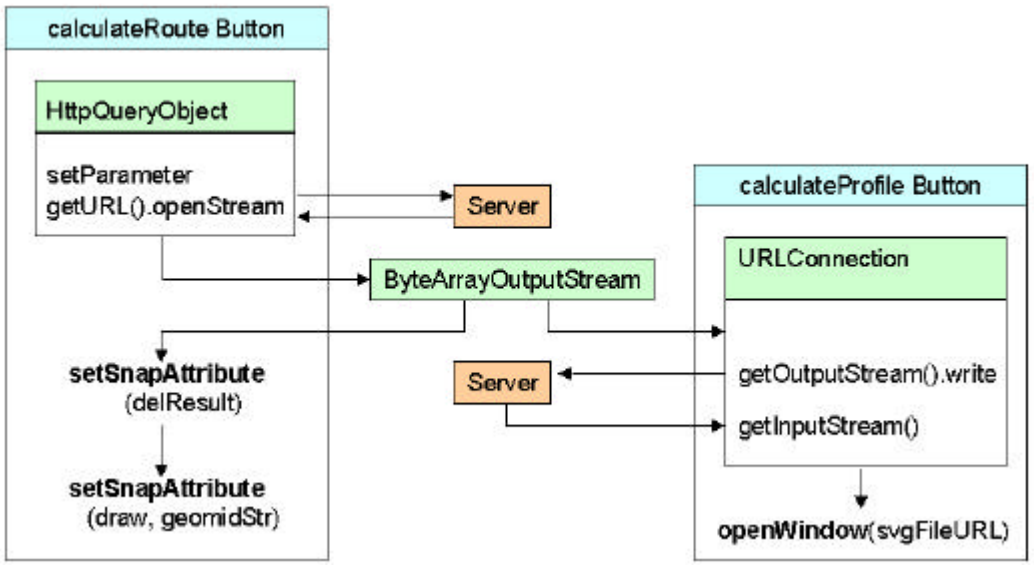


Abb. 16: Aufbau der Buttons calculateRoute und calculateProfile. Das Resultat der Routenberechnung wird im ByteArrayOutputStream gespeichert.

6.1.3 JavaScript Funktionalität

Das Snap-Tool ist ein Programm, das alle Funktionen ausführt, die in irgendeiner Weise auf die Karte zugreifen. Es gibt drei Funktionen, die vom Map Controller aufgerufen werden: getSNAP-Data, um Informationen über einen angewählten Punkt abzufragen, setSNAPData, um Variablen zu setzen sowie den eventHandlerSNAP.

Der eventHandlerSNAP prüft zuerst, welcher Bean aktiv ist. Ist das „Infos“-Tool aktiv, werden die benötigten Informationen aus dem „map.svg“ gelesen und in einen String geschrieben, der dann über getSNAPData abgefragt und dargestellt werden kann. Im Wanderroutenplaner wird der eventHandlerSNAP dazu benötigt, den Start- oder Zielpunkt zu markieren (Funktion drawPoint).

Die Funktion setSNAPData nimmt eine besondere Stellung ein, da hiermit nicht nur Variablen geschrieben werden, sondern gleichzeitig Funktionen innerhalb des Snap-Tools aufgerufen werden können. Die beiden übergebenen Variablen „name“ und „value“ sind eigentlich dazu vorgesehen, der Variablen „name“ den Wert „value“ zuzuweisen. SetSNAPData, welche im Java-Bean mit „setSNAPAttribute“ aufgerufen wird, testet aber, ob die Variable „name“ einem Schlüsselwort entspricht und ruft dementsprechende Funktionen auf.

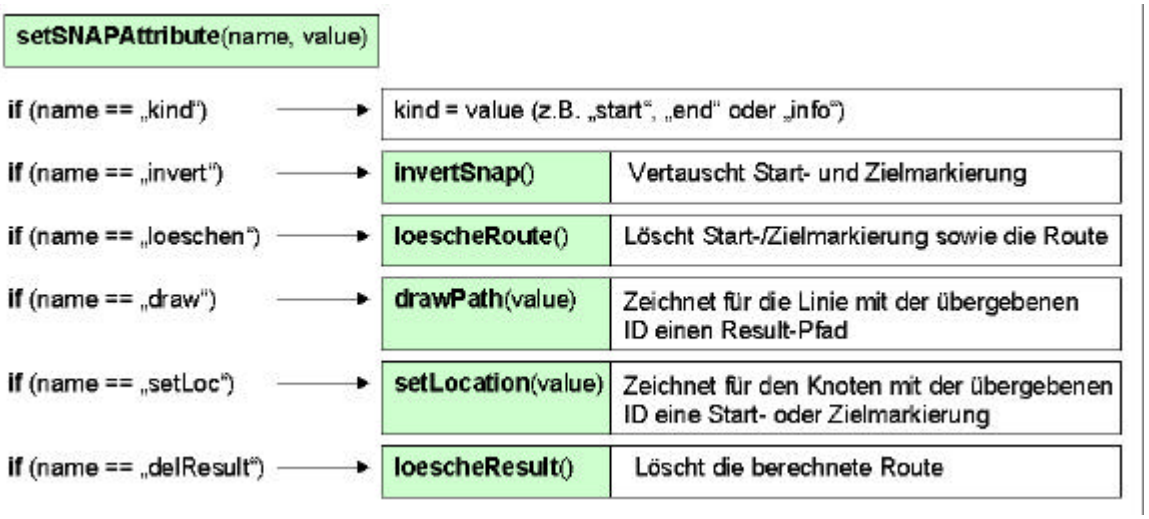


Abb. 17: Aufbau der Funktion setSNAPAttribute. Über die Angabe des Parameters “name” werden die JavaScript-Funktionen aufgerufen.

6.2 Netzanalyse

6.2.1 Überblick über die Programmstruktur

Die Java-Applikation für die Berechnung des optimalen Weges wird über ein Servlet angesteuert. Falls der richtige URL vom Client an die Web-Komponente des Servers übergeben wird, wird der URL vom Servlet decodiert und die gewünschte Applikation (Profilberechnung oder Netzanalyse) ausgeführt. Die folgenden Parameter werden der Applikation zur Wegberechnung übergeben:

- InputStream: inGraph — Das gesamte Wanderwegnetz wird beim ersten Start des Servlets geladen. Die Abfrage wird über den QueryManager zur Datenbank geleitet. Als Antwort wird ein XML-File zurückgeschickt. Dieses bleibt bis zum Abschalten des Servers der Applikation erhalten und kann für jede Neuberechnung des Weges wiederverwendet werden. Es muss also keine erneute Datenbankabfrage gestartet werden, was die Performance stark erhöht.
- PrintWriter: out — In den Printwriter wird das Resultat als XML-File geschrieben. Dieses wird dem Client zurückgeschickt.

? String: origin, destination — Startpunkt und Endpunkt werden als String der Applikation übergeben. Die Werte dieser Strings werden dem vom Client zugeschickten URL entnommen.

? Float: pA, pH, pB — Diese Parameter entsprechen den Benutzerinteressen der Attribute „Aussicht“, „historisch wertvoller Weg“ und „Belag“. Die Werte der Paramter werden ebenfalls dem URL entnommen.

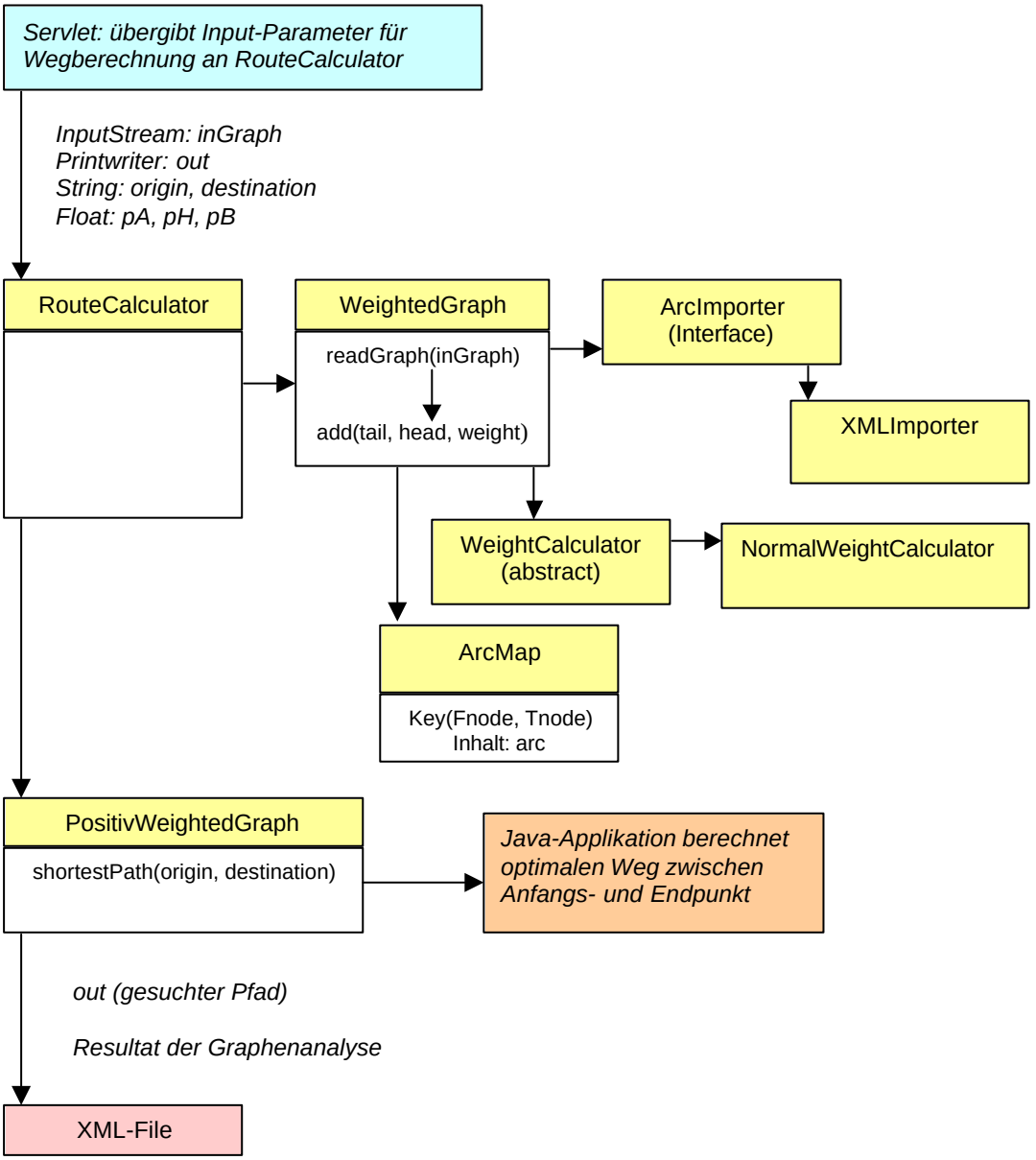


Abb. 18: Ablauf der Netzanalyse. Ein Graph wird eingelesen und der optimale Weg von Start- zu Zielpunkt wird mittels Dijkstra-Algorithmus berechnet.

6.2.2 Elemente der Netzanalyse

Die Applikation zur Netzanalyse gliedert sich in 3 Teile:

- Einlesen des Wanderwegnetzes
- Berechnung der Widerstandswerte
- Berechnung des kürzesten Weges

Der Teil der Wegberechnung - die Umsetzung des Dijkstra-Algorithmus - ist ein Open Source Code, der auf dem Internet zur Verfügung gestellt wird (Siehe [13]). Da es sich dabei um eine Java-Applikation handelt, kann sie ohne grössere Anpassungen übernommen werden. Der Input dieser Applikation besteht aus dem Start, dem Ziel und dem Netz (Wanderwegnetz) als Knoten-/Kanten-Modell.

Das Netz ist so aufgebaut, dass jede Kante mit deren Anfangs- und Endknoten sowie Länge beziehungsweise Widerstandsfaktor beschrieben wird. Jede Kante ist gerichtet und zwar vom ersten zum zweiten Knoten, das heisst, dass der Widerstandsfaktor einer Kante ist nicht für beide Richtungen identisch ist. Es ist also wichtig, dass jede Kante zweimal (für beide Richtungen) definiert wird. Eine Kante vom Punkt 10 zu Punkt 20 mit den Gewichten 50 (von 10 nach 20) und 100 (von 20 nach 10) wird wie folgt eingeben:

10 20 50
20 10 100

Die Beschreibung des Wanderwegnetzes erfolgt im XML-Format, welches als Resultat der Datenbankabfrage generiert wird. Das folgende Beispiel zeigt eine Kante mit Attributen:

```
<arcs>
    <row>
        <col>aaa</col>          GID
        <col>bbb</col>          FNODE
        <col>ccc</col>          TNODE
        <col>ddd</col>          LENGTH
        <col>eee</col>          GEOM_ID
        <col>fff</col>          BELAG
        <col>ggg</col>          TIMETO
        <col>hhh</col>          TIMEFROM
        <col>iii</col>          AUSSICHT
        <col>kkk</col>          HISTORISCH
    </row>
    …
</arcs>
```

6.2.3 Einlesen des Graphen und Berechnung des Widerstandswertes

Die Klasse „WeightedGraph“ enthält die Methode readGraph(), welche das Einlesen eines Graphen sowie die Bildung der Widerstandswerte der einzelnen Kanten aus der Zeit, den Benutzerinteressen je Attribute sowie den Eigenschaften einer Kante ermöglicht. Der XMLImporter liest einen Graphen im XML-Format ein und legt die eingelesenen Paramter in einem ArcObject ab.

Mit dem NormalWeightCalculator werden die Widerstandswerte für jede Kante in beide Richtungen berechnet (getWeigthTo(), getWeightFrom()). Anfangsknoten, Endknoten und Widerstandswert jeder Kante je Richtung werden mit der Methode add() an die Java-Applikation zur Berechnung des kürzesten Weges weitergegeben.

Die Kanten mit allen Attributen werden zusätzlich in einer ArcMap abgelegt (nur 1 Ablage pro Kante). Die ArcMap ist eine HashMap, wo die Kanten mit einem Identifikator (Key) abgelegt werden können. Der Key setzt sich aus dem Anfangs- und Endpunkt zusammen. Mit dem Key kann auf die einzelnen Elemente einer HashMap zugegriffen werden.

Mit Hilfe dieser ArcMap kann nach der Berechnung des kürzesten Weges ausfindig gemacht werden, in welcher Richtung die Kante benutzt wird. Das ist für die Zeitinformation einer Kante besonders wichtig, da die benötigte Zeit in die eine Richtung nicht der Zeit in die andere Richtung entspricht. Die Richtungsinformation geht bei der Berechnung verloren, da nur noch die Knoten und nicht mehr die Kanten als Resultat ausgegeben werden.

Um die Marschrichtung über eine Kante zu bestimmen, muss in der ArcMap überprüft werden, ob der Key (zusammengesetzt aus Anfangs- und Endknoten) vorhanden ist. Falls dies der Fall ist, entspricht die Marschrichtung der ursprünglichen Richtung der Kante, nämlich von FNODE zu TNODE. Die Marschzeit beträgt TIMETO. Ist der Key in der ArcMap nicht auffindbar, so ist die Marschrichtung entgegengesetzt der ursprünglichen Richtung und die Marschzeit entspricht der Zeit TIMEFROM. Die Kante wird vom Endpunkt (TNODE) zum Anfangspunkt (FNODE) durchlaufen.

6.2.4 Resultat-Ausgabe

Als Resultat der Netzanalyse wird ein XML-File mit folgenden Elementen erstellt:

```
<path>
    <arc>
        <col>183</col>          GEOM_ID
        <col>183</col>          GID
        <col>725</col>          LENGTH [Meter]
        <col>653</col>          TIME [Sekunden]
    </arc>
    …
</path>
```

Das Resultat-File wird dem Client zurückgeschickt. Dieser kann es ebenfalls XML-Files lesen und die Wanderwegabschnitte mit der entsprechenden GEOM_ID anders einfärben. So wird dem Benutzer auf der Clientseite das Resultat der Netzanalyse präsentiert.

6.3 Höhenprofil

6.3.1 Programmstruktur

Die Höhenprofilberechnung erhält die Parameter GID, Time und Length aus dem InputStream, welcher vom Client an den Server geschickt wird und dem Pfad aus der Netzanalyse entspricht. Das Servlet steuert sämtliche Programmteile. Zu Beginn des Programms wird das Grid mit den ESRIAsciGridReader eingelesen, dann wird in „ComputeProfile“ mit dem PathReader anhand der GID für jede Kante die Geometrie eingelesen. Diese Datenbankabfrage wird mit Hilfe des QueryReaders durchgeführt. Sämtliche Parameter werden in einer PolyLinie gespeichert. Verschiedene Methoden werden nun auf die PolyLine angewandt: „simplify“ löscht Punkte, „arrange“ ordnet die Liniensegmente, „extractHeight“ führt die Höheninterpolation im Grid durch und „project“ erstellt das Höhenprofil, das in „printSVG“ als Vektorgrafik gezeichnet wird. Zum Schluss wird das Höhenprofil als HTML-File ausgegeben.

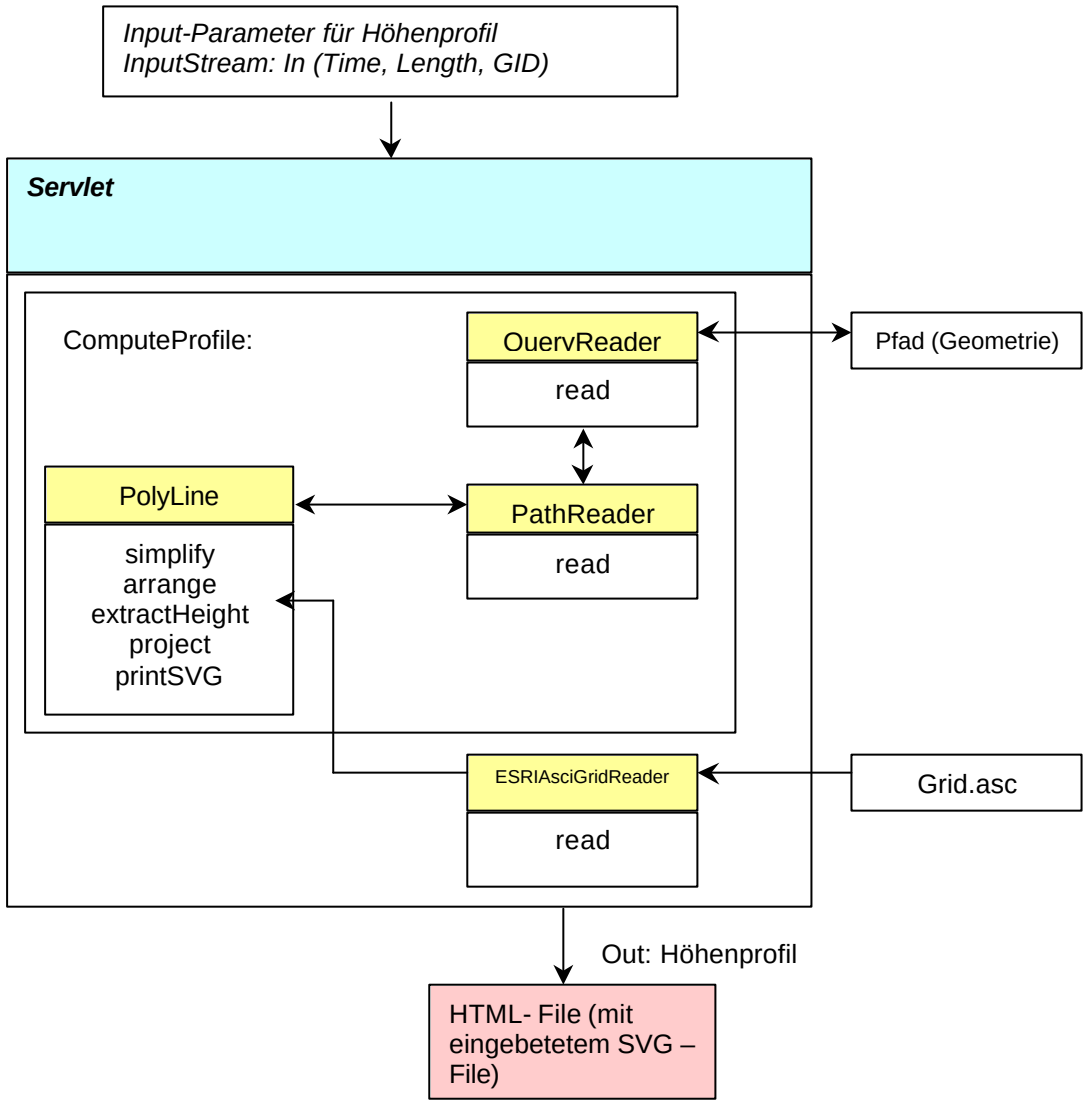


Abb: 19: Programmablauf Höhenprofilberechnung

6.3.2 Einlesen der Daten

Grid

Ausgangspunkt für die Implementierung des Höhenprofils ist ein schon bestehendes Programm zum Einlesen eines Grids (Java-Klasse: ESRIGridReader), sowie einige Methoden mit zusätzlicher Funktionalität, welche am Institut für Kartographie entwickelt worden sind.

Für das Testgebiet Stanserhorn (14 x 14 km) wird ein digitales Höhenmodell (stans.asc) eingelesen. Dieses File enthält einen Ausschnitt aus dem DHM 25 mit den Koordinateneckpunkten:

[658‘000/188‘000; 677‘000/188‘000; 677‘000/206‘000; 658‘000/ 206‘000]

Route

Ein optimaler Weg besteht aus mehreren Arcs. Jeder dieser Arcs enthält verschiedene Attribute:

- die genaue Länge (Length)
- die Wanderzeit (Time)
- die Identifikationsnummer (GID)
- die Geometrie des Pfad (path)

Die Variable „path“ enthält die Landeskoordinaten der einzelnen Stützpunkte eines Arcs. Die Werte für die Attribute Length, Time und GID stehen als InputStream zur Verfügung. Die Geometrie des in der Netzanalyse ermittelten optimalen Weges wird mit den Java-Klassen „QueryReader“ und „PathReader“ über den Server aus der Datenbank herausgelesen.

6.3.3 Bearbeitung der Polyline

Um die Performance der Höhenprofilberechnung zu gewährleisten, muss die Anzahl Punkte pro Arc auf ein Optimum reduziert werden. Der Simplifizierungsalgorithmus „simplify“ wählt jeden n-ten Punkt einer Linie aus, wobei standardmässig jeder 10-te Punkt gewählt wird. Zusätzlich wird sichergestellt, dass der erste sowie der letzte Punkt jedes Arcs beibehalten wird.

Jeder Arc wird als Java-Objekt „Line“ gespeichert, welches die Variablen „Point“, „LineId“, „Time“ und „Length“ enthält. Die Attribute „LineId“, „Time“ und „Length“ werden direkt mit den Werten aus der Datenbank belegt.

Der Pfad „path“, welcher aus der Datenbank eingelesen wird, ist ein String und enthält die Y- und X-Werte der Stützpunkte (in Landeskoordinaten) als Zeichenfolge und muss deshalb zerlegt werden. Mit der Methode hasmoreTokens und nextToken aus der Klasse StringTokenizer wird der String in Tokens zerteilt. Diese werden dann als x- und y-Werte für jeden Punkt eingelesen. Der z-Wert ist zu diesem Zeitpunkt noch leer. Die Punkte werden in eine ArrayList gespeichert.

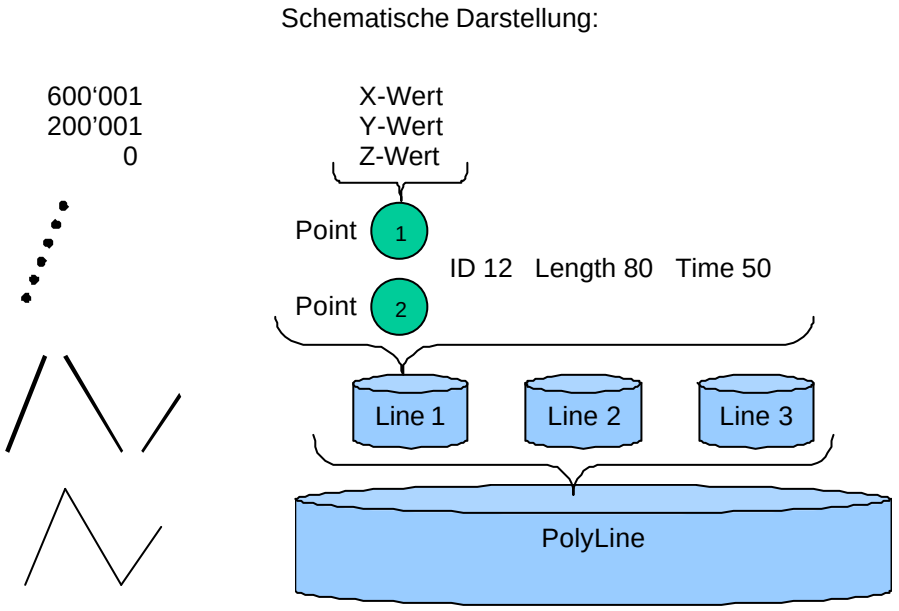


Abb. 20: Objektmodell der PolyLine. Jede Line enthält eine ID, Länge, Zeit und mehrere Punkte. Die Linienpunkte wiederum bestehen aus Landeskoordinaten.

Alle Linien werden nun in eine PolyLinie abgespeichert. Es handelt sich hierbei wiederum um eine ArrayList, das heisst einen Behälter voller Linien.

Beim Einlesen der Linien-ID ist die Richtung der Linie unbekannt. Es ist nicht klar, welches der Anfangs- und welches der Endpunkt einer Linie ist. Die Methode „arrange“ prüft, wie die aufeinanderfolgenden Linien zusammenpassen und kehrt falls nötig die Reihenfolge der Punkte einer Linie, damit die Koordinaten des Endpunkts der Line mit den Koordinaten des Anfangspunkts der folgenden Linie übereinstimmen.

6.3.4 Berechnung des Profils

Höheninterpolation

Die Methode „extractHeight“ weist jedem Linienpunkt (X, Y) die entsprechende Höhe (Z) zu. Zuerst wird berechnet, in welchem Quadranten sich der Linienpunkt befindet. Mit der Funktion

*Value = column + (row * geoRef.nbrX)*

wird auf das Grid zugegriffen.

Nun kann die Höhe des Linienpunktes interpoliert werden. An den Quadranteneckpunkten werden die X-, Y- und Z-Koordinaten aus dem Grid gelesen. Um alle vier Eckpunkthöhen zu verwenden, muss bilinear interpoliert werden.

Nach der Interpolation enthalten alle Linienpunkte X-, Y- und Z-Koordinaten.

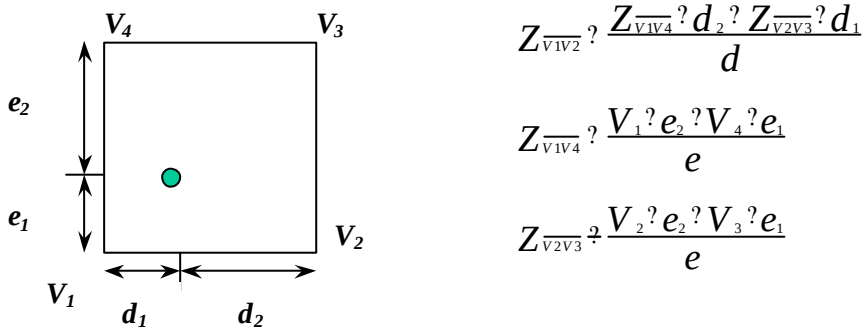


Abb. 21: Funktionsweise, wie das Programm den richtigen Quadranten findet und in diesem mit den vier Quadranteneckpunken eine Interpolation durchführt. So kann die Höhe von jedem beliebigen Punkt im Grid bestimmt werden.

Abwicklung

Um die Linien abwickeln zu können, werden in der Methode „project“ die Distanzen zwischen den einzelnen Linienpunkten berechnet. Dazu wird für alle Punkte eines Arcs aus den X,Y-Koordinaten die Distanz zum Nachbarpunkt bestimmt, welche benötigt werden, um das Höhenprofil graphisch darzustellen (Abstand zwischen den Höhenwerten).

Die im Höhenprofil für die Wanderstrecke und die Wanderlänge angegebenen Zahlenwerte werden nicht aus der Linie berechnet, sondern aus der Datenbank gelesen (Attribute „Length“ und „Time“), damit die im Höhenprofil angegebenen Werte sicher korrekt sind. Die Längenabwicklung wird dadurch ein wenig verzogen, da sie die Distanzen aus den simplifizierten Linien berechnet.

Die einzelnen Distanzen werden nun für jeden Arc aufsummiert. Der erste Linienpunkt erhält dabei den Wert Null. Jeder Punkt einer Linie besteht jetzt aus einem Abstandswert vom ersten Punkt des Arcs (= Nullpunkt) und aus einem entsprechenden Höhenwert des Punktes.

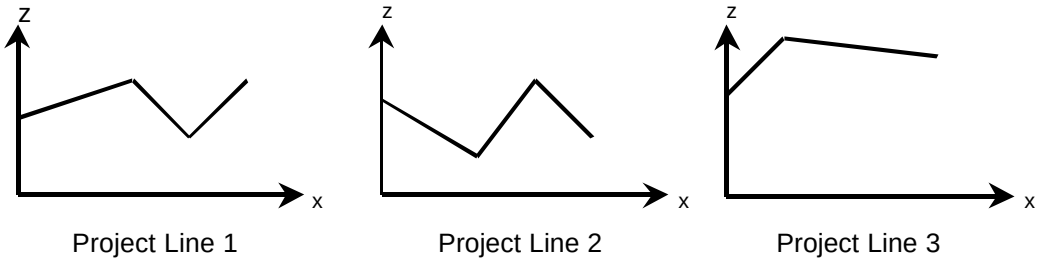


Abb 22: Jede Linie besteht nicht mehr aus X,Y;Z Koordinaten, sondern aus dem Abstand vom ersten Linienpunkt als x-Wert und der entsprechenden Höhe als z-Wert. Grafisch gesehen entspricht jede Linie nun einem Graphen mit dem Abstand vom ersten Linienpunkt als x-Wert und der entsprechenden Höhe als z-Wert.

Die einzelnen Linien der Polylinie, deren Punkte jetzt aus den aufsummierten Distanzen (x) und der Höhe (z) bestehen, werden nun aneinandergereiht. Zum ersten Punkt einer Linie wird der x-Wert des letzten Punktes der vorhergehenden Linie addiert. Sämtliche Punkte werden in ein File gespeichert.

6.3.5 Darstellung des Höhenprofils in SVG

Das Resultat der Höhenprofilberechnung wird als SVG-Graphik dargestellt. SVG ist ein Vektorgraphikformat auf XML-Basis. Diese Graphik wird in ein HTML-Objekt eingebettet und so dem Client zurückgeschickt. Der Client kann das berechnete Höhenprofil in seinem Browserfenster nur betrachten, falls dieser einen SVG-Viewer zur Verfügung hat.

Die Methode printSVG() in der Klasse PolyLine ändert eine SVG-Vorlage so ab, dass das berechnete Höhenprofil optimal in das für das Profil vorgesehene Rechteck passt. Da SVG auf XML aufbaut, ist es möglich, die DOM-Struktur der SVG-Vorlage abzuändern oder zu ergänzen. Auf die einzelnen Elemente eines XML-Dokumentes kann zugegriffen werden. Wichtig dabei ist, dass die zu bearbeitenden Elemente in der SVG-Vorlage mit einer ID versehen werden. So können diese direkt bearbeitet werden. Der Vorteil von Vorlagen liegt darin, dass je nach Benutzerinteresse unterschiedliche Vorlagen mit anderem Graphik-Design für den Client bereitgestellt werden können.

„printDOM()“ ist eine weitere Methode in PolyLine, welche das neugebildete DOM-Dokument in ein File schreibt. Dieses wird in ein HTML-Dokument eingefügt und als Resultat der Höhenprofilberechnung dem Client zurückgeschickt.

7. Schlussfolgerungen

7.1 Erkenntnisse

Der Prototyp des Wanderroutenplaners setzt die Vorstellungen des Grobkonzeptes um, wobei einige Ideen in leicht abgeänderter Form implementiert sind.

Bei der Konfrontation der Software mit einigen Testpersonen hat sich gezeigt, dass der Prototyp benutzerfreundlich und einfach zu bedienen ist. Das Thema „optimieren von Wanderrouten“ scheint dabei echt auf Interesse zu stossen. Für Wanderer ist sicherlich ein Bedürfnis da, neue Routen zu generieren, die von den publizierten Wanderrouten abweichen. Der Wanderroutenplaner ist dazu sehr geeignet.

Die Resultate sind plausibel und sinnvoll, auch wenn einige Problemfelder auftreten:

- Durch die sehr grossen Höhenunterschiede im Testgebiet wird die optimale Verbindung zweier Orte falls möglich nicht über das Stanserhorn, sondern unten im Tal durchgehen. Der „Umweg“ über den Berg ist einfach zu gross und kann nicht durch entsprechende Eigenschaften der Wege kompensiert werden.
- Die Definition des Attributes Aussicht bezieht sich nur auf die Frage, ob ein Weg im Wald liegt oder nicht. Diese Annahme ist im Prinzip nicht richtig, da die Höhe über Meer, sowie die Exposition berücksichtigt werden müsste. Die Bergwege werden mit der jetzigen Definition der Attribute kaum bevorzugt.
- Das Attribut Hartbelag hat nur wenig Einfluss auf die Berechnung, da die asphaltierten Wege meistens sowieso die direkte Verbindung darstellen.

Es ist also ersichtlich, dass die Erfassung der Attributdaten sehr wichtig ist und mit hoher Genauigkeit erfolgen muss. Es ist wahrscheinlich notwendig, noch zusätzliche Attribute zu erfassen, wie zum Beispiel die Höhenlage eines Weges. Auf diese Weise könnten die echten Panoramawege besser einbezogen werden.

Sicherlich muss der Benutzer Zwischenstationen (via) definieren können, was im Moment nicht möglich ist. Eventuell will er sogar die minimale und maximale Wanderzeit angeben. In diesem Fall müsste der ganze Berechnungsalgorithmus noch einmal überarbeitet werden.

Trotz der erwähnten Schwachpunkte sind markante Unterschiede bei der Routenwahl festzustellen, wenn die Präferenzen des Wanderers geändert werden. Ein Beispiel zeigt *Abb. 23*.

Die Berechnung der optimalen Route erfolgt sehr schnell (< 1 Sek.), was die Performance der Applikation beeinträchtigt, ist die relativ speicherintensive Basiskarte. Für eine kommerzielle Anwendung wird es unerlässlich sein, ausschliesslich Vektordaten zu verwenden. Ebenfalls muss ein stärker generalisiertes Wanderwegnetz eingesetzt werden. Das verwendete Netz ist für die Internet-Anwendung zu grossmassstäblich.

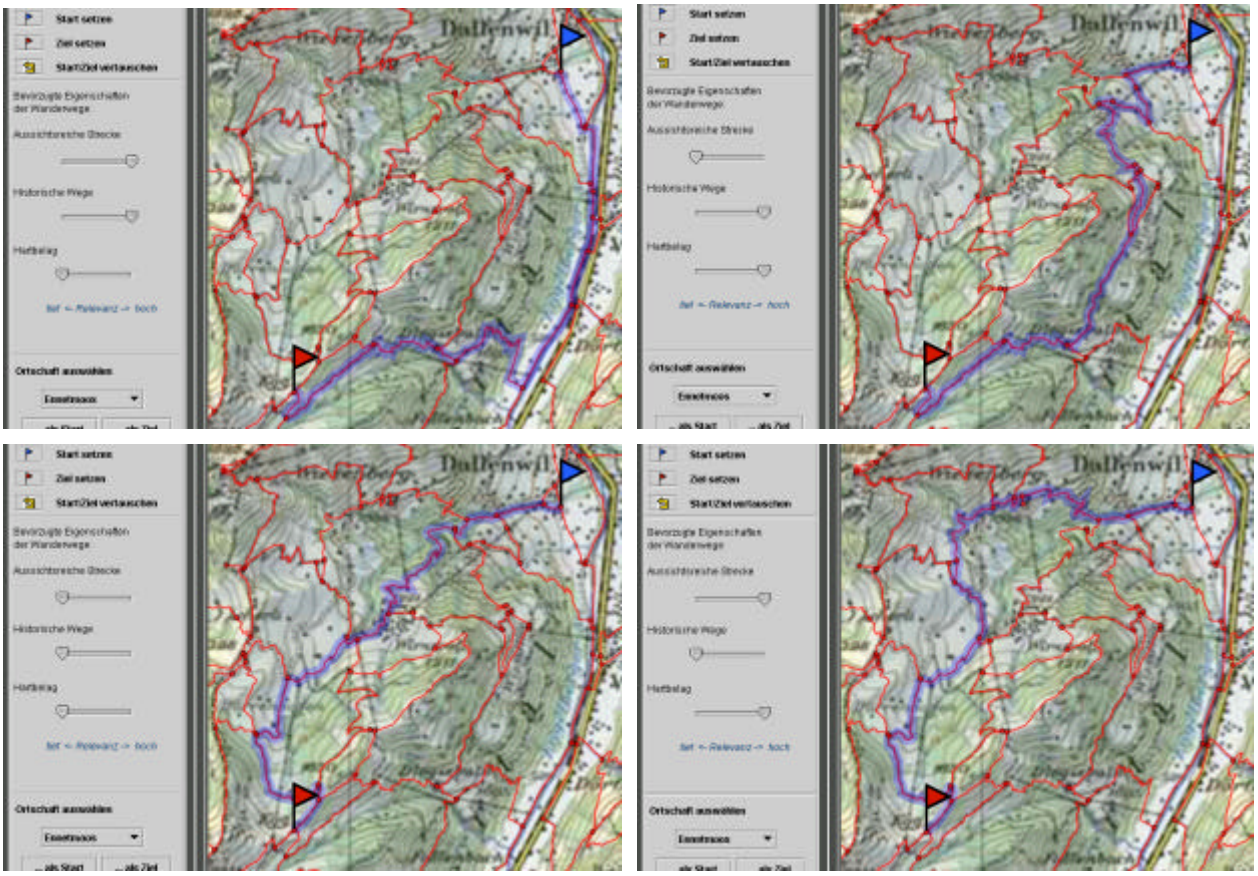


Abb. 23: Beispiel einer Routenberechnung. Bei verschiedener Gewichtung der Eigenschaften werden neue Routen optimiert.

7.2 Ausblick

Die Möglichkeit, über das Internet Wanderrouten zu planen, wird sich sicherlich realisieren lassen, auch wenn die Frage der Datenrechte erst gelöst werden muss. Natürlich muss auch darauf geschaut werden, ob seitens der Wanderer überhaupt ein Bedürfnis nach einem Wanderroutenplaner besteht. Um das herauszufinden, könnte der Prototyp bei der Bergstation Stanserhorn auf einem Computer installiert werden.

Die Datenbank muss noch mit zusätzlich interessanter Information ergänzt werden. Dies können beispielsweise die Namen der Knoten oder die Lage von öV-Haltestellen und Restaurants sein. Sind diese Daten in der Datenbank vorhanden, können sie im Höhenprofil dargestellt werden und so den Informationsgehalt des Profils erheblich erhöhen.

Neben dem Höhenprofil, das bereits in druckbarer Form vorhanden ist, wäre es auch denkbar, die Karte mit der berechneten Route in ein druckbares Format zu bringen. So könnte der Wanderer Plan und Höhenprofil ausdrucken und damit gut ausgerüstet die Wanderung antreten.

7.3 Schwierigkeiten

Im Laufe der Arbeit haben sich einige Probleme ergeben. Es war sicherlich eine Herausforderung für uns, die Arbeit für ein Projekt auf drei Personen zu verteilen. So war denn auch der Koordinationsaufwand nicht unerheblich. Schwierigkeiten bereitete uns neben der Einarbeitung in die technischen Umgebungen vor allem die Aufgabe, die drei unabhängig voneinander programmierten Teile zu einem Ganzen zusammenzusetzen.

Rückblickend wurde das Grobkonzept im Prinzip ohne Kenntnis der technischen Details der Realisierung erstellt. Die Arten möglicher Interaktion und Kommunikation sind zwar theoretisch bekannt gewesen, doch die praktische Umsetzung lässt die Theorie in einem anderen Licht erscheinen.

Mit der Verwendung der GEOWARN-Software konnten wir eine Reihe von Funktionalität übernehmen und mussten sozusagen nicht bei Null beginnen. Neben diesen Vorteilen war diese Wahl aber auch mit Restriktionen verbunden, wie zum Beispiel der Tatsache, dass adaptives Zooming (noch) nicht möglich ist.

Die verwendeten Standards wie Java, XML und SVG schienen uns am Anfang etwas kompliziert, doch ein allfälligen Ausbau des Tools ist nun sehr einfach zu bewerkstelligen und die einzelnen Klassen können auch in anderen Projekten verwendet werden.

Die eigentliche Testphase war zu kurz. Zwar konnten die einzelnen Teile getestet werden, doch dann verging noch einige Zeit, bis alle Komponenten zusammen funktionierten. Vor allem die Intuitivität des User Interface kann erst durch Aussenstehende ohne Vorwissen getestet werden. Hier wurde ein Mangel entdeckt ohne dass wir noch Zeit hatten, ihn zu beheben: Der Nutzer kann auf zwei völlig unterschiedliche Weisen den Start- und Zielknoten bestimmen, was ohne vorherige Instruktionen zu Verwirrungen führt.

Abkürzungen

GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
JDBC	Java DateBase Connectivity
SQL	Structured Query Language
JPEG	Joint Photografic Expert Group
XML	Extensible Markup Language
DOM	Document Object Model
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator
HTML	HyperText Markup Language
ÖV	öffentlicher Verkehr
IVS	Inventar historischer Verkehrswege der Schweiz

Literaturverzeichnis

[1] D. Flananan (2001): *JAVA in a nutshell*. Verlag O'Reilly, Köln.

[2] D. Bell, M. Parr (2002): *JAVA for students*. Pearson Education, Harlow.

[3] D. Louis, P. Müller (2000*)*: *Jetzt lerne ich Java – Der einfache Einstieg in die Internet-programmierung.* Markt und Technik Verlag, München.

[4] D. J. Koosis, D. Koosis (1999): *Java 2 für Dummies.*, MITP-Verlag, Bonn.

[5] S. Schneider (2002): *HTML und JavaScript zum Nachschlagen*. SYBEX-Verlag, Düsseldorf.

[6] R. Sedgewick (1999): *Algorithmen in C++*. Addison-Wesley Publishing Company, Inc.

[7] H. Spona (2001): *SVG Webgraphiken mit XML*. Verlag Moderne Industrie Buch AG & Co. KG, Landsberg.

[8] E. R. Harold, W. S. Means (2001): *XML in a nutshell*. O'Reilly & Ass. Inc., Sebastopol.

[9] A. Carosio (1999): *Geoinformationssysteme Band 1*. Institut für Geodäsie und Photogrammetrie, ETH Zürich.

[10] H. Schauer(1998): *Einführung in die Programmierung, Vorlesungsskript*. Institut für Informatik, Uni Zürich.

[11] Institut für Verteilte Systeme, Universität Magdeburg: Einführung, Algorithmen und Datenstrukturen (Feb. 03).
`http://ivs.cs.uni-magdeburg.de/~dumke/EAD/Skript41.html`

[12] Electrical und Electronic Engineering, University of Western Australia: Dijkstra Algorithmus (Feb. 03).
`http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dijkstra.html`

[13] P. M. Williams, University of Sussex: „Shortest Path" Java-Applikation (Feb. 03).
`http://www.cogs.susx.ac.uk/local/teach/dats/notes/html/`

[14] Java-Dokumentation (Feb. 03).
`http://java.sun.com/products/servlet/2.3/javadoc/index.html`

[15] The Java Developpers Almanac 1.4 (Feb. 03).
`http://javaalmanac.com/`

[16] Rondorama Stanserhorn, Luzern (Feb. 03).
`http://www.stanserhorn.ch`

[17] Schweizer Wanderwege (Feb. 03).
`http://www.swisshiking.ch`

[18] W. Schickard, Institut für Informatik, Universität Tübingen: Interpolationsmethoden (Feb. 03).
`http://www.gris.uni-tuebingen.de/projects/vis/coursebook/visualization/classification/enrichment`

[19] OLYMPUS, Digital Microscope : Interaktives Java Tutorial (Feb. 03).
`http://www.mic-d.com/java/digitalimaging/sizepanel/`

[20] Inventar historischer Verkehrswege der Schweiz (Feb. 03).
`http://www.ivs.unibe.ch/deutsch/inventar.htm`

[21] Landeskarte der Schweiz – Offizielle Wanderkarte der SAW, 1: 50'000, 245 T, STANS, Bundesamt für Landestopographie, Ausgabe 1997

Anhang 1: Inhalt der CD-ROM

Directory	Inhalt/Beschreibung
Berichte	- Schlussbericht VTB - Schlusspräsentation - Poster
Code/GUI	Java-Klassen des GUI: Implementierung der Beans „Wanderroutenplaner“ und „Infos“ (route.jar)
Code/Höhenprofilberechnung	Java-Klassen zur Höhenprofilberechnung
Code/Netzberechnung	- Netzberechnung_mitServlet (implementiert im Wanderroutenplaner) - Netzberechnung_StandAlone_V1 (vom Wanderroutenplaner unabhängige Version für eine Netzanalyse durchzuführen)
Code/ProfileServlet	Interaktiver Wanderroutenplaner: - Config (Konfigurationsfile des Query-Managers) - public_html (Steuerung Client) - repository (Resultat der Höhenprofilberechnung) - resources (Digitale Terrainmodelle, Vorlagen für Höhenprofil) - temp - WEB-INF (Steuerung der Servlets)
Code/Servlets	- ProfileServlets.java - RouteServlets.java
DB_QM	Query-Manager mit der gesamten Datenbank (Testdatensatz Stanserhorn): - DB_NW_neu.mdb (Database) - config.xml - QueryManagerUI.jar
Diverses	- Netbeans 3.4.1 (Version für Windows) - Java JRE 1.4.0_03 - Rohdaten (von SAW)
JAVA_Dokumentation	Internetauftritt (index.html) mit Links zu den Java-Dokumentationen, den schematischen Darstellungen (svg-Graphiken) der einzelnen Programmelemente und einer Hilfe für die Bedienung des interaktiven Wanderroutenplaners
Widerstandswerte	Tests der Berechnungsformel des Widerstands-wertes

Anhang 2: Installationsvoraussetzungen

User:

- Browser (MS InternetExplorer 5.5 oder höher)
- PlugIn svg-Viewer (Adobe)
- Java 2
- JRE 1.4.0 oder höher

Server:

- JRE 1.4.0 oder höher
- Servlet Engine für Servlet 2.4 (z.B. Tomcat 4.1)

Anhang 3: Pfad-Bearbeitung

Die Pfade, die als Shapefiles vorhanden sind, werden als Coverage gespeichert und anschliessend als Generate-File exportiert. In diesem Format werden die Feature-IDs mit zugehörigen Landeskoordinaten in ein Text-File geschrieben. Nach einer Nachbearbeitung können die Koordinaten in die Datenbank übernommen werden.

Um die Vektordaten darzustellen, müssen sie als SVG gespeichert werden, wozu ein Import der Shapefiles in Adobe Illustrator nötig ist. Nun können die Daten als SVG-File abgespeichert werden, wobei in Illustrator automatisch ein Bildkoordinatensystem definiert wird.

Die IDs der SVG-Elemente werden beim Export aus Illustrator falsch gespeichert, was in Ultra Edit korrigiert wird:

```
<path id="_x0031_05" ...
      ...                    (unterschiedliche Anzahl Zeilen pro Pfad)
<path id="_x0031_06" ...
```

SVG-Element nach Bearbeitung in UltraEdit:

```
<path id="105" ...
      ...                    (unterschiedliche Anzahl Zeilen pro Pfad)
<path id="106" ...
```

Makro „Path“:

```
Insert Mode
ColumnModeOff
HexOff
UnixReOff
Key RIGHT ARROW
        …
Key DEL
        …
Key RIGHT ARROW
Key DEL
Key HOME
Key DOWN ARROW
Loop 20
IfCharIs "<"
ExitLoop
Else
Key DOWN ARROW
EndLoop
```

Nachdem die Parameter „fill“ und „stroke“ gelöscht worden sind (die Darstellung wird in „map.xml“ definiert) können die Elemente in das Konfigurationsfile „map.svg“ kopiert werden.

Die Referenzierung der Bildkoordinaten geschieht in „project.xml“.

Anhang 4: Empirische Überprüfung des Berechnungsalgorithmus

Folgende Beispiele verdeutlichen, wie der Widerstandswert eines Attributes einer Kante festgelegt wurde. Es wird darauf geachtet, dass der Punkt, bei dem noch ein doppelt so langer Weg in Kauf genommen wird, bei ungefähr einem Interessenswert von 0.5 des entsprechenden Attributes liegt. Der Interessenswert wird von Benutzer definiert.

Die Eigenschaft der Attribute (schlecht, mittel, gut) wird durch folgende Zahlenwerte beschrieben:

Schlecht ? 0.73

Mittel ? 0.5

Gut ? 0.23

Diese Zahlenwerte erfüllen die oben beschriebene Eigenschaft am besten (Siehe untenstehende Beispiele).

Ein Weg, der bei voller Gewichtung (1) eines Attributes, diese Eigenschaft erfüllt, wird einem Weg mit halber Länge, der diese Eigenschaft nicht erfüllt, vorgezogen.

Weglänge [Einheiten]	Aussicht	Historisch	Belag	w
2	0.27	0.5	0.5	0.74
1.5	0.5	0.5	0.27	0.90
1	0.73	0.5	0.5	0.83

Verhältnis Weglänge2/Weglänge1 **0.8915663**

>> doppelte Weglänge wird in Kauf genommen

w = Widerstandswert

Eigenschaften		
schlecht	mittel	gut
0.73	0.5	0.27

Gewichtung eingeben:	
Aussicht:	1
Historischer Weg:	0
Belag:	0

1 sehr wichtig
0 gar nicht wichtig

Eine Kante, die die gewünschte Eigenschaft bei voller Gewichtung (1) erfüllt, kann maximal 2.244 mal länger sein als eine Kante, die diese Eigenschaft nicht erfüllt.

Weglänge [Einheiten]	Aussicht	Historisch	Belag	w
2.244	0.27	0.5	0.5	0.83
1.5	0.5	0.5	0.27	0.90
1	0.73	0.5	0.5	0.83

Verhältnis Weglänge2/Weglänge1 **1.0003373**

>> doppelte Weglänge wird in Kauf genommen

w = Widerstandswert

Eigenschaften		
schlecht	mittel	gut
0.73	0.5	0.27

Gewichtung eingeben:	
Aussicht:	1
Historischer Weg:	0
Belag:	0

1 sehr wichtig
0 gar nicht wichtig

Andererseits kann aber auch die Gewichtung variiert werden. Das heisst, das Interesse des Benutzers wird angepasst. In diesem Fall kann das Interesse bis ca. 0.53 sinken, so dass doch noch immer die Kante mit doppelter Länge vorgezogen wird.

Weglänge [Einheiten]	Aussicht	Historisch	Belag	w
2	0.27	0.5	5	0.49
1.5	0.5	0.5	0.27	0.55
1	0.73	0.5	0.5	0.49

Verhältnis Weglänge2/Weglänge1 **0.9985623**

>> doppelte Weglänge wird gerade noch in Kauf genommen

w = Widerstandswert

Eigenschaften		
schlecht	mittel	gut
0.73	0.5	0.27

Gewichtung eingeben:	
Aussicht:	0.53
Historischer Weg:	0
Belag:	0

1 sehr wichtig
0 gar nicht wichtig

Die geringe Gewichtung des Interessens des Benutzer für die Aussicht (0.2), zeigt deutlich, dass in solch einem Fall die effektiv kürzere Kante gewählt wird.

Weglänge [Einheiten]	Aussicht	Historisch	Belag	w
2	0.27	0.5	0.27	0.31
1.5	0.5	0.5	0.27	0.30
1	0.73	0.5	0.5	0.25

Verhältnis Weglänge2/Weglänge1 **1.2520325**

>> kurze Weglänge wird gewählt

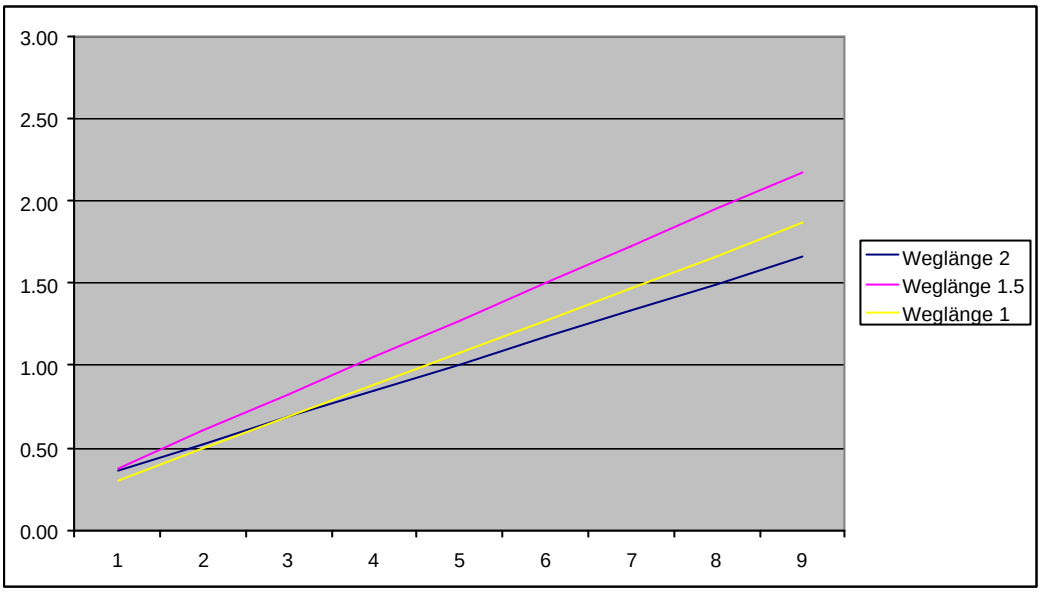
w = Widerstandswert

Eigenschaften		
schlecht	mittel	gut
0.73	0.5	0.27

Gewichtung eingeben:	
Aussicht:	0.2
Historischer Weg:	0
Belag:	0

1 sehr wichtig
0 gar nicht wichtig

Folgende Darstellungen zeigen den Verlauf der Änderung des Widerstandwertes bei unterschiedlicher Gewichtung der Attribute, wobei die verschiedenen Attribute jeweils die selbe Gewichtung erhalten. In den verschiedenen Darstellungen ändern sich jeweils die Eigenschaften der Kanten.



	Aussicht	Historisch	Belag
2	0.27	0.27	0.27
1.5	0.50	0.50	0.50
1	0.73	0.73	0.50

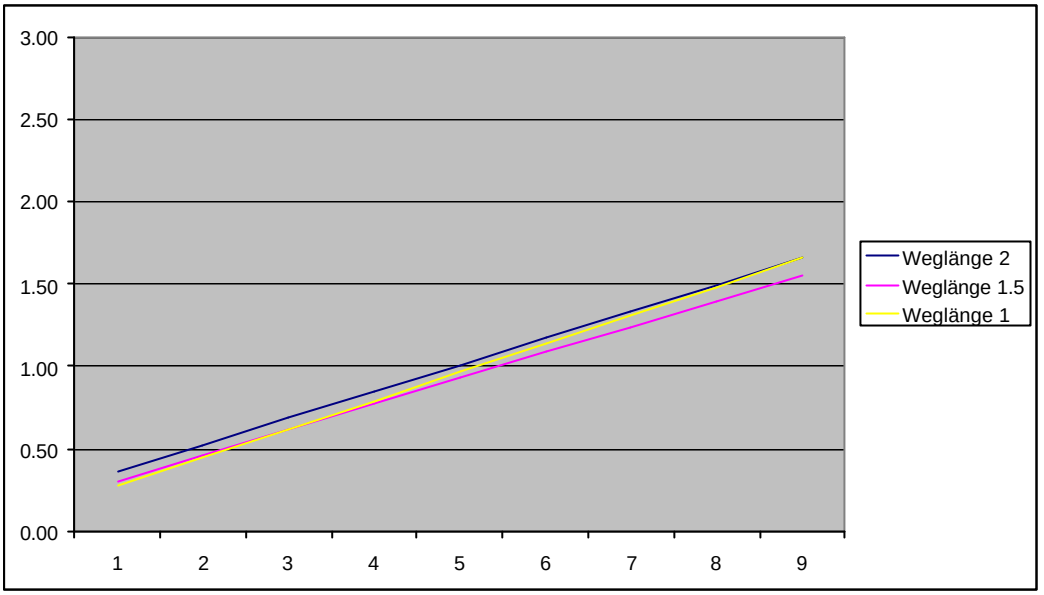
Eigenschaften		
schlecht	mittel	gut
0.73	0.50	0.27

Gewichtung eingeben:	
Aussicht:	0-1
Historischer Weg:	0-1
Belag:	0-1

1 sehr wichtig
0 gar nicht wichtig

Berechnung des Widerstandswertes Aufgrund der Weglänge und der Gewichtung (Gewichtung bei den Attributen jeweils die selbe)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	Gewicht
2	0.36	0.52	0.69	0.85	1.01	1.17	1.33	1.50	1.66	
1.5	0.38	0.60	0.83	1.05	1.28	1.50	1.73	1.95	2.18	
1	0.30	0.49	0.69	0.88	1.08	1.28	1.47	1.67	1.86	
Weglänge										



	Aussicht	Historisch	Belag
2	0.27	0.27	0.27
1.5	0.27	0.27	0.50
1	0.73	0.50	0.50

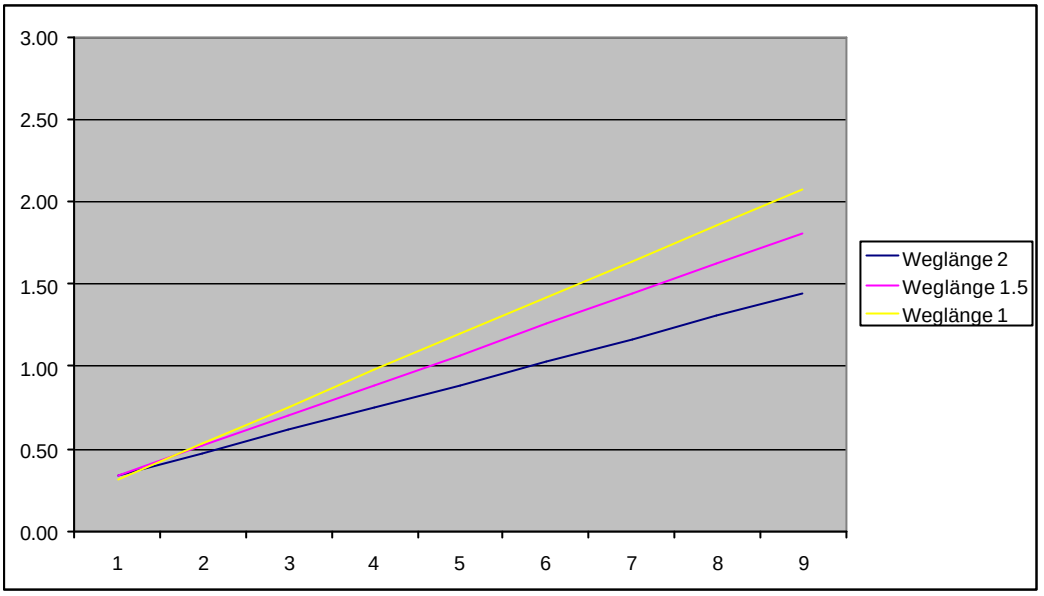
Eigenschaften		
schlecht	mittel	gut
0.73	0.50	0.27

Gewichtung eingeben:	
Aussicht:	0-1
Historischer Weg:	0-1
Belag:	0-1

1 sehr wichtig
0 gar nicht wichtig

Berechnung des Widerstandswertes Aufgrund der Weglänge und der Gewichtung (Gewichtung bei den Attributen jeweils die selbe)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	Gewicht
2	0.36	0.52	0.69	0.85	1.01	1.17	1.33	1.50	1.66	
1.5	0.31	0.46	0.62	0.77	0.93	1.09	1.24	1.40	1.55	
1	0.27	0.45	0.62	0.79	0.97	1.14	1.31	1.48	1.66	
Weglänge										



	Aussicht	Historisch	Belag
2	0.23	0.23	0.23
1.5	0.23	0.50	0.50
1	0.73	0.73	0.73

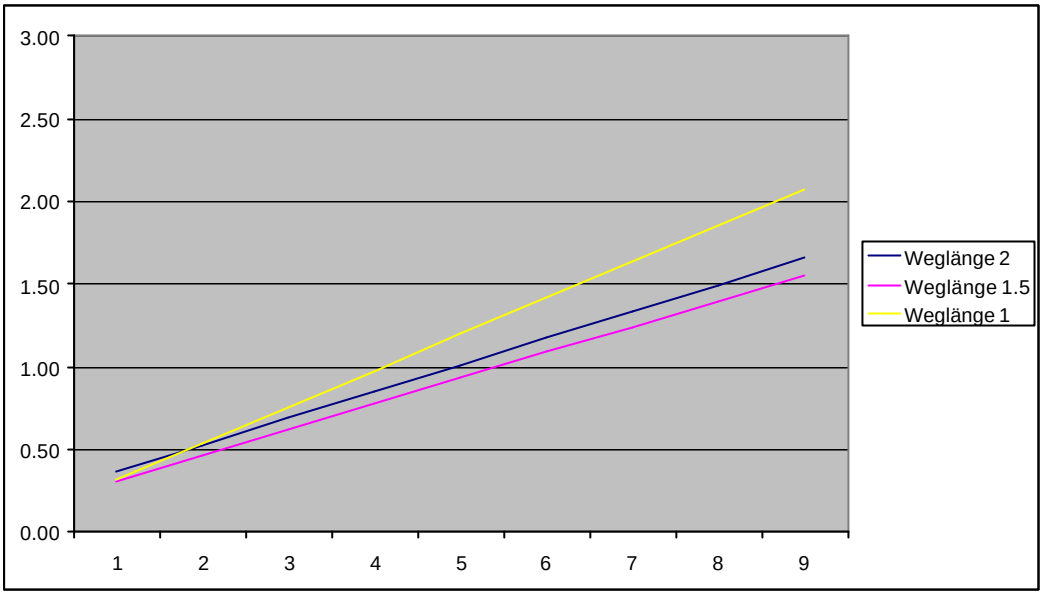
Eigenschaften		
schlecht	mittel	gut
0.73	0.50	0.23

Gewichtung eingeben:	
Aussicht:	0-1
Historischer Weg:	0-1
Belag:	0-1

1 sehr wichtig
0 gar nicht wichtig

Berechnung des Widerstandswertes Aufgrund der Weglänge und der Gewichtung (Gewichtung bei den Attributen jeweils die selbe)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	Gewicht
2	0.34	0.48	0.61	0.75	0.89	1.03	1.17	1.30	1.44	
1.5	0.33	0.52	0.70	0.89	1.07	1.26	1.44	1.63	1.81	
1	0.32	0.54	0.76	0.98	1.20	1.41	1.63	1.85	2.07	
Weglänge										



	Aussicht	Historisch	Belag
2	0.27	0.27	0.27
1.5	0.27	0.27	0.50
1	0.73	0.73	0.73

Eigenschaften		
schlecht	mittel	gut
0.73	0.50	0.27

Gewichtung eingeben:	
Aussicht:	0-1
Historischer Weg:	0-1
Belag:	0-1

1 sehr wichtig
0 gar nicht wichtig

Berechnung des Widerstandswertes Aufgrund der Weglänge und der Gewichtung (Gewichtung bei den Attributen jeweils die selbe)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	Gewicht
2	0.36	0.52	0.69	0.85	1.01	1.17	1.33	1.50	1.66	
1.5	0.31	0.46	0.62	0.77	0.93	1.09	1.24	1.40	1.55	
1	0.32	0.54	0.76	0.98	1.20	1.41	1.63	1.85	2.07	
Weglänge										

Anhang 5: Konfigurationsfile für Query-Manager

```
<?xml version="1.0"?>

<query-manager vers="1.0">
    <project>
        <shortDesc>StansTestDaten</shortDesc>
        <desc>Configuration file for the QueryManager for GreeceA (Geowarn)</desc>
    </project>

    <connection-list>

    <connection>
        <description>Connection to MS-Access J with dbc-ODBC bridge</description>
        <class-name>org.geowarn.querymanager.JdbcConnectionObject</class-name>
        <alias>conn_odbc_StansTestDaten</alias>
        <instances>1</instances>
        <config-param-list>
            <param name="jdbcDriver">sun.jdbc.odbc.JdbcOdbcDriver</param>
            <param name="url">jdbc:odbc:StansTestDaten</param>
        </config-param-list>
    </connection>

    </connection-list>

    <query-list>

    <!-- ARCS from StansTestDaten -->

        <query>
            <alias>getArcs</alias>
            <description>Get the Arcs of the whole graph</description>
            <connection>conn_odbc_StansTestDaten</connection>
            <class-name>org.geowarn.querymanager.JdbcQueryObject</class-name>
            <config-param-list>
                <param name="sqlStatement.1"><![CDATA[ SELECT * FROM A_1_all]]></param>
            </config-param-list>
            <param-list>
            </param-list>
        </query>

    </query-list>
</query-manager>
```