23. Human-Computer Interaction

In the internet—within http://www.hcibib.org/—the following definition is found: "Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them .... Some of its special concerns are:

- the joint performance of tasks by humans and machines;
- the structure of communication between human and machine;
- human capabilities to use machines (including the learnability of interfaces);
- algorithms and programming of the interface itself;
- engineering concerns that arise in designing and building interfaces;
- the process of specification, design, and implementation of interfaces.

... Human-computer interaction thus has science, engineering, and design aspects."

Many of these topics do only marginally concern algorithms in the classical sense. Therefore, in this chapter we concentrate on human-computer scenario for problem solving where the machines are forced to do lots of computation, and the humans have a role as intelligent controllers and directors.

23.1. Multiple-choice systems

Humans are able to think, to feel, and to sense—and they adopt quickly to a new situation. We can also compute, but not too well. In contrast, computers are giants in computing—they crunch bits and bytes like maniacs. However, they cannot do anything else but computing—especially they are not very flexible. Combining the different gifts and strengths of humans and machines in appropriate ways may lead to impressive performances.

One suitable approach for such team work is "*Multiple-Choice Optimisation*". In a "Multiple-Choice System" the computer gives a clear handful of candidate solutions, two or three or five ... Then a human expert makes the final choice amongst these alternatives. One key advantage of a proper multiple-choice approach is that the human is not drown by deluges of data.

Multiple-Choice Systems may be especially helpful in realtime scenarios of the following type: In principle there is enough time to compute a perfect solution. But certain parameters of the problem are unknown or fuzzy. They concretise only in a very late moment, when there is no more time for elaborate computations. Now assume that the decision maker has used multiple-choice algorithms to generate some good candidate solutions in beforehand. When the exact problem data show up he may select an appropriate one of these alternatives in realtime.

An example from vehicle routing is given. A truck driver has to go from A to Z. Before the trip he uses PC software to find two or three different good routes and prints them out. During the trip radio gives information on current traffic jams or weather problems. In such moments the printed alternatives help the driver to switch routes in realtime.

However, it is not at all easy to have the computer finding good small samples of solutions. Naively, the most natural way seems to be the application of k-best algorithms: Given a (discrete) optimisation problem with some objective function, the k best solutions are computed for a prescribed integer k. However, such k-best solutions tend to be *micro mutations* of each other instead of *true alternatives*.

Figure 23.1 exhibits a typical pattern: In a grid graph of dimension 100×100 the goal was to find short paths from the lower left to the upper right corner. The edge lengths are random numbers, not indicated in the diagram. The 1000 (!) shortest paths were computed, and their union is shown in the figure. The similarity amongst all these paths is striking. Watching the picture from a somewhat larger distance will even give the impression of only a single path, drawn with a bushy pencil. (The computation of alternative short paths will also be the most prominent example case in Section 23.2)

Often the term´´multiple-choice" is used in the context of "multiple-choice tests". This means something completely different. The difference between multiple-choice optimisation and multiple-choice tests lies in the type and quality of the candidate solutions:

- In multiple-choice tests always at least one of the answers is "correct", whereas others may be right or wrong. Beforehand an authority (the test designer) has prepared the question together with the candidate answers and the decision which of them are correct ones.
- In the optimisation situation nothing is clear: Perhaps all of the candidate solutions are ok, but it may also be that they all are not appropriate. And there is typically no master who tells the human whether his choice is good or not. Because of this uncertainty many humans really need some initiation time to accept their role within a multiple-choice system.

23.1.1. Examples of multiple-choice systems

(1) **Short Paths**

Starting in the early 1990's, PC programs for vehicle routing have become more and more popular. In 1997 the Dutch software company AND was first to sell such a program which did not only compute the "best" (= shortest or quickest) route but

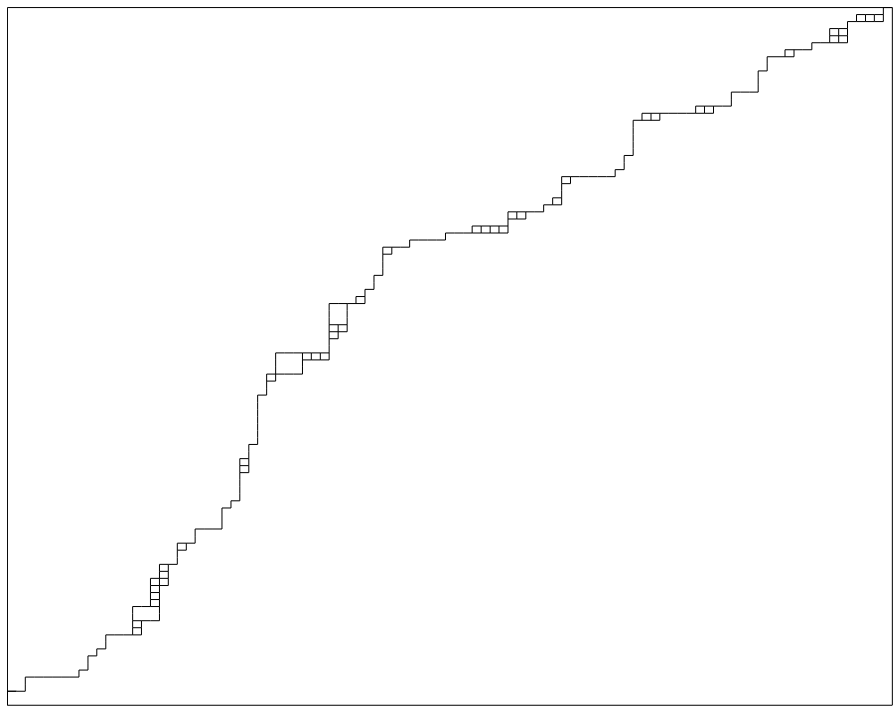

Figure 23.1 1000 shortest paths in a 100×100 grid-graph, printed in overlap.

also one or two alternatives. The user had the choice to request all these candidate solutions simultaneously or one after the other. The user was also allowed to determine some parameters for route visualisation, namely different colours and thickness for best, second, third choice. Related is work by F. Berger. She developed a method to identify linear structures (like roads, rails, rivers, ...) in grey level satellite images. Typically, candidate structures are not unique, and the algorithm of Berger makes several alternative proposals. The Berger method is based on algorithms for generating short alternative paths.

(2) **Travelling Salesperson Problem and the Drilling of Holes in Circuit Boards**

In the Travelling Salesperson Problem (TSP) N locations are given and their mutual distances. The task is to find a shortest round trip through all N points. TSP is NP-complete. One important application in electronic industry is the drilling of holes in circuit boards. Here the locations are the points where the drill has to make the holes; the goal is to minimise the time needed by the drill. In practice, however, it turns out that the length of the drilling tour is not the only criterion for success: Depending on the drilling tour there occur small or more severe tensions in the circuit board. Especially different tours may give different levels of tension. Unfortunately, the degrees of tension can not easily be computed in beforehand. So it makes sense

to compute a few alternative short drilling tours and select that one which is best with respect to the minimisation of tension.

(3) **Internet Search Engines**

In most cases an internet search engine will find tons of hits, but of course a normal human user is not able nor willing to look through all of them. So, one of the key tasks for a search engine designer is to find good shortlisting mechanisms. As a rule of thumb, the first ten hits in the output should be both most relevant and sufficiently spread. In this field and also in e-commerce Multiple-Choice Systems are often called "*Recommender Systems*".

(4) **Trajectories for Interplanetary Space Missions**

Space missions to distant planets, planetoids, and comets are high-tech adventures. Two key aspects are budget restrictions and the requirement that the probes need extremely high speeds to reach their destinations in time. "Gravity assist" maneuvers help to speed up missiles by narrow flybys at intermediate planets, thus saving fuel and time. In recent years trajectories with gravity assists have become more and more complex, typically involving whole sequences of several flybys. Prominent examples are the mission Cassini to planet Saturn with flyby sequence Venus-Venus-Earth-Jupiter, the mission Rosetta to Comet "67P/Churyumov-Gerasimenko" with flyby sequence Earth-Mars-Earth-Earth, and the Messenger-mission to Mercury with flyby sequence Earth-Venus-Venus-Mercury-Mercury. The current art of trajectory computing allows to finetune a principal route. However, first of all such principal routes have been designed by human engineers with their fantasy and creativity. Computer-generation of (alternative) principal flyby tours is still in its infancies.

(5) **Chess with Computer Assistance**

Commercial chess computers came up in the late 1970's. Their playing strength increases steadily, and nowadays the best PC programs play on one level with the best human players. However, teams with both human and computer members are stronger than humans alone or computers alone. One of these authors (Althöfer) made many chess experiments with Multiple-Choice Systems: In a setting called "*3-Hirn*" ("Triple Brain" in English, but the German term 3-Hirn has been adopted internationally) two different chess programs are running, typically on two independent PC's. Each one proposes a single candidate move, and a human player has the final choice amongst these (at most) two move candidates. In several experiments 3-Hirn showed amazing performance. The final data point was a match in 1997: two computer programs with Elo rating below 2550 each and a human amateur player (Elo 1900) beat the German No. 1 player (GM Yussupov, Elo 2640) by 5-3 in tournament play, thus achieving an event performance of higher than Elo 2700. After this event top human professionals were no longer willing to fight against 3-Hirn teams. The strength of 3-Hirn is to a large extent explained by the combination of two "orthogonal" chess strengths: chess computers propose only moves which are tactically sound and the human player contributes his strength in long-range planning.

Today, all top human chess professionals prepare intensively for their tournament games with the help of chess programs by analysing openings and games in multiple-choice mode. Even more extreme is the situation in correspondence chess, where players are officially allowed to use computer help within their games.

(6) **Travel and Holiday Information**
When someone plans a journey or a holiday, he typically compares different routes or offers, either at the railway station or in a travel agency or from home via internet. Customers typically do not inspect thousands of offers, but only a smaller or larger handful. In real life lots of (normal and strange) strategies can be found how companies, hotels, or airlines try to place their products amongst the top choices. For instance, it is common (bad) policy by many airlines to announce unrealistic short flight times. The only intention is to become top-placed in software (for travel agencies) which sorts all flights from A to B by ascending flight times. In many cases it is not an easy task for the customer to realize such tricks for successful "performance" in shortlisting processes.

(7) **RNA-Foldings**
Computation of RNA-foldings is one of the central topics in computational biology. The most prominent algorithms for this are based on dynamic programming. There exist online repositories, where people get alternative solutions in realtime.

Exercises

23.1-1 Collect practice in operating a multiple-choice system by computer-aided play of the patience game FreeCell. Download the tool BigBlackCell (BBC) from http://www.minet.uni-jena.de/∼BigBlackCell/ and make yourself acquainted with the program. After some practising a normal user with the help of BBC should be able to solve in the average more than 60 FreeCell instances per hour.

23.2. Generating multiple candidate solutions

23.2.1. Generating candidate solutions with heuristics

Many optimisation problems are really hard, for instance the NP-complete ones. Exact (but slow) branch and bound procedures and unreliable (but quick) heuristics are two standard ways to find exact or approximate solutions. When the task is to generate several alternative solutions it is possible to make a virtue of necessity: there are normally many more good solutions than perfect ones – and different heuristics or heuristics with random elements will not always return the same good solution.

So, a simple strategy is to apply one or several heuristics repeatedly to the same problem, and to record the solutions generated during this process. Either, exactly as many solutions as needed are generated. Or a larger preliminary set of solutions is produced, giving the chance for improvement by shortlisting. Natural shortlisting criteria are quality and spread. Concerning spread, distance measures on the set of admissible solutions may be a helpful tool in conjunction with clustering algorithms.

Repeated runs of a single heuristic. The normal situation is that a heuristic contains randomness to a certain extent. Then no additional efforts are necessary: the heuristic is simply executed in independent runs, until enough different good solutions have been generated. Here we use the Travelling Salesperson Problem (TSP) for N points as an example to demonstrate the approaches. For exchange heuris-

tics and insertion heuristics on the TSP we give one example each, highlighting the probabilistic elements.

In the TSP with symmetric distances $d(i,j)$ between the points local search with 2-exchanges is a standard exchange heuristic. In the following pseudo-code $T(p)$ denote the p-th component of vector T.

LOCAL-SEARCH-WITH-2-EXCHANGES-FOR-TSP(N,d)

1 Generate a random starting tour $T=(i_1,i_2,\ldots,i_N)$.
2 **while** there exist indices p,q with $1\le p<q\le N$ and $q\ge p+2$, and
$d(T(p),T(p+1))+d(T(q),T(q+1))>$
$d(T(p),T(q))+d(T(p+1),T(q+1))$
▷ For the special case $q=N$ we take $q+1=1$.
3 **do** $T\leftarrow(i_1,\ldots,i_p,i_q,i_{q-1},\ldots,i_{p+1},i_{q+1},\ldots,i_N)$
4 compute the length l of tour T
5 **return** T,l

Random elements in this heuristic are the starting tour and the order in which edge pairs are checked in step 2. Different settings will lead to different local minima. In large problems, for instance with 1,000 random points in the unit square with Euclidean distance it is quite normal when 100 independent runs of the 2-exchange heuristic lead to almost 100 different local minima.

The next pseudo-code describes a standard insertion heuristic.

INSERTION-HEURISTIC-FOR-TSP(N,d)

1 generate a random permutation $(i_1,i_2,\ldots,i_N)$ from the elements of $\{1,2,\ldots,N\}$
2 $T\leftarrow(i_1,i_2)$
3 **for** $t\leftarrow 2$ **to** $N-1$
4 **do** find the minimum of $d(T(r),i_{t+1})+d(i_{t+1},T(r+1))-d(T(r),T(r+1))$
for $r\in\{1,\ldots,t\}$
▷ Here again $r+1=1$ for $r=t$.
let the minimum be at $r=s$
5 $T\leftarrow(T(1),\ldots,T(s),i_{t+1},T(s+1),\ldots,T(t))$
6 compute the length l of tour T
7 **return** T,l

So the elements are inserted one by one, always at the place where insertion results at minimal new length.

The random element is the permutation of the N points. Like for the 2-exchanges, different settings will typically lead to different local minima. Sometimes an additional chance for random choice occurs when for some step t the optimal insertion place is not unique.

Many modern heuristics are based on analogies to nature. In such cases the user has even more choices: In Simulated Annealing several (good) intermediate solutions from each single run may be taken; or from each single run of a Genetic Algorithm several solutions may be taken, either representing different generations or multiple

solutions of some selected generation.

A special technique for repeated exchange heuristics is based on the perturbation of local optima: First make a run to find a local optimum. Then randomise this first optimum by a sequence of random local changes. From the resulting solution start local search anew to find a second local optimum. Randomise this again and so on. The degree of randomisation steers how different the local optima in the sequence will become.

Even in case of a deterministic heuristic there may be chances to collect more than only one candidate solution: In tiebreak situations different choices may lead to different outcomes, or the heuristic may be executed with different precisions (=number of decimals) or with different rounding rules. In Subsection 23.2.4 penalty methods are described, with artificial modification of problem parameters (for instance increased edge lengths) in repeated runs. In anytime algorithms —like iterative deepening in game tree search—also intermediate solutions (for preliminary search depths) may be used as alternative candidates.

Collecting candidate solutions from different heuristic programs. When several heuristics for the same problem are available, each one of them may contribute one or several candidate solutions. The 3-Hirn setting, as described in item (5) of Subsection 23.1.1, is an extreme example of a multiple-choice system with more than one computer program: the two programs should be independent of each other, and they are running on distinct computers. (Tournament chess is played under strict time limits at a rate of three minutes per move. Wasting computational resources by having two programs run on a single machine in multi-tasking mode costs 60 to 80 rating points [9]). The chess programs used in 3-Hirn are standard of-the-shelf products, not specifically designed for use in a multiple-choice setting.

Every real world software has errors. Multiple-choice systems with independent programs have a clear advantage with respect to catastrophic failures. When two independent programs are run, each with the same probability p for catastrophic errors, then the probability for a simultaneous failure reduces to p^2. A human controller in a multiple-choice system will typically recognise when candidate solutions have catastrophic failures. So the "mixed" case (one normal and one catastrophic solution) with probability $2p(1-p)$ will not result in a catastrophe. Another advantage is that the programs do not need to have special k-best or k-choice mechanisms. Coinciding computer proposals may be taken as an indication that this solution is just really good.

However, multiple-choice systems with independent programs may also have weak spots:

- When the programs are of clearly different strength, this may bring the human selector in moral conflicts when he prefers a solution from the less qualified program.
- In multistep actions the proposals of different programs may be incompatible.
- For a human it costs extra time and mental energy to operate more than one program simultaneously.
- Not seldom – depending on programs and operating systems – a PC will run

unstably in multi-tasking mode.

And of course it is not always guaranteed that the programs are really independent. For instance, in the late 1990's dozens of vehicle routing programs were available in Germany, all with different names and interfaces. However, they all were based on only four independent program kernels and data bases.

23.2.2. Penalty method with exact algorithms

A more controlled way to find different candidate solutions is given by the **penalty method**. The main idea of this method is illustrated on the route planning example. Starting with an optimal (or good) route R_1 we are looking for an alternative solution R_2 which fulfills the following two criteria as much as possible.

(i) R_2 should be good with respect to the objective function. Otherwise it is not worthwhile to choose it. In our example we have the length (or needed time) of the route as first objective.

(ii) R_2 should have not too much in common with the original solution. Otherwise it is not a *true* alternative. In case of so called *micro mutations* the risk is high that all these similar candidates have the same weak spots. In our example a "measure for similarity" may be the length of the parts shared by R_1 and R_2.

This means R_2 should have a short length but it should also have only little in common with R_1. Therefore we use a combination of the two objective functions – the length of the route and the length of the road sections shared by R_1 and R_2. This can be done by punishing the sections used by R_1 and solving the shortest path problem with this modified lengths to get the solution R_2.

By the size of the penalties different weightings of criteria (i) and (ii) can be modelled.

A most natural approach is to use **relative penalty factors**. This means that the length of each section belonging to R_1 is multiplied by a factor $1+\varepsilon$.

PENALTY-METHOD-WITH-RELATIVE-PENALTY-FACTORS(G, s, t, ε)

1 find the shortest path R_1 from node s to node t in the weighted graph $G = (V, E, w)$
2 **for** all $e \in E$
3 **do if** e belongs to R_1
4 **then** $\widehat{w}(e) \leftarrow w(e) \cdot (1+\varepsilon)$
5 **else** $\widehat{w}(e) \leftarrow w(e)$
6 find the the shortest path R_2 from node s to node t in the modified graph$\widehat{G} = (V, E, \widehat{w})$
7 compute its unmodified length $w(R_2)$
8 **return** (R_1, R_2) and $(w(R_1), w(R_2))$

Consider the following example.

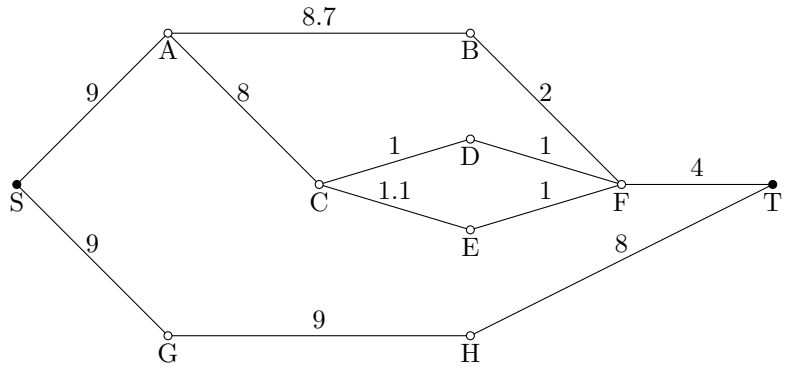

Figure 23.2 The graph for Examples 23.1, 23.2 and 23.6.

Example 23.1 Given is a graph $G = (V, E)$ with weighted edge lengths. In Figure 23.2 the numbers denote the length of the according edges. The shortest path from S to T is P_D via S - A - C - D - F - T with length 23. Multiplying all edge lengths of P_D by 1.1 and solving the obtained shortest path problem gives the alternative solution P_B via S - A - B - F - T with modified length 25.0 and normal length 23.7. The shared parts of P_D and P_B are S-A and F-T with total length 13.

The size of ε has to be appropriate for the situation. In the commercial vehicle routing program [**?**] all sections of a shortest (or fastest) route were multiplied by 1.2, i.e., $\varepsilon = 0.2$. Then the alternative route was computed. In [**?**] recognition of linear structures (streets, rivers, airport lanes) in satellite images was done by shortest path methods. Here $\varepsilon = 1.0$ turned out to be an appropriate choice for getting interesting alternative candidates.

Instead of relative penalty factors **additive penalties** might be used. That means we add a constant term ε to all edges we want to punish. The only modification of the algorithm above is in step 4.

4^* **then** $\widehat{w}(e) \leftarrow w(e) + \varepsilon$

Example 23.2 Given is the graph $G = (V, E)$ from Example 23.1 (see Figure 23.2). The shortest path from S to T is still P_D via S - A - C - D - F - T with length 23. Adding 0.1 to all edges of P_D and solving the resulting shortest path problem gives the alternative solution P_E via S - A - C - E - F - T with modified length 23.4 and normal length 23.1. P_D and P_E have three edges in common.

In principle this approach with additive penalties is not worse in comparison with multiplicative penalties. However, the method with multiplicative penalties has the advantage to be immune against artificial splits of edges.

For a generalisation of the penalty method from routing problems the following definition of optimisation problems is helpful.

Definition 23.1 *Let E be an arbitrary finite set and S a set of subsets of E. E is called the* ***base set*** *and the elements of S are* ***feasible subsets*** *of E. Let $w : E \to \mathbb{R}$ be a real valued weight function on E. For every $B \in S$ we set $w(B) = \sum_{e \in B} w(e)$.*

The optimisation problem $\min_{B \in S} w(B)$ is a ***Sum Type Optimisation Problem*** *or in short "$\sum$-**type problem**".*

Remarks:

1. The elements $B \in S$ are also called ***feasible solutions.***
2. By substitution of w by $-w$ every maximisation problem can be formulated as a minimisation problem. Therefore we will also call maximisation problems $\sum$-type problems.

Examples of $\sum$-type problems

- Shortest Path Problem
- Assignment Problem
- Travelling Salesperson Problem (TSP)
- Knapsack Problem
- Sequence Alignment Problem

Example 23.3 Consider the Knapsack Problem. Given a set of items $I = \{I_1, I_2, \ldots, I_n\}$, a weight function $w : I \to \mathbb{R}^+$, a value function $v : I \to \mathbb{R}^+$, and a knapsack capacity C. What is the most valuable collection of items whose weight sum does not exceed the knapsack capacity?

Choosing I as base set and S as the family of all subsets whose weight sum is smaller or equal to C gives a representation as a $\sum$-type problem: maximise $v(B)$ over all $B \in S$.

Abstract formulation of the penalty method for $\sum$-type problems

Definition 23.2 *Let E be an arbitrary set and S the set of feasible subsets of E. Let $w : E \to \mathbb{R}$ be a real-valued and $p : E \to \mathbb{R}^{\geq 0}$ a non-negative real-valued function on E.*

For every $\varepsilon > 0$, let B_ε be one of the optimal solutions of the problem

$$\min_{B \in S} f_\varepsilon(B) \ ,$$

$$\text{with} \quad f_\varepsilon(B) := w(B) + \varepsilon \cdot p(B) \ .$$

With an algorithm which is able to solve the unpunished problem $\min_{B \in S} w(B)$ we can also find the solutions $B\varepsilon$. We just have to modify the function w by replacing $w(e)$ by $w(e) + \varepsilon \cdot p(e)$ for all $e \in E$. B_ε is called an ***ε−penalty solution*** *or an* ***ε−alternative.***

Additionally we define the solution B_∞ of the problem

$$\operatorname{lex}\min_{B\in S}(p(B), w(B)) \quad \textit{(minimisation with respect to the lexicographical order)},$$

which has a minimal value $p(B)$ and among all such solutions a minimal value $w(B)$.

Remark. If both w and p are positive real-valued functions, there is a symmetry in the optimal solutions: B^* is an $\varepsilon-$penalty solution $(0 < \varepsilon < \infty)$ for the function pair (w, p), if and only if B^* is a $\frac{1}{\varepsilon}-$penalty solution for the pair (p, w).

To preserve this symmetry it makes sense to define B_0 as an optimal solution of the problem

$$\operatorname{lex}\min_{B\in S}(w(B), p(B)) \ .$$

That means B_0 is not only an optimal solution for the objective function w, but among all such solutions it has also a minimal p-value.

Example 23.4 We formulate the concrete Example 23.1 from page 1105 in this abstract $\sum$-type formulation. We know the shortest path P_D from S to T and search for a "good" alternative solution. The penalty function p is defined by

$$p(e) = \begin{cases} w(e) & \text{if } e \text{ is an edge of the shortest path } P_D \ , \\ 0 & \text{else} \ . \end{cases}$$

Finding penalty solutions for all parameters $\varepsilon \geq 0$. Often it is a priori not clear which choice of the penalty parameter ε produces good and interesting alternative solutions. With a "divide-and-conquer" algorithm one is able to find *all* solutions which can be produced by *any* parameter ε.

For finite sets S we give an efficient algorithm which generates a "small" set $\mathcal{B} \subseteq S$ of solutions with the following properties.

- For each element $B \in \mathcal{B}$ there exists an $\varepsilon \in \mathbb{R}^+ \cup \{\infty\}$ such that B is an optimal solution for the penalty parameter ε.
- For each $\varepsilon \in \mathbb{R}^+ \cup \{\infty\}$ there exists an element $B \in \mathcal{B}$ such that B is optimal for the penalty parameter ε.
- $\mathcal{B}$ has a minimal number of elements among all systems of sets which have the two properties above.

We call a solution B which is optimal for at least one penalty parameter **penalty-optimal**. The following algorithm finds a set of penalty-optimal solutions which covers all $\varepsilon \in \mathbb{R}^+ \cup \{\infty\}$.

For easier identification we arrange the elements of the set $\mathcal{B}$ in a fixed order $(B_{\varepsilon(1)}, B_{\varepsilon(2)}, \ldots, B_{\varepsilon(k)})$, with $0 = \varepsilon(1) < \varepsilon(2) < \cdots < \varepsilon(k) = \infty$.
The algorithm has to check that for $\varepsilon(i) < \varepsilon(i{+}1)$ there is no ε with $\varepsilon(i) < \varepsilon < \varepsilon(i{+}1)$ such that for this penalty parameter ε neither $B_{\varepsilon(i)}$ nor $B_{\varepsilon(i+1)}$ is optimal. Otherwise it has to identify such an ε and an ε-penalty solution B_ε. In step 11 of the pseudo code below the variable Border(i) is set to 1 if it turns out that such an intermediate ε does not exist.

We present the pseudocode and give some remarks.

Algorithm for finding a sequence $\mathcal{B}$ of penalty-optimal solutions covering all $\varepsilon \geq 0$ for the problem

$$\min_{B \in S} f_\varepsilon(B)$$
$$\text{with } f_\varepsilon(B) = w(B) + \varepsilon \cdot p(B) \quad .$$

DIVIDE-AND-COVER(w, p)

compute B_0, which minimises $w(B)$ and has a $p(B)$-value as small as possible.
compute B_∞, which minimises $p(B)$ and has a $w(B)$-value as small as possible.
if $p(B_0) = p(B_\infty)$
  then $\mathcal{B} \leftarrow \{B_0\}$; $\mathcal{E} \leftarrow (0)$; Border$\leftarrow \emptyset$
    (B_0 minimises the functions w *and* p and is optimal for all ε.)
  else $k \leftarrow 2$; $\mathcal{E} = (\varepsilon(1), \varepsilon(2)) \leftarrow (0, \infty)$; Border$(1) \leftarrow 0$; $\mathcal{B} \leftarrow (B_0, B_\infty)$.
    while There is an $i \in \{1, 2, \ldots, k-1\}$ with Border$(i) = 0$.
    do $\overline{\varepsilon} \leftarrow \frac{w(B_{\varepsilon(i+1)}) - w(B_{\varepsilon(i)})}{p(B_{\varepsilon(i)}) - p(B_{\varepsilon(i+1)})}$
      Find an optimal solution $B_{\overline{\varepsilon}}$ for the parameter $\overline{\varepsilon}$.
      if $f_{\overline{\varepsilon}}(B_{\overline{\varepsilon}}) = f_{\overline{\varepsilon}}(B_{\varepsilon(i)}) = f_{\overline{\varepsilon}}(B_{\varepsilon(i+1)})$
        then Border$(i) \leftarrow 1$
        else $\mathcal{B} \leftarrow (B_{\varepsilon(1)}, \ldots, B_{\varepsilon(i)}, B_{\overline{\varepsilon}}, B_{\varepsilon(i+1)}, \ldots, B_{\varepsilon(k)})$
          $\mathcal{E} \leftarrow (\varepsilon(1), \ldots, \varepsilon(i), \overline{\varepsilon}, \varepsilon(i+1), \ldots, \varepsilon(k))$
          Border$\leftarrow$ (Border$(1), \ldots,$ Border$(i), 0,$Border$(i+1), \ldots,$ Border$(k-1))$
          $k \leftarrow k + 1$
return $\mathcal{B}, \mathcal{E}$, Border

At the end $\mathcal{B}$ is a sequence of different penalty-optimal solutions and the vector $\mathcal{E}$ includes consecutive epsilons.

This algorithm is based on the following properties:

(1) If B is an ε-optimal solution then there exists an interval $I_B = [\varepsilon_l, \varepsilon_h]$, $\varepsilon_l, \varepsilon_h \in \mathbb{R} \cup \{\infty\}$, such that B is optimal for all penalty parameters $\varepsilon \in I_B$ and for no other parameters.

(2) For two different solutions B and B' with nonempty optimality intervals I_B and $I_{B'}$, only three cases are possible.

- $I_B = I_{B'}$. This happens iff $w(B) = w(B')$ and $p(B) = p(B')$.
- I_B and $I_{B'}$ are disjoint.
- $I_B \cap I_{B'} = \{\overline{\varepsilon}\}$, this means the intersection contains only a single epsilon. This happens if I_B and $I_{B'}$ are neighbouring intervals.

By finiteness of E there are only finitely many feasible solutions $B \in S$. So, there can be only finitely many optimality intervals. Hence, (1) and (2) show that the interval $[0, \infty]$ can be decomposed into a set of intervals $\{[0 = \varepsilon_1, \varepsilon_2], [\varepsilon_2, \varepsilon_3], \ldots, [\varepsilon_k, \varepsilon_{k+1} = \infty]\}$. For each interval we have a different solution B which is optimal for all ε in this interval. We call such a solution an **interval representative.**

(3) The aim of the algorithm is to find the *borders* of such optimality intervals and for each interval a representing solution. In every iteration step an interval representative of a new interval or a new border between two different intervals will be found (in steps 7–13). When there are k optimality intervals with $k \geq 2$ it is sufficient to solve $2k-1$ problems of the type $\min_{B \in S} w(B) + \varepsilon \cdot p(B)$ to detect all of them and to find representing solutions.

Unimodality property of the alternatives. When only one ε-alternative shall be computed the question comes up which penalty parameter should be used to produce a "best possible" alternative solution. If the penalty parameter is too small the optimal and the alternative solution are too similar and offer no real choice. If the parameter is too large the alternative solution becomes too poor. The best choice is to take some "medium" ε.

We illustrate this effect in the following route planning example.

Example 23.5 Assume that we have to plan a route from a given starting point to a given target. We know the standard travel times of all road sections and are allowed to plan for *two* different routes. In last minute we learn about the real travel times and can choose the fastest of our two candidate routes.

Let the first route be the route with the smallest standard travel time and the second one a route found by the penalty method. Question: Which penalty parameter should we use to minimise the real travel time of the fastest route?

Concretely, consider randomly generated instances of the shortest path problem in a weighted directed grid graph $G = (V, E, w)$ of dimension 25×25. The weights of the arcs are independently uniformly distributed in the unit interval $[0, 1]$. We compute P_0, a path from the lower left corner to the upper right with minimal weight. Afterwards we punish the edges of path P_0 by multiplying by $1 + \varepsilon$ and calculate a whole set of ε-penalty solutions $P_{\varepsilon_1}, P_{\varepsilon_2}, \ldots, P_{\varepsilon_{30}}$ for $\varepsilon = 0.025,\ 0.050,\ \ldots,\ 0.750$. We get 30 solution pairs $\{S_0, S_{\varepsilon_1}\}, \{S_0, S_{\varepsilon_2}\}, \ldots, \{S_0, S_{\varepsilon_{30}}\}$ and can compare these.

The weight $w(e)$ of an arc e is the *standard travel time without time lags*, i.e. the minimal needed travel time on a free road without any traffic jam. The *real* travel time $\hat{w}(e)$ of this arc may differ from $w(e)$ as follows:

$$\hat{w}(e) = \begin{cases} \lambda_c(e) \cdot w(e) & : \text{ with probability } p \\ w(e) & : \text{ with probability } 1-p \end{cases}$$

independently for all edges e. Here the $\lambda_c(e)$ are independent random numbers, uniformly distributed in the interval $[1, c]$. The parameter $0 \leq p \leq 1$ is called **failure probability** and the parameter $c \geq 1$ is called **failure width**.

For every pair $\{S_0, S_{\varepsilon_i}\}$ we calculate the minimum of the two function values $\hat{w}(S_0)$ and $\hat{w}(S_{\varepsilon_i})$. To get a direct impression of the benefit of having two solutions instead of one we scale with respect to the real value of the optimal solution S_0.

$$\phi_{\varepsilon_i} = \frac{\min\{\hat{w}(S_0), \hat{w}(S_{\varepsilon_i})\}}{\hat{w}(S_0)} \quad \text{for } i = 1, \ldots, 30\ .$$

We computed the values ϕ_{ε_i} for 100,000 randomly generated 25×25 grid graphs with failure probability $p = 0.1$ and failure width $c = 8$. Figure 23.3 shows the averages $\overline{\phi}_{\varepsilon_i}$ for $\varepsilon_1 = 0.025,\ \varepsilon_2 = 0.050,\ \ldots,\ \varepsilon_{30} = 0.750$.

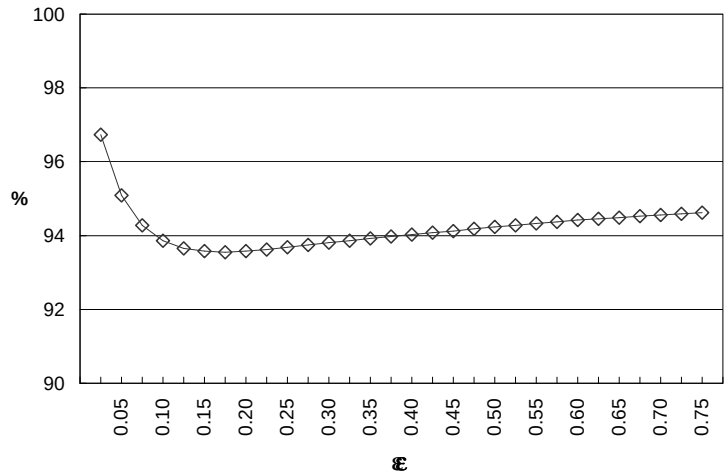

Figure 23.3 $\overline{\phi}_{\varepsilon_i}$ for $\varepsilon_1 = 0.025,\ \varepsilon_2 = 0.050,\ \ldots,\ \varepsilon_{30} = 0.750$ on 25×25 grids.

As seen in Figure 23.3, the expected quality $\phi\varepsilon$ of the solution pairs is unimodal in ε. That mean that $\phi\varepsilon$ first decreases and then increases for growing ε. In this example $\varepsilon^* \approx 0.175$ is the optimal penalty parameter.

In further experiments it was observed that the optimal parameter ε^* is decreasing in the problem size (e.g. $\varepsilon^* \approx 0.6$ for shortest paths in 5×5-grids, $\varepsilon^* \approx 0.175$ for 25×25 and $\varepsilon^* \approx 0.065$ for 100×100 grid graphs).

Monotonicity properties of the penalty solutions. Independently whether all ε-penalty solutions are generated or only a single one (as in the prevoius pages), the following structural properties are provable: With increasing penalty factor ε we get solutions $B\varepsilon$ where

- the penalty part p of the objective function is fitted monotonically better (the solution contains less punished parts),
- the original objective function w is getting monotonically worse, in compensation for the improvement with respect to the penalty part.

These facts are formalised in the following theorem.

Theorem 23.3 *Let $w : E \to \mathbb{R}$ be a real-valued function and $p : E \to \mathbb{R}^+$ a positive real-valued function on E. Let B_ε be defined for $\varepsilon \in \mathbb{R}^+$ according to Definition 23.2. The following four statements hold:*
(i) $p(B\varepsilon)$ is weakly monotonically decreasing in ε.
(ii) $w(B\varepsilon)$ is weakly monotonically increasing in ε.
(iii) The difference $w(B\varepsilon) - p(B\varepsilon)$ is weakly monotonically increasing in ε.
(iv) $w(B\varepsilon) + \varepsilon \cdot p(B\varepsilon)$ is weakly monotonically increasing in ε.

Proof Let δ and ε be two arbitrary nonnegative real numbers with $0 \leq \delta < \varepsilon$.

Because of the definition of $B\delta$ and $B\varepsilon$ the following inequalities hold.

(i) In case $\varepsilon < \infty$ we have

$$w(B\varepsilon) + \varepsilon \cdot p(B\varepsilon) \le w(B\delta) + \varepsilon \cdot p(B\delta)\ , \tag{23.1}$$

$$w(B\varepsilon) + \delta \cdot p(B\varepsilon) \ge w(B\delta) + \delta \cdot p(B\delta)\ . \tag{23.2}$$

Subtracting (23.2) from (23.1) we get

$$(\varepsilon - \delta) \cdot p(B\varepsilon) \le (\varepsilon - \delta) \cdot p(B\delta) \qquad | : (\varepsilon - \delta) > 0$$

$$\Leftrightarrow \quad p(B\varepsilon) \le p(B\delta)\ . \tag{23.3}$$

In case $\varepsilon = \infty$ inequality (23.3) follows directly from the definition of B_∞ .

(ii) Subtracting (23.3) multiplied with δ from (23.2) we get

$$w(B\varepsilon) \ge w(B\delta). \tag{23.4}$$

(iii) Subtracting (23.3) from (23.4) we get

$$w(B\varepsilon) - p(B\varepsilon) \ge w(B\delta) - p(B\delta).$$

(iv) With (23.2) and $\varepsilon > \delta \ge 0$ we have

$$w(B\delta) + \delta \cdot p(B\delta) \le w(B\varepsilon) + \delta \cdot p(B\varepsilon) \le w(B\varepsilon) + \varepsilon \cdot p(B\varepsilon)$$

$$\Rightarrow \quad w(B\varepsilon) + \varepsilon \cdot p(B\varepsilon) \ge w(B\delta) + \delta \cdot p(B\delta)\ .$$

■

Generating more than one alternative solution for the same penalty parameter ε. If we have a solution S_0 and want to get *more than one* alternative solution we can use the penalty method several times by punishing S_0 with different penalty parameters $\varepsilon_1 < \cdots < \varepsilon_m$, getting alternative solutions $S_{\varepsilon_1}, S_{\varepsilon_2}, \ldots, S_{\varepsilon_m}$. This method has a big disadvantage, because only the shared parts of the main solution S_0 and each alternative solution are controlled by the values ε_i. But there is no direct control of the parts shared by two different alternative solutions. So, S_{ε_i} and S_{ε_j} may be rather similar for some $i \ne j$.

To avoid this effect the penalty method may be used *iteratively* for the same ε.

ITERATIVE-PENALTY-METHOD(w, p, k, ε)

1 solve the original problem $\min w(B)$ and find the optimal solution S_0.
2 define the penalty function as $p_1(B) \leftarrow \varepsilon \cdot w(B \cap S_0)$.
3 solve the modified problem $\min w(B) + \varepsilon \cdot p_1(B)$ and find the solution S_1.
4 **for** $j \leftarrow 2$ **to** k
5 **do** $p_j(B) \leftarrow \varepsilon \cdot w(B \cap S_0) + \varepsilon \cdot w(B \cap S_1) + \cdots + \varepsilon \cdot w(B \cap S_{j-1})$
6 solve the modified problem $\min w(B) + \varepsilon \cdot p_j(B)$ and find the solution S_j.
7 **return** $(S_0, S_1, \ldots, S_k)$

Step 5 may be replaced by the variant 5*

5* **do** $p_j(B) \leftarrow \varepsilon \cdot w(B \cap (S_0 \cup S_1 \cup \cdots \cup S_{j-1}))$

In the first case (5) a part of a solution belonging to r of the j solutions $S_0, S_1, \ldots$ and S_{j-1} is punished by the factor $r \cdot \varepsilon$. In the second case (5*) a part of a solution is punished with multiplicity one if it belongs to at least one of $S_0, S_1, \ldots$ or S_{j-1}.

The differences in performance are marginal. However, in shortest path problems with three solutions S_0, S_1 and S_2 setting (5) seemed to give slightly better results.

Example 23.6 Take the graph $G = (V, E)$ from Figure 23.2. For penalty parameter $\varepsilon = 0.1$ we want to find three solutions. The shortest path from S to T is P_D via S - A - C - D - F - T with length 23. Multiplying all edges of P_D by 1.1 and solving the obtained shortest path problem gives the alternative solution P_B via S - A - B - F - T.

Applying setting (5) we have to multiply the edge lengths of (A, C), (C, D), (D, F), (A, B) and (B, F) by penalty factor 1.1. The edges (S, A) and (F, T) have to be multiplied by factor 1.2 (double penalty). The optimal solution is path P_H via S - G - H - T.

Applying setting (5*) we have to multiply the edge lengths (S, A), (A, C), (C, D), (D, F), (F, T), (A, B) and (B, F) by penalty factor 1.1. The optimal solution of this modified problem is path P_E via S - A - C - E - F - T.

23.2.3. The linear programming - penalty method

It is well known that shortest path problems as well as many other network flow problems can be solved with **Linear Programming**. Linear Programming may also be used to generate alternative solutions. We start with the description of Linear Programming for the basic shortest path problem.

The shortest path problem formulated as a linear program.

Consider a directed graph $G = (V, E)$ and a function $w : E \to \mathbb{R}^+$ assigning a length to every arc of the graph. Let s and t be two distinguished nodes of G.

Which is the shortest simple path from s to t in G?

For every arc $e = (i, j) \in E$ we introduce a variable x_e. Here x_e shall be 1 if e is part of the shortest path and x_e shall be 0 otherwise.

With $S(i) = \{j \in V : (i, j) \in E\} \subseteq V$ we denote the set of the successors of node i and with $P(i) = \{j \in V : (j, i) \in E\} \subseteq V$ we denote the set of the predecessors of node i. The linear program $LP_{ShortestPath}$ is formulated as follows:

$$\min \sum_{e \in E} w(e) \cdot x_e$$

$$\text{s.t.} \sum_{j \in S(s)} x_{(s,j)} - \sum_{j \in P(s)} x_{(j,s)} = 1 \quad \text{flow-out condition for the starting node } s$$

$$\sum_{j \in S(t)} x_{(t,j)} - \sum_{j \in P(t)} x_{(j,t)} = -1 \quad \text{flow-in condition for the target node } t$$

$$\sum_{j \in S(i)} x_{(i,j)} - \sum_{j \in P(i)} x_{(j,i)} = 0 \quad \text{for all nodes } i \in V \backslash \{s, t\}$$

KIRCHHOFF conditions for all interior nodes

$$0 \leq x_e \leq 1 \text{ for all } e \in E.$$

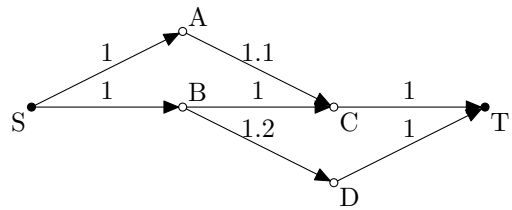

Figure 23.4 Example graph for the LP-penalty method.

By the starting and target conditions node s is a source and node t is a sink. Because of the KIRCHHOFF conditions there are no other sources or sinks. Therefore there must be a "connection" from s to t.

It is not obvious that this connection is a simple path. The variables x_e might have non-integer values or there could be circles anywhere. But there is a basic theorem for network flow problems [1, p. 318] that the linear program $LP_{ShortestPath}$ has an optimal solution where all $x_e > 0$ have the value 1. The according arcs with $x_e = 1$ represent a simple path from s to t.

Example 23.7 Consider the graph in Figure 23.4. The linear program for the shortest path problem in this graph contains six equality constraints (one for each node) and seven pairs of inequality constraints (one pair for each arc).

$$
\begin{aligned}
&\min(x_{SA} + x_{SB} + x_{BC} + x_{CT} + x_{DT}) \cdot 1 + x_{AC} \cdot 1.1 + x_{BD} \cdot 1.2 \\
&\text{s.t. } x_{SA} + x_{SB} = 1, \\
&\quad x_{CT} + x_{DT} = 1\ , \\
&\quad x_{SA} - x_{AC} = 0, \\
&\quad x_{SB} - x_{BC} - x_{BD} = 0\ , \\
&\quad x_{AC} + x_{BC} - x_{CT} = 0, \\
&\quad x_{BD} - x_{DT} = 0\ , \\
&\quad 0 \le x_{SA}, x_{SB}, x_{AC}, x_{BC}, x_{BD}, x_{CT}, x_{DT} \le 1\ .
\end{aligned}
$$

The optimal solution has $x_{SB} = x_{BC} = x_{CT} = 1$.

A linear program which gives two alternative paths from s to t

Here we give an LP-representation for the task to find *two* alternative routes from s to t.

For every arc $e = (i, j) \in E$ we introduce two variables x_e and y_e. If the arc e is used in both routes, then both x_e and y_e shall have the value 1. If e is a part of only one route, x_e shall be 1 and y_e shall be 0. Otherwise x_e and y_e shall both be 0. $\varepsilon > 0$ is a penalty parameter to punish arcs used by both routes.

With this in mind we can formulate the linear program $LP_{2-ShortPaths}$

$$\min f(x,y) := \sum_{e\in E} w(e)\cdot x_e + (1+\varepsilon)\cdot w(e)\cdot y_e$$

$$\text{s.t.} \sum_{j\in S(s)} x_{(s,j)} + y_{(s,j)} - \sum_{j\in P(s)} x_{(j,s)} + y_{(j,s)} = 2 \quad \text{condition for the starting node } s$$

$$\sum_{j\in S(t)} x_{(t,j)} + y_{(t,j)} - \sum_{j\in P(t)} x_{(j,t)} + y_{(j,t)} = -2 \quad \text{condition for the target node } t$$

$$\sum_{j\in S(i)} x_{(i,j)} + y_{(i,j)} - \sum_{j\in P(i)} x_{(j,i)} + y_{(j,i)} = 0 \quad \text{KIRCHHOFF conditions}$$

$$\text{for all } i \in V\backslash\{s,t\}$$

$$0 \le x_e, y_e \le 1 \text{ for all } e \in E\ .$$

Example 23.8 Consider again the graph in Figure 23.4. The linear program for the 2-alternative-paths problem in this graph contains six equality constraints (one for each node) and $2\cdot 7 = 14$ pairs of inequality constraints.

$$\begin{aligned}
\min\quad & (x_{SA}+x_{SB}+x_{BC}+x_{CT}+x_{DT})\cdot 1 + x_{AC}\cdot 1.1 + x_{BD}\cdot 1.2 \\
& + \big[(y_{SA}+y_{SB}+y_{BC}+y_{CT}+y_{DT})\cdot 1 + y_{AC}\cdot 1.1 + y_{BD}\cdot 1.2\big]\cdot(1+\varepsilon) \\
\text{s.t. } & x_{SA}+y_{SA}+x_{SB}+y_{SB} = 2, \\
& x_{CT}+y_{CT}+x_{DT}+y_{DT} = 2\ , \\
& x_{SA}+y_{SA}-x_{AC}-y_{AC} = 0, \\
& x_{SB}+y_{SB}-x_{BC}-y_{BC}-x_{BD}-y_{BD} = 0\ , \\
& x_{AC}+y_{AC}+x_{BC}+y_{BC}-x_{CT}-y_{CT} = 0, \\
& x_{BD}+y_{BD}-x_{DT}-y_{DT} = 0\ , \\
& 0 \le x_{SA}, x_{SB}, x_{AC}, x_{BC}, x_{BD}, x_{CT}, x_{DT}, y_{SA}, y_{SB}, y_{AC}, y_{BC}, y_{BD}, y_{CT}, y_{DT} \le 1\ .
\end{aligned}$$

This linear program can be interpreted as a minimal cost flow problem.

Where is the connection between the linear program and the problem to find two candidate routes from s to t?

Theorem 23.4 *If the linear program $LP_{2-ShortPaths}$ has an optimal solution then it has also an optimal solution (x,y) with the following properties.*
There are disjoint sets $E_1, E_2, E_3 \subseteq E$ with
(i) $E_1 \cap E_2 = \emptyset$, $E_1 \cap E_3 = \emptyset$ and $E_2 \cap E_3 = \emptyset$,
(ii) $x_e = 1$, $y_e = 0$ for all $e \in E_1 \cup E_2$,
(iii) $x_e = 1$, $y_e = 1$ for all $e \in E_3$,
(iv) $x_e = 0$, $y_e = 0$ for all $e \notin E_1 \cup E_2 \cup E_3$.
(v) $E_1 \cup E_3$ represents a path P_1 from s to t and $E_2 \cup E_3$ represents a path P_2 from s to t. E_3 is the set of arcs used by both paths.

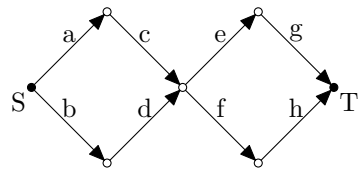

Figure 23.5 An example for a non-unique decomposition in two paths.

(vi) No other pair (Q_1, Q_2) of paths is better than (P_1, P_2), i.e.,

$$w(P_1) + w(P_2) + \varepsilon \cdot w(P_1 \cap P_2) \leq w(Q_1) + w(Q_2) + \varepsilon \cdot w(Q_1 \cap Q_2) \ , \quad \textit{for all pairs } (Q_1, Q_2) \ .$$

That means the sum of the lengths of P_1 and P_2 plus a penalty for arcs used twice is minimal.

We conclude with some remarks.

- For each arc e there are two variables x_e and y_e. This can be interpreted as a street with a **normal lane** and an additional **passing lane**. Using the passing lane is more expensive than using the normal lane. If a solution uses an arc only once, it takes the cheaper normal lane. But if a solution uses an arc twice, it has to take both the normal and the passing lane.

- The decomposition of the solution (x, y) into two paths from the starting node to the target node is in most cases not unique. With the arcs $a, b, \ldots, g, h$ in Figure 23.5 we can build two different pairs of paths from S to T, namely $(a - c - e - g,\ b - d - f - h)$ and $(a - c - f - h,\ b - d - e - g)$. Both pairs are equi-optimal in the sense of Theorem 23.4. So the user has the chance to choose between them according to other criteria.

- The penalty method and the LP-penalty method generally lead to different results. The penalty method computes the best single solution and a suitable alternative. The LP-penalty method computes a pair of good solutions with relatively small overlap. Figure 23.4 shows that this pair not necessarily contains the best single solution. The shortest path from S to T is $P_1 = S$–B–C–T with length 3. For all $\varepsilon > 0.1$ the ε-penalty solution is $P_2 = S$–A–C–T. The path pair (P_1, P_2) has a total lengths of 6.1 and a shared length of 1.0. But for $\varepsilon > 0.2$ the LP-Penalty method produces the pair $(P_2, P_3) = (S$–A–C–T, S–B–D–$T)$ with a total length of 6.3 and a shared length of zero.

- Finding k candidate routes for some larger number $k > 2$ is possible, if we introduce k variables $x_e^0, x_e^1, \ldots, x_e^{k-1}$ for each arc e and set the supply of s and

the demand of t to k. As objective function we can use for instance

$$\min f(x^0, \ldots, x^{k-1}) := \sum_{e \in E} \sum_{j=0}^{k-1} (1 + j \cdot \varepsilon) \cdot w(e) \cdot x_e^j$$

or

$$\min f(x^0, \ldots, x^{k-1}) := \sum_{e \in E} \sum_{j=0}^{k-1} (1 + \varepsilon)^j \cdot w(e) \cdot x_e^j \ .$$

- The LP-penalty method does not only work for shortest path problems. It can be generalised to arbitrary problems solvable by linear programming.
- Furthermore an analogous method – the *Integer Linear Programming Penalty Method* – can be applied to problems solvable by integer linear programming.

23.2.4. Penalty method with heuristics

In Subsection 23.2.2 we discussed the penalty method in combination with exact solving algorithms (e.g. Dijkstra-algorithm or dynamic programming for the shortest path problem). But also in case of heuristics (instead of exact algorithms) the penalty method can be used to find multiple candidate solutions.

Example 23.9 A well known heuristic for the TSP-problem is local search with 2-exchange steps (cp. Subsection 23.2.1).

PENALTY-METHOD-FOR-THE-TSP-PROBLEM-WITH-2-EXCHANGE-HEURISTIC

1 apply the 2-exchange heuristic to the unpunished problem getting a locally (but not necessarily globally) optimal solution T
2 punish the edges belonging to T by multiplying their lengths with $1 + \varepsilon$
3 apply the 2-exchange heuristic to the punished problem getting an alternative solution $T\varepsilon$
4 compute the unmodified length of $T\varepsilon$
5 the pair $(T, T\varepsilon)$ is the output

Question: Which penalty parameter $\varepsilon \geq 0$ should be used to minimise the travel time of the fastest route?

An experiment analogous to the one described in Example 23.5 was executed for TSP instances with 25 random cities in the unit square. Figure 23.6 shows the scaled averages for $\varepsilon_0 = 0.000$, $\varepsilon_1 = 0.025$, $\ldots$, $\varepsilon_{30} = 0.750$. So, the expected quality $\phi\varepsilon$ of the solution pairs is (again) unimodal in the penalty factor ε. That means that $\phi\varepsilon$ first decreases and then increases for growing ε. In this example $\varepsilon^* \approx 0.175$ is the optimal penalty parameter.

In further experiments it was observed that the optimal penalty parameter ε^* is decreasing in the problem size.

Exercises

23.2-1 The following programming exercise on the Travelling Salesperson Problem helps to get a feeling for the great variety of local optima. Generate 200 random

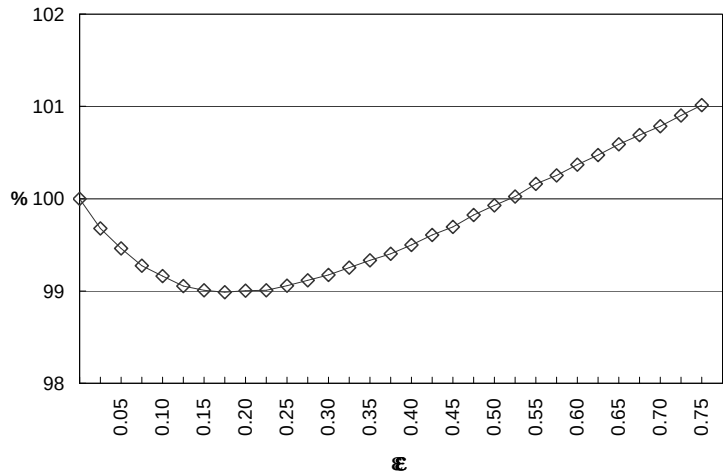

Figure 23.6 $\overline{\phi}_{\varepsilon_i}$ for $\varepsilon_0 = 0,\ \varepsilon_1 = 0.025,\ \ldots,\ \varepsilon_{30} = 0.750$ on 25×25 grids.

points in the 2-dimensional unit-square. Compute distances with respect to the Euclidean metric. Make 100 runs of local search with random starting tours and 2-exchanges. Count how many different local minima have been found.

23.2-2 Enter the same key words into different internet search engines. Compare the hit lists and their diversities.

23.2-3 Formulate the Travelling Salesperson Problem as a $\sum$-type problem.

23.2-4 Proof the assertion of the remark on page 1107.

23.2-5 How does the penalty function $p(e)$ look like in case of additive penalties like in Example 23.2?

23.2-6 Prove the properties (1) and (2) on page 1108.

23.2-7 Apply the DIVIDE AND COVER algorithm (page 1108) to the shortest path problem in Figure 23.2 with starting node S and target node T. Set $w(e) =$ length of e for each road section, and $p(e) =$ length of e for the road sections belonging to the shortest path P_D via S - A - C - D - F - T and $p(e) = 0$ for all other sections. So, the penalty value of a whole path is the length of this part shared with P_D.

23.2-8 Find a penalty parameter $\varepsilon > 0$ for Example 23.6 such that the first setting (5) produces three different paths but the second setting (5*) only two different paths for $k = 3$.

23.3. More algorithms for interactive problem solving

There are many other settings where a human controller has access to computer-generated candidate solutions. This section lists four important cases and concludes with a discussion of miscellaneous stuff.

23.3.1. Anytime algorithms

In an anytime-setting the computer starts to work on a problem, and almost from the very first moment on candidate solutions (the best ones found so far) are shown on the monitor. Of course, the early outputs in such a process are often only preliminary and approximate solutions – without guarantee of optimality and far from perfect.

An example: Iterative deepening performs multiple depth-limited searches – gradually increasing the depth limit on each iteration of the search. Assume that the task is to seek good solutions in a large rooted tree $T = (V, E)$. Let $f : V \to \mathbb{R}$ be the function which is to be maximised. Let V_d be the set of all nodes in the tree at distance d from root.

ITERATIVE-DEEPENING-TREE-SEARCH(T, f)

```
1  Opt ← f(root)
2  d ← 1
3  while d < ∞
4        do Determine maximum Max_d of f on V_d
5        if Max_d > Opt
6           then Opt ← Max_d
7        d ← d + 1
```

All the time the currently best solution (Opt) is shown on the monitor. The operator may stop at any moment.

Iterative deepening is not only interesting for HCI but has also many applications in fully automatic computing. A prominent example is game tree search: In tournament chess a program has a fixed amount of time for 40 moves, and iterative deepening is the key instrument to find a balanced distribution of time on the single alpha-beta searches.

Another frequent anytime scenario is repeated application of a heuristic. Let $f : A \to \mathbb{R}$ be some complicated function for which elements with large function values are searched. Let H be a probabilistic heuristic that returns a candidate solution for this maximisation problem (A, f). For instance, H may be local search or some other sort of hill-climbing procedure. H is applied again and again in independent runs, and all the time the best solution found so far is shown.

A third anytime application is in Monte Carlo simulations, for instance in Monte Carlo integration. A static approach would take objective values at a prescribed number of random points (1,000 or so) and give the average value in the output. However, already the intermediate average values (after 1, 2, 3 etc. data points – or after each block of 10 or 50 points) may give early indications in which region the final result might fall and whether it really makes sense to execute all the many runs. Additional display of variances and frequencies of outliers gives further information for the decision when best to stop the Monte Carlo run.

In human-computer systems anytime algorithms help also in the following way: during the ongoing process of computing the human may already evaluate and compare preliminary candidate solutions.

23.3.2. Interactive evolution and generative design

Genetic Algorithms are search algorithms based on the mechanics of natural selection and natural genetics. Instead of single solutions whole populations of solutions are manipulated. Genetic Algorithms are often applied to large and difficult problems where traditional optimisation techniques fall short.

Interactive evolution is an evolutionary algorithm that needs human interaction. In interactive evolution, the user selects one or more individual(s) of the current population which survive(s) and reproduce(s) (with mutations) to constitute the new generation. So, in interactive evolution the user plays the role of an objective function and thus has a rather active role in the search process.

In fields like art, architecture, and photo processing (including the design of phantom photos) *Generative Design* is used as a special form of interactive evolution. In Generative Design *all* solutions of the current generation are shown simultaneously on the screen. Here typically "all" means some small number N between 4 and 20. Think of photo processing as an example, where the user selects modified contrast, brightness, colour intensities, and sharpness. The user inspects the current candidate realizations, and by a single mouse click marks the one which he likes most. All other solutions are deleted, and N mutants of the marked one are generated. The process is repeated (open end) until the user is happy with the outcome. For people without practical experience in generative design it may sound unbelievable, but even from poor starting solutions it takes the process often only a few iterations to come to acceptable outcomes.

23.3.3. Successive fixing

Many problems are high-dimensional, having lots of parameters to adjust. If sets of good solutions in such a problem are generated by repeated probabilistic heuristics, the following interactive multi-stage procedure may be applied: First of all several heuristic solutions are generated and inspected by a human expert. This human especially looks for "typical" pattern in the solutions and "fixes" them. Then more heuristic solutions are generated under the side condition that they all contain the fixed parts. The human inspects again and fixes more parts. The process is repeated until finally everything is fix, resulting in one specific (and hopefully good) solution.

23.3.4. Interactive multicriteria decision making

In multicriteria decision making not only one but two or more objective functions are given. The task is to find admissible solutions which are as good as possible with respect to all these objectives. Typically, the objectives are more or less contradictory, excluding the existence of a unanimous optimum. Helpful is the concept of "efficient solutions", with the following definition: For an efficient solution there exists no other solution which is better with respect to at least one objective and not worse with respect to all the others.

A standard first step in multicriteria decision making is to compute the set of all efficient solutions. In the bicriteria case the "efficient frontier" can be visualized in a 2-dimensional diagram, giving the human controller a good overview of what is

possible.

23.3.5. Miscellaneous

- *Graphical Visualisation of Computer Solutions*
  It is not enough that a computer generates good candidate solutions. The results also have to be visualized in appropriate ways. In case of a single solution important parts and features have to be highlighted. And, even more important, in case of concurring solutions their differences and specialities have to be stressed.

- *Permanent Computer Runs with Short Intermediate Human Control*
  A nickname for this is "1+23h mode", coming from the following picture: Each day the human sits in front of the computer for one hour only. In this hour he looks at the computer results from the previous 23 hours, interacts with the machine and also briefs the computer what to do in the next 23 hours. So, the human invests only a small portion of his time while the computer is running permanently.

  An impressive example comes from correspondence chess. Computer help is officially permitted. Most top players have one or several machines running all around the clock, analysing the most critical positions and lines of play. The human players collect these computer results and analyse only shortly per day.

- *Unexpected Errors and Numerical Instabilities*
  "Every software has errors!" This rule of thumb is often forgotten. People too often simply believe what the monitor or the description of a software product promises. However, running independent programs for the very same task (with a unique optimal solution) will result in different outputs unexpectedly often. Also numerical stability is not for free. Different programs for the same problem may lead to different results, due to rounding errors. Such problems may be recognised by applying independent programs.

  Of course, also hardware has (physical) errors, especially in times of ongoing miniaturisation. So, in crucial situations it is a good strategy to run an identical program on fully independent machines - best of all operated by independent human operators.

Exercises

23.3-1 For a Travelling Salesperson Problem with 200 random points (x_i, y_i) in the unit square $[0,1] \times [0,1]$ and Euclidean distances, generate 100 locally optimal solutions (with 2-exchanges, see Subsection 23.2.1) and count which edges occur how often in these 100 solutions. Define some threshold K (for instance $K = 30$) and fix all edges which are in at least K of the solutions. Generate another 100 local optima, without allowing the fixed edges to be exchanged. Repeat until convergence and compare the final result with typical local optima from the first series.

Chapter Notes

In the technical report [**?**] lots of experiments on the penalty method for various sum type problems, dimensions, failure widths and probabilities are described and analysed. The proof of Theorem 23.3 was originally given in [**?**]) . In e-commerce multiple-choice systems are often called "Recommender Systems" [10], having in mind customers for whom interesting products have to be listed. Understandably, commercial search engines and e-companies keep their shortlisting strategies secret.

A good class book on Genetic Algorithms is [8]. Interactive Evolution and Generative Design are described in [6]. There is a lot of literature on multicriteria decision making, one of the standard books being [7].

In the book [2] the story of 3-Hirn and its successes in tournament chess is told. The final match between "3-Hirn" and GM Yussupov is described in [3]. [4] gives more general information on improved game play by multiple computer hints. In [5] several good k-best realizations of iterative deepening in game tree search are exhibited and discussed. Screenshots of these realizations can be inspected at http://www.minet.uni-jena.de/www/fakultaet/iam/personen/k-best.html. [9] describes the technical background of advanced programs for playing chess and other games.

There is a nice online repository, run by M. Zuker and D.H. Turner at http://www.bioinfo.rpi.edu/applications/mfold/. The user may enter for instance RNA-strings, and in realtime alternative foldings for these strings are generated. Amongst other data the user may enter parameters for "maximum number of computed foldings" (default = 50) and "percent suboptimality number" (default = 5 %).

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993. 1113

[2] I. Althöfer. *13 Jahre 3-Hirn.* Published by the author, 1998. 1121

[3] I. Althöfer. List-3-Hirn vs. grandmaster Yussupov – report on a very experimental match. *ICCA Journal*, 21:52–60 and 131–134, 1998. 1121

[4] I. Althöfer. Improved game play by multiple computer hints. *Theoretical Computer Science*, 313:315–324, 2004. 1121

[5] I. Althöfer, J. de Koning, J. Lieberum, S. Meyer-Kahlen, T. Rolle, J. Sameith. Five visualisations of the k-best mode. *ICCA Journal*, 26:182–189, 2003. 1121

[6] W. Banzhaf. Interactive evolution. In T. Back, D. B. Fogel, Z. Michalewicz, T. Baeck (Eds.) *Handbook of Evolutionary Computation.* IOP Press, 1997. 1121

[7] T. Gal, T. Stewart, T. Hanne. (Eds.). *Multicriteria Decision Making.* Kluwer Academic Publisher, 1999. 1121

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989. 1121

[9] E. A. Heinz. *Algorithmic Enhancements and Experiments at High Search Depths.* Vieweg Verlag, Series on Computational Intelligence, 2000. 1103, 1121

[10] P. Resnick, H. R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56–58, 1997. 1121

This bibliography is made by HBibTEX. First key of the sorting is the name of the authors (first author, second author etc.), second key is the year of publication, third key is the title of the document.

Underlying shows that the electronic version of the bibliography on the homepage of the book contains a link to the corresponding address.

Index

This index uses the following conventions. Numbers are alphabetised as if spelled out; for example, “2-3-4-tree" is indexed as if were “two-three-four-tree". When an entry refers to a place other than the main text, the page number is followed by a tag: *exe* for exercise, *exa* for example, *fig* for figure, *pr* for problem and *fn* for footnote.

The numbers of pages containing a definition are printed in *italic* font, e.g.

time complexity, *583*.

Name Index

This index uses the following conventions. If we know the full name of a cited person, then we print it. If the cited person is not living, and we know the correct data, then we print also the year of her/his birth and death.