The 23rd ECMI Modelling Week
European Student Workshop
on Mathematical Modelling
in Industry and Commerce

Wrocław University of Technology, Wrocław, Poland, August 23-30, 2009

Report of project group 4 on

How to climb a mountain?

How to Climb a Mountain? Simulating efficient ways to the mountain top.

Marc-Nicolas Glöckner, *University of Paderborn, Germany*

Miłosz Marzec, *Wrocław University , Poland*

Nicodemus Banagaaya, *Eindhoven University of Technology, The Netherlands*

Rasmus Henningsson, *Lund University, Sweden*

Sina Meister, *Dresden University, Germany*

Till Schröter , *Oxford University, UK*

Instructor
Thomas Goetz, *Kaiserslautern University, Germany*

November 2, 2009

1 Introduction

During the recent 23rd ECMI Modelling Week in the beautiful city of Wroclaw in Poland, our group worked on the question of optimally reaching the top of a mountain.
This question is of relevance for many regions of the world. Whilst mountain rescue is an application where this knowledge can make a difference between life and death, the question is of greater importance for mountain communities that are mainly reached by foot and where survival and the standard of living depends on finding efficient ways of providing supply.
The problem was approached in two stages. The first stage focused on quantifying the problem and gaining the insights necessary to formulate the problem consistently. The second stage was concerned with implementing the quantitative procedures eventually leading to a solution of the problem.

To formalise the problem, we focused on two quantities that are pertinent in the given setting. One is the amount of the energy (in kcal) needed to reach the mountain top. The other is the time needed to arrive at the summit. Both quantities provide information about the quality of a chosen path. The total energy needed is of interest if the aim is to transport goods, whereas in emergency situations time is of greater importance.
To model the behaviour of a person walking at steady pace in mountainous terrain we had to

make sure that routes which are only accessible to climbers are not considered by the algorithm. This lead to a restriction on the maximum steepness of the walkers path to 45 degrees.
Based on those assumptions empirical data is available that links the slope of a path to the energy consumed [1], and to the velocity along the way [2].
Finally, using the available data we had to decide on a model of the hiker movements along data points. Altitude data of mountains is usually given on a square basis, where every square represents a region of same altitude. Based on that a hiker situated at the lower left corner of a square is in our simulation allowed to walk along the horizontal and vertical sides of the square and able to traverse it in some specific angles (as can be seen in figure 3).

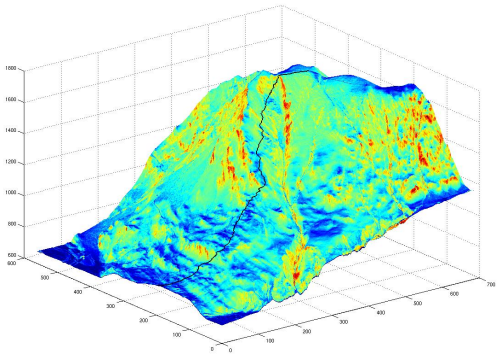

Figure 1: Energy optimal path up the mountain 'Wank'. - The colour represents the steepness of the slope. (red=steep)

In terms of the energy the slope a of a line between two grid points relates to the energy needed per unit length via

$$2.635 + 17.37a + 42.37a^2 - 21.43a^3 + 14.93a^4$$

according to [1]. In terms of time the slope a relates to the velocity in kilometres per hour via $6e^{(-3.5|a+0.05|)}$ according to [2].

Upon implementation of the Dijkstra algorithm, altitude data of the mountain 'Wank', situated in the Bavarian alps, along a $5m \times 5m$ grid was available to test the modelling approach.

The next section will be used to discuss the algorithm and its implementation.
Section 3 is concerned with the data interpretation. Finally in Section 4 we discuss different modelling approaches to the same problem and conclude.

2 Dijkstra's Algorithm

Dijkstra's algorithm is an algorithm to find the best path between a starting node and any other node in a graph with weighted edges. The weights should be nonnegative. Figure 2 illustrates a simple graph with two different paths between a start and an end node. The red path is optimal since it has a lower total cost.

The idea behind the algorithm is to search many paths at once and to always explore the cheapest paths first. This guarantees that whenever a path is extended to a new node, that must be the best path to that node from the starting node. (See 2.1 for details.)

To use Dijkstra's algorithm to solve our problem, we constructed a graph from the map data as follows:

- Nodes at map grid points (unless too steep).
- Edges between neighbouring points (as depicted in figure 3).
- Edge weight given by cost function.

It is considered too steep to walk through a node if the mountain slopes more than 45 degrees in any direction at that grid point.

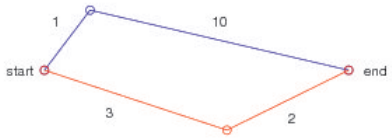

Figure 2: Simple graph

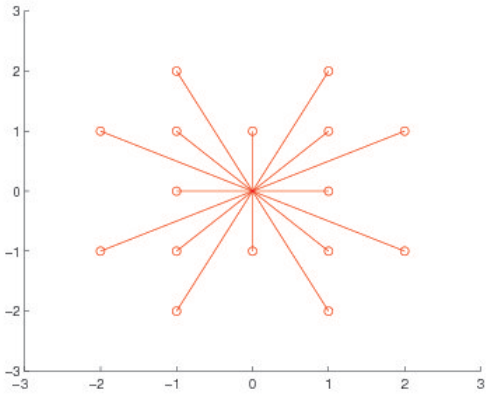

Figure 3: Node paths

2.1 Implementation of the algorithm

Since the graph is constructed from a map, data that needs to be stored per node can be conveniently put in a $M \times N$ matrix, where each element corresponds to a grid point in the map. For each node we need to store:

- The total cost to reach that node (initialised at infinity).
- The previous node in the optimal path leading to this node.

To keep track of what node to visit next, a priority queue is used. The queue is sorted by cost, i.e. at the front of the queue is the cheapest node to visit next. To get good performance, we implemented the queue using a binary heap. At initialisation, the starting node is put in the queue (with zero cost). The algorithm then runs like this:

- Get the next node from the priority queue.
- For each neighbour of this node:
  If this is the cheapest path found so far to that neighbour:
  - Update the cost matrix with the new cost.
  - Store this node as the previous node of the neighbour.
  - Put the neighbour in the priority queue.
- Repeat until the priority queue is empty.

The worst case performance can be shown to be $\mathcal{O}((E+N)\cdot\log(N))$, where E is the number of edges and N the number of nodes, when using a binary queue for the priority queue. To illustrate the algorithm running, figure 5-11 shows the state as more and more iterations are performed. We are looking at a small area around the starting node. Note that while it does not show in the images, each node is associated with a height and each edge with a cost. At the first iteration, the pattern of reachable nodes matches figure 3 as expected. During the next few iterations, paths to the left (downwards) are explored. Whenever we retrieve the next node from the priority queue, we have found the best path to that node. This is true since all other paths to that node must go through other nodes that are not yet visited and thereby the cost is greater (otherwise, one of those nodes would be at the front of

the queue). Thus by extending the visited area one node at a time, we are guaranteed to find the optimal path to each node. The last image shows the paths when all nodes in the area are visited. At two places close to the starting point, we can see paths crossing. We would normally not expect intersections when using Dijkstra's algorithm since there are never two paths arriving at the same node(the cheapest one would always be chosen). But these intersections do not occur at nodes but rather in between nodes. This is due to the discretization we have chosen of the map, where there are not nodes at all intersecting paths. Consider the case in figure 4. The two bottom nodes are at height a and the top ones at height b. Assume that the costs to arrive at the bottom nodes are equal (or almost equal). In this case, due to the non-linearity of the cost function, it might be cheaper to go from the bottom left node to the top right node than going from the bottom left node to the top left node. I.e. even if the path is longer, the cost might be lower since it is not as steep. The same argument holds for the bottom right node, thus giving the intersecting paths. In this light, the intersections can be viewed as an example of that it sometimes is more efficient to zigzag when it's too steep.

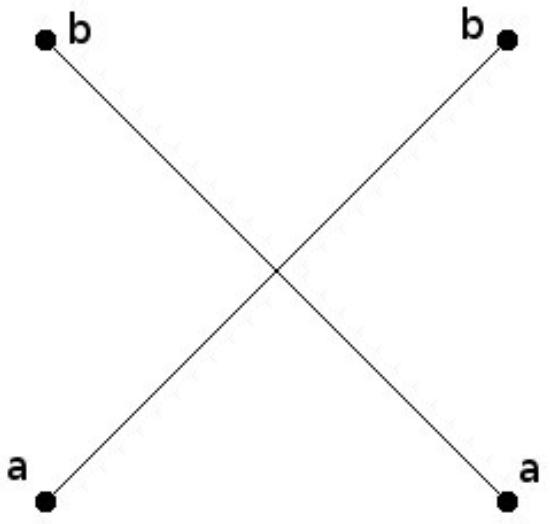

Figure 4: Paths might intersect when it's steep if it's cheaper to go diagonally than to go straight.

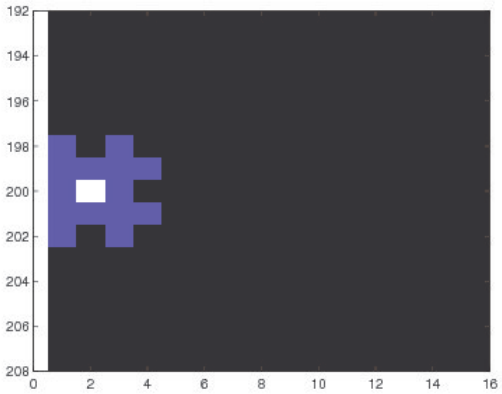

Figure 5: Step 1,,White = visited nodes, Blue = reachable nodes, Black = unreachable nodes, Red = paths

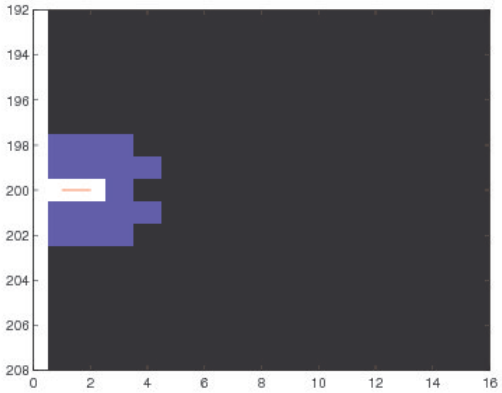

Figure 6: Step 2,White = visited nodes, Blue = reachable nodes, Black = unreachable nodes, Red = paths

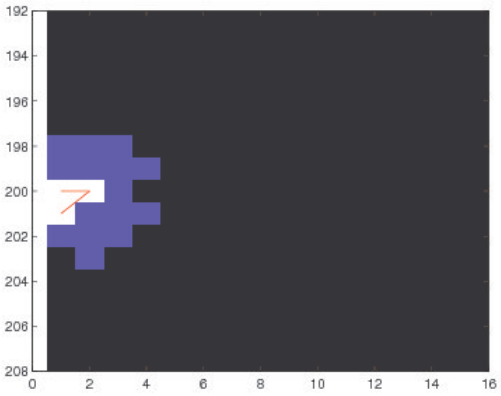

Figure 7: Step 3,White = visited nodes, Blue = reachable nodes, Black = unreachable nodes, Red = paths

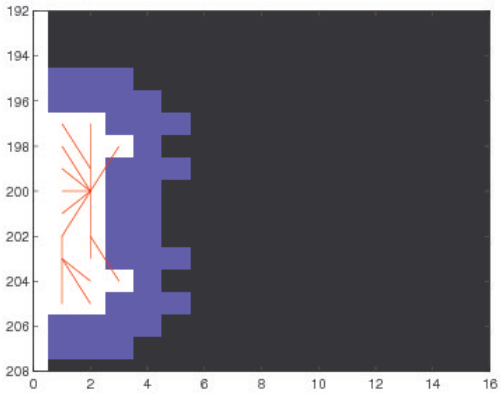

Figure 9: Step 20,White = visited nodes, Blue = reachable nodes, Black = unreachable nodes, Red = paths

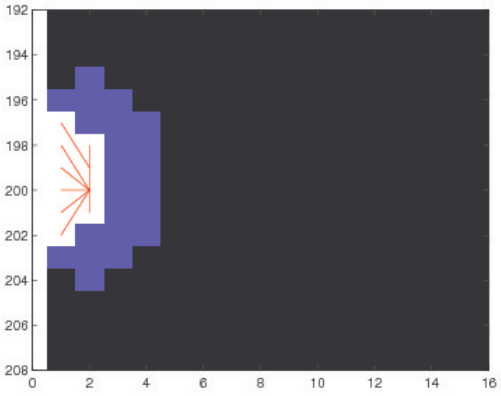

Figure 8: Step 10,White = visited nodes, Blue = reachable nodes, Black = unreachable nodes, Red = paths

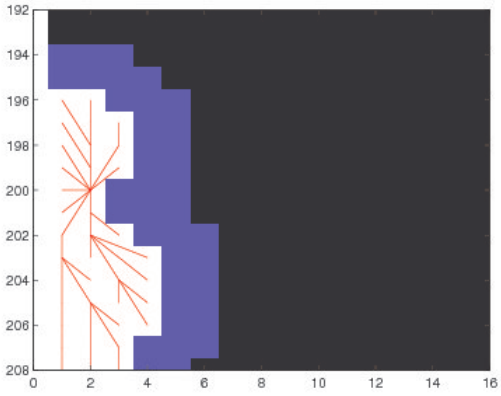

Figure 10: Step 40,White = visited nodes, Blue = reachable nodes, Black = unreachable nodes, Red = paths

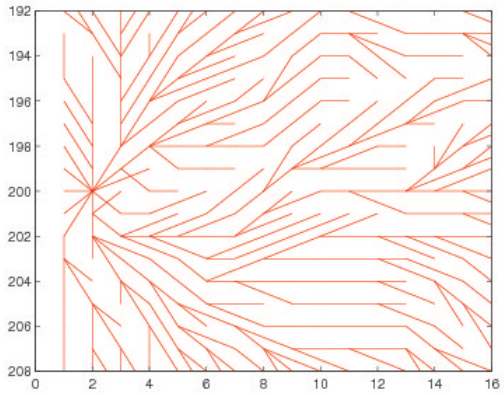

Figure 11: Step 2000,White = visited nodes, Blue = reachable nodes,
Black = unreachable nodes, Red = paths

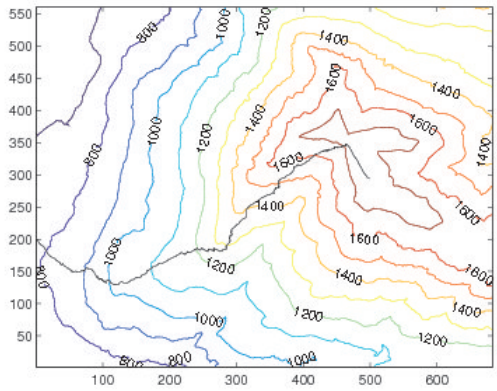

Figure 13: Solution of the default situation - The height is depicted with contours. (red=high)

3 Numerical outputs

Our default situation is to climb up mountain Wank starting nearby 'Garmisch-Partenkirchen' on a energy optimal path and not walking on flanks which are steeper than 45 degrees.

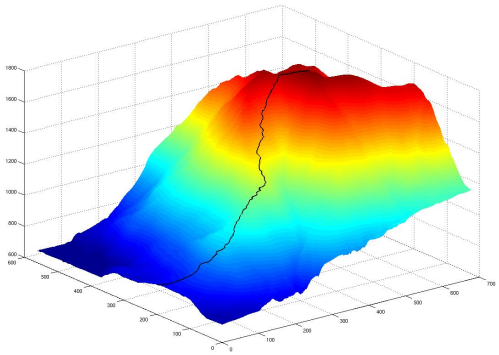

Figure 12: Solution of the default situation

This solution (see figure 12 and figure 13) is a little bit different to the actual paths on mountain Wank (see figure 14).

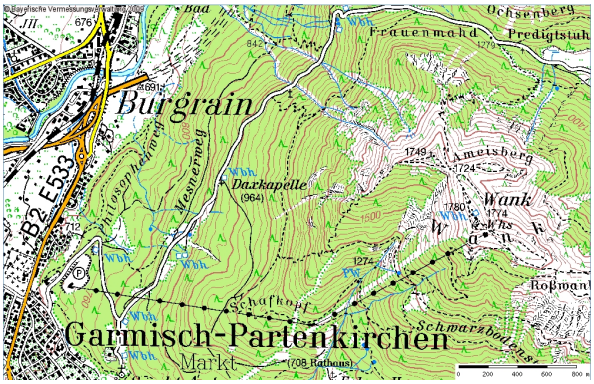

Figure 14: Map of mountain Wank

But it starts zigzagging along steep flanks of the mountain (see figure 15), a path pattern frequently observed in mountainous areas, and successfully avoids too steep slopes as can be seen in figure 16. The path is nowhere steeper than 25 degrees.

Figure 15: Zigzagging

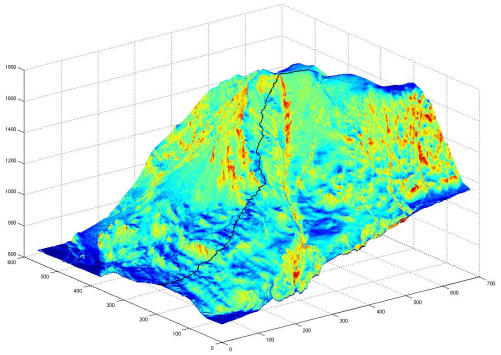

Figure 16: Solution of the default situation - The colour represents the steepness of the slope. (red=steep)

3.1 Different starting points

The starting point influences the path just in the beginning. The two different paths are meeting after half of the way (see figure 17).

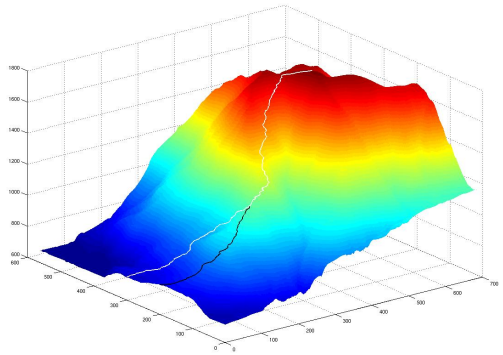

Figure 17: Different starting points: starting near 'Garmisch-Partenkirchen' (black) and near 'Burgrain' (white)

3.2 Different cost functions: energy & time optimal

Moreover, we tested time against energy optimal scenarios and it turned out that the paths only differed slightly. This comes to no surprise as the energy consumption function and the inverse of the velocity function are the same at every slope angle despite a scaling factor (see figure 19).

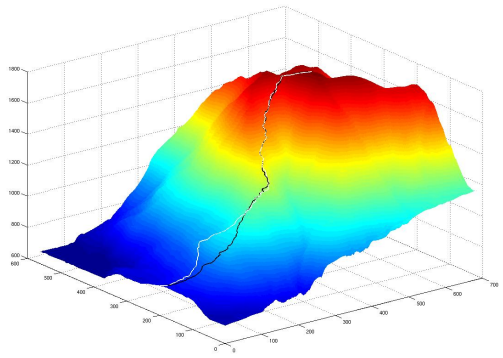

Figure 18: The energy optimal path (black) and the time optimal path (white).

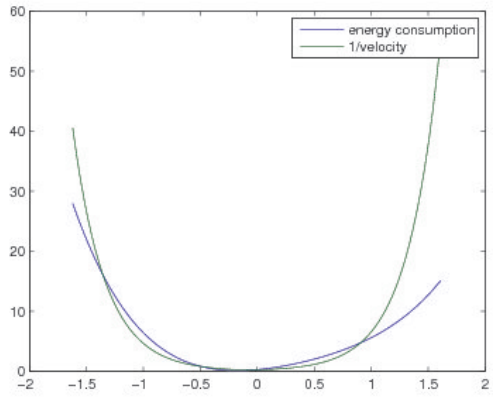

Figure 19: The energy consumption and the velocity depending on the slope.

	energy opt.	time opt.
Energy in $kcal$	926	927
Total time in h	1.9971	1.9931
Distance in km	4.3033	4.0874

3.3 Going up & down

Going up and down while minimising the energy consumption is leading to slightly different paths since the energy consumption function is not symmetric to zero (see figure 19) as it costs least energy to walk down a gentle slope.

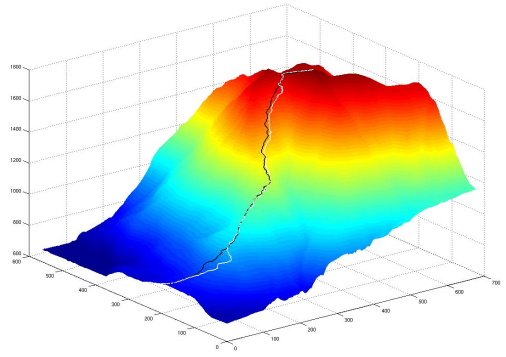

Figure 20: up & down (optimal energy)

Going up and down while minimising the time needed is leading to the same path. That is because of the symmetry (to zero) of the velocity function (see figure 19).

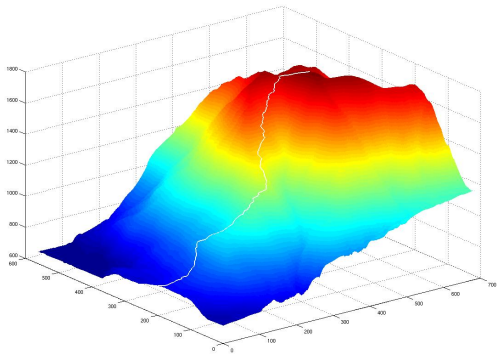

Figure 21: up & down (optimal time)

3.4 Steep flanks - unwalkable nodes

In our algorithm we prohibited walking on flanks ('unwalkable nodes') which are steeper than 45 degrees. Figures 22 to 25 show that this is a reasonable angle for our problem.

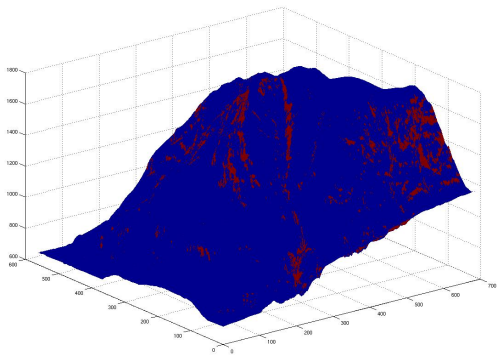

Figure 22: Flanks steeper than 45 are prohibited (red). Too steep areas are excluded, but the mountain area is walkable in most parts.

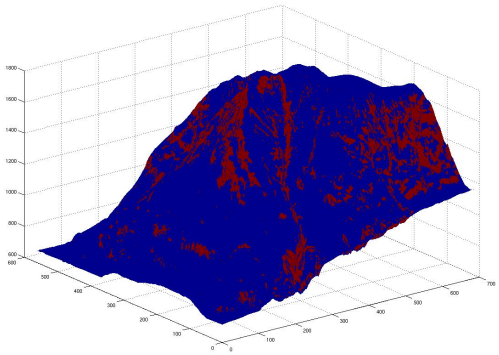

Figure 23: Flanks steeper than 40 are prohibited (red). The prohibited area is getting connected - it is getting difficult to walk the mountain without ending up at a dead end.

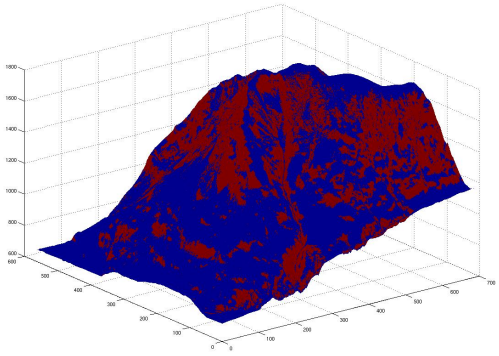

Figure 24: Flanks steeper than 35 are prohibited (red).

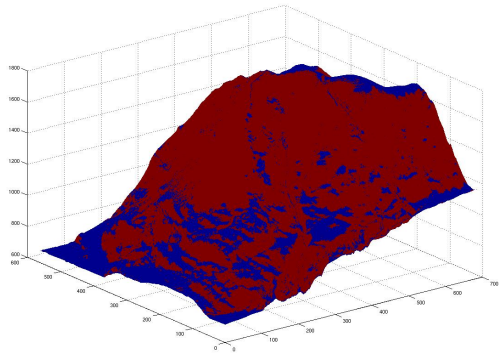

Figure 25: Flanks steeper than 25 are prohibited (red). Almost the whole mountain is unwalkable.

3.5 Another mountain: Mount Everest

Finally we tested the algorithm on Mount Everest altitude data (see figure 26). To our pleasant surprise the energy optimal path suggested by our algorithm coincides in general to the classical route via the Southeast Ridge (see figure 27). However, we had to loosen the steepness restriction of the walkable flanks and allow climbing at up to 90 degrees steep slopes for this solution. Anyway the path itself is not steeper than 30 degrees.

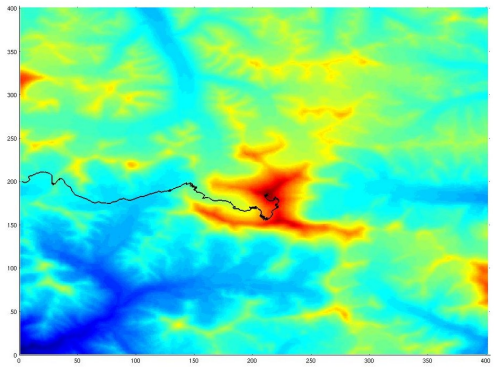

Figure 26: Mount Everest - Energy optimal path to the Mount Everest summit from the Base Camp. (red=high)

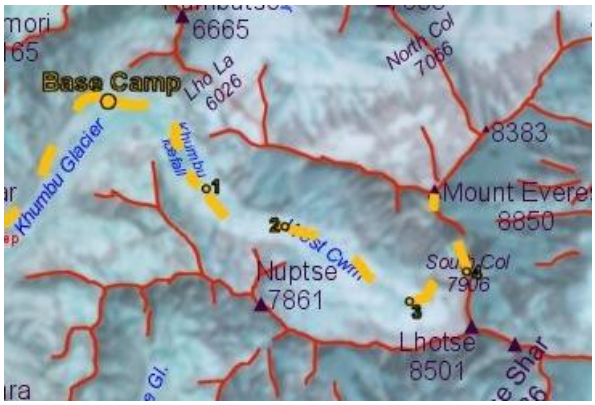

Figure 27: Mount Everest - Classical route to the Mount Everest summit from the Base Camp via Southeast Ridge.

4 Other models

In our solution we worked directly on the discrete $5m \times 5m$ grid point map. Alternatively you could search for the optimal path to the top on the continuous surface of the mountain. In our case the $5m \times 5m$ grid point map is the most detailed information we had about the surface of the mountain, so you have to interpolate the surface.
Again we need a cost function dependent on the slope of our path. But now our total cost is no longer a sum but an integral along the path. To be able to calculate the integral, of total energy needed, along the path we have to be able to determine the gradient in every point on the surface. For this reason we need the surface to be differentiable.
To find such a surface you can use bi-cubic splines or Bezier surfaces. Because Bezier surfaces differ too much in height on the grid points we prefer a bi-cubic spline interpolation.
When you finally have a differentiable surface you have to find am optimal way up the mountain. For this problem you can use the calculus of variations. In detail you search the path $u : [a, b] \to \mathbb{R} \times \mathbb{R}$ with $u(a) = x$ and $u(b) = y$, where x is the starting point and y is the end point and u is continuous so that u minimises the total energy spent. We define set of all functions u that fit in these conditions as X. To find this optimal u we have to define a cost function $F = F(x, u(x), u'(x))$. In our case the cost function depends on the slope of the path. We can find this slope by using the information of the position and direction of the path u in our 2-dimensional map to calculate the derivative on the surface. You can do this by algorithmic differentiation, for example. When you have the cost function, you search for $\inf\{I(u) = \int_a^b F(x, u(x), u'(x))\,\mathrm{d}x : u \in X\}$. This is a typical problem in calculus of variations and you can solve it with algorithms from this theory.
The advantage of this alternative ansatz is that you can choose between a lot more different paths to climb the mountain. In our algorithm we are limited on the finite number on grid points we have to walk on, if you use a continuous surface of the mountain you can walk in a lot more directions from each point and you are maybe able to walk on higher slopes because you are able to do zigzagging in a lot more different angles.
On the other hand the time needed to find a optimal solution grows a lot. You first have to interpolate the whole surface and after this you have to run very complex algorithms to find solutions for the problem of calculus of variations or a discrete approximation of it. Even if you limit the search space X to a smaller number of functions, for example continuous differentiable functions the time is still a lot higher than in our algorithm, what is important if we want to use the algorithm to find optimal paths along a mountain for mountain rescue for example.

5 Conclusion

Finally, we found an optimal path up the mountain in a very efficient way. There seems to be a strict limitation on the paths available because of the need to walk along grid points. But if we look at the differences between time and energy optimised paths, we see that the resulting energy spent or time needed differs only very little. In respect to the generality of the cost function, that has to be quite a rough approximation to reality, our limitation is not so bad.
To improve the model we propose to extend the cost function and allow a dynamic energy management along the path. Unfortunately we have no such function available that combines energy, slope and velocity. If you have such a function you can first optimise the energy management along the optimal

path we found, or even include the energy management in the search for an optimal path.
If you optimise the energy management along the path you could maybe run faster on steep slopes and slower on gentle slopes or the other way around to arrive at the top in the same time with less energy. If you include the energy management in you search algorithm, you maybe even choose a different path, were you climb higher slopes but with lower velocity to find a different optimal path.

References

[1] Llobera, Marcos and Tim Sluckin "Zigzagging: Theoretical Insights on climbing strategies", J. Theo. Bio, No 249, 206-217 (2007)

[2] Pingel, Thomas J. "Modelling slope as a contributor to route selection in mountainous areas", Working Paper (2009)