4 Approximation Schemes

In the previous two chapters we got to know several more or less good approximation algorithms. In many cases, we still do not know whether a better solution can be obtained. Would it be possible to achieve better approximation guarantees by spending more runtime? If so, the error should approach 0 as the runtime of the algorithm goes to infinity. However, the increase in the runtime to obtain a better solution should not be too dramatic. These considerations led to the notion of *approximation schemes*.

Definition 4.1 Let Π be an optimization problem and let A be an approximation algorithm for Π.

(a) A is a *polynomial time approximation scheme* (PAS) for Π if for every instance I and every constant $\epsilon \in (0,1)$, A computes in time $O(poly(|I|))$ a feasible solution for I with a relative error of at most ϵ.

(b) A is a *fully polynomial time approximation scheme* (FPAS) for Π if A is a PAS and has a runtime of $O(poly(|I|, 1/\epsilon))$.

Also the short forms PTAS and FPTAS can be frequently found in the literature. For a PAS we demanded a runtime that is only polynomial in $|I|$. For ϵ we did not formulate any requirements, i.e. ϵ is treated like a constant. Runtimes of the form $O(|I|^{1/\epsilon})$ are therefore allowed, whereas runtimes of the form $O((1/\epsilon)^{|I|})$ are not allowed. The notion of an FPAS is therefore more restrictive than the notion of a PTAS.

The use of ϵ only extends the input for an FPAS by $\Theta(\log(1/\epsilon))$ bits. Nevertheless, we demand that the runtime of the FPAS has to be polynomial in $1/\epsilon$ and not polynomial in $\log(1/\epsilon)$. If we demanded that the runtime of an FPAS had to be $O(poly(|I|, \log(1/\epsilon)))$, then the existence of such an FPAS for an optimization problem whose decision variant is **NP**-complete and for which the values can be represented with $O(poly(|I|))$ bits would imply that $\mathbf{P} = \mathbf{NP}$.

In principle, two ways can be imagined about how approximation schemes work:

(a) Depending on ϵ, only one (or a small number of) feasible solution(s) is determined (we will get to know such an algorithm in this chapter)

(b) after a feasible solution has been determined, this is improved by new feasible solutions until the allowed runtime has been used (such algorithms will be presented in Chapter 7).

Note that according to Remark 3.2 we have that in a minimization problem, $1 + \epsilon_A = \rho_A$, and in a maximization problem, $1 - \epsilon_A = 1/\rho_A$. Hence, it follows from the definition of an FPAS for an optimization problem that one can find solutions with a constant approximation ratio, say 5/4, in polynomial time. In the case of a minimization problem we would have to select the error as $\epsilon^* = 1/4$ and in the case of a maximization problem we would have to select the error as $\epsilon^* = 1/5$.

4.1 A pseudo-polynomial exact algorithm

We showed in Section 2.4 that one cannot approximate the knapsack problem up to some fixed absolute error, unless $\mathbf{P} = \mathbf{NP}$. We will see now that the situation drastically improves when using the relative error. But first we present an exact algorithm for KNAPSACK.

At the beginning of Chapter 1 we already briefly mentioned that the runtime of an algorithm may depend on the numbers used in the input. Let $P_{\max} = \max\{p_j : j \in \{1, \ldots, n\}\}$ be the largest price in the instance $I = \langle W, vol, p, C\rangle$. Since there are n objects and every object can be assumed (w.l.o.g.) to fit into the knapsack,

$$P_{\max} \leq \mathrm{OPT}(I) \leq n \cdot P_{\max}\,. \tag{1}$$

Definition 4.2 Let $\mathrm{maxnr}(I)$ be the absolute value of the largest number in I. An algorithm is called *pseudo-polynomial* if its runtime is $O(poly(|I|, \mathrm{maxnr}(I)))$.

For KNAPSACK it holds according to (1) that $\mathrm{OPT}(I) \leq n \cdot \mathrm{maxnr}(I)$. For every $j \in \{0, 1, \ldots, n\}$ and $\alpha \in \mathbb{Z}$ let $F_j(\alpha)$ be the *smallest* volume required to achieve a total value of *at least* α when choosing only among the first j objects. Then it holds:

$$\begin{array}{lll} (1) & \alpha \leq 0,\ j \in \{0, \ldots, n\}: & F_j(\alpha) = 0 \\ (2) & \alpha \geq 1,\ j = 0: & F_j(\alpha) = \infty \\ (3) & \alpha \geq 1,\ j \in \{0, \ldots, n\}: & F_j(\alpha) = \min\{F_{j-1}(\alpha - p_j) + \mathrm{vol}(j), F_{j-1}(\alpha)\} \end{array}$$

By ∞ we denote that it is impossible to achieve the desired value α.

(1) says that the empty knapsack has the value 0, and (2) says that one cannot achieve a value different from 0 with an empty knapsack.

The recursive expression in (3), which is also known as *Bellmann's equation*, says the following: if the smallest necessary volume for a value of at least α has been calculated for the objects $1, \ldots, j-1$ and then the object j with $\mathrm{vol}(j)$ and value p_j is added, then either nothing changes concerning a smallest possible volume for α, or the value $\alpha - p_j$ achieved with the objects $1, \ldots, j-1$ together with object j results in a smaller volume to achieve a value of at least α. Since the equation covers both possible cases to obtain the smallest possible volume for j objects (either object j belongs or does not belong to a solution of minimum volume), it is not difficult to prove by induction that there cannot be a selection of objects that can achieve a value of α with a lower volume then achieved by the $F_j(\alpha)$'s. Hence, we are searching for the largest α for which $F_n(\alpha)$ still fits into a knapsack with capacity C, i.e. $\max\{\alpha :\ F_n(\alpha) \leq C\} = \mathrm{OPT}(I)$.

The rules above yield the following algorithm for the computation of $F_j(\alpha)$. It is a typical example of a *dynamic program*.

```
Algorithm DynKnapsack

  α = 0
  repeat
    α = α + 1     (*)
    for j = 1 to n do
      F_j(α) = min{F_{j-1}(α - p_j) + vol(j), F_{j-1}(α)}
  until C < F_n(α)
  output α - 1
```

If one organizes the $F_j(\alpha)$'s in a table, the table is filled downwards in a row-wise fashion Note that it is important for the numbers to be integers. This is the reason for the "+1" in line $(*)$. The correctness of the following theorem results from the discussion above.

Theorem 4.3 DYNKNAPSACK *computes on input I the value* $\mathrm{OPT}(I)$ *in pseudo-polynomial time* $O(n \cdot \mathrm{OPT}(I)) = O(n^2 \cdot P_{\max}) = O(n^2 \cdot \mathrm{maxnr}(I))$.

Since $\mathrm{OPT}(I) = \Theta(2^{|I|})$, the runtime is $O(n \cdot 2^{|I|})$, i.e. it is exponential in the length of the input. It is easy to extend DYNKNAPSACK in a way that not only the value of an optimal solution but also the optimal solution itself can be output. This will be an assignment. Thus, if the domain for the Knapsack problem is restricted to instances I with $\mathrm{maxnr}(I) \leq p(|I|)$ for some polynomial p, then the Knapsack problem can be solved exactly in polynomial time.

4.2 A fully polynomial approximation scheme

With the help of the pseudo-polynomial algorithm presented above we will construct an approximation scheme that computes a feasible solution that depends on ϵ.

Since the largest price influences the runtime of DynKnapsack, one should simply reduce all prices by some fixed k, or in other words, replace all prices p_j by $\lfloor p_j/k \rfloor$, solve the new problem I_{red} and take its solution as the solution for the original problem. This results in the following algorithm:

Algorithm AR_k

(1) reduce all prices by k
(2) compute with the help of DynKnapsack an optimal solution R_k for the reduced prices
(3) output R_k

The problem with the algorithm is that the use of $\lfloor \cdot \rfloor$ produces a (relative) error. How large can this error be? Note that the relative error cannot be bigger than 1, since the Knapsack problem is a maximization problem.

Theorem 4.4 *AR_k has on input I a relative error of $\epsilon_{\text{AR}_k}(I) \leq k \cdot n/P_{\max}$ and a runtime of $O(n^2 \cdot P_{\max}/k)$.*

Proof. Let R^* be the index set of an optimal solution for I and let R_k be the index set computed for I_{red} by the algorithm AR_k. Since R_k is an optimal solution for I_{red},

$$\text{OPT}(I_{\text{red}}) \geq \sum_{j \in R^*} \lfloor \frac{p_j}{k} \rfloor .$$

Furthermore, R_k is a feasible solution for I, because we did not change the volumes of the articles and the capacity of the knapsack. It holds that

$$\begin{aligned}
\text{AR}_k(I) &= \sum_{j \in R_k} p_j \geq \sum_{j \in R_k} k \cdot \lfloor \frac{p_j}{k} \rfloor = k \cdot \text{OPT}(I_{\text{red}}) \\
&\geq k \sum_{j \in R^*} \lfloor \frac{p_j}{k} \rfloor \overset{(*)}{\geq} k \sum_{j \in R^*} \left(\frac{p_j}{k} - 1 \right) = \sum_{j \in R^*} (p_j - k) \\
&\geq \text{OPT}(I) - k \cdot |R^*| \geq \text{OPT}(I) - kn .
\end{aligned}$$

Note that for a large k the expressions after $(*)$ can be negative. It would be more precise if one continues with $\max\{\cdot, 0\}$ instead. However, for our purposes the bound will be sufficient.

With this inequality we obtain according to the definition of the relative error and with the help of Inequality (1) that

$$\epsilon_{\text{AR}_k}(I) = \frac{|\text{AR}_k(I) - \text{OPT}(I)|}{\text{OPT}(I)} \leq \frac{k \cdot n}{\text{OPT}(I)} \leq \frac{k \cdot n}{P_{\max}} .$$

In order to solve the reduced problem in an exact way, we need $O(n^2 \cdot P_{\max}/k)$ steps according to Theorem 4.3. □

Note that for a large k the theorem can be trivial, because the relative error of a maximization problem cannot be bigger than 1.

In order to obtain an FPAS we have to show that in time $O(poly(|I|, 1/\epsilon))$ every $\epsilon \in (0, 1)$ can be reached as the relative error. Since the parameters n and $P_{\max}$ are already given by the input I, we can only influence ϵ by the choice of k.

Algorithm FPASKNAPSACK(I, ϵ)

determine from I the parameters n and $P_{\max}$
$k = \epsilon \cdot P_{\max}/n$
run AR_k on I and output its solution

Obviously, Theorem 4.4 yields the following result:

Theorem 4.5 FPASKNAPSACK *is an FPAS for the Knapsack problem with runtime* $O(n^3/\epsilon)$.

There is even an FPAS for KNAPSACK with a runtime of $O(n \log(1/\epsilon) + 1/\epsilon^4)$ (see [IK75] or Chapter 9 of [Hoc96]).

4.3 An impossibility result for approximation schemes

It is not difficult to see that there cannot be an approximation algorithm for graph coloring problems with an approximation ratio of less than 4/3. From this it follows that there cannot be a PAS or FPAS for these problems. One can generalize this to the following result.

Theorem 4.6 *Let* Π *be an optimization problem and suppose that for some fixed* $k \in \mathbb{N}$ *the question "Given an input* I*, is* $\mathrm{OPT}(I) \le k$*?" (if* Π *is a maximization problem) resp. "Given an input* I*, is* $\mathrm{OPT}(I) \ge k$*?" (if* Π *is a minimization problem) is* **NP***-complete. If there is an (F)PAS for* Π*, then* **P** $=$ **NP**.

We will further generalize this theorem in order to prove an even stronger impossibility result for fully polynomial approximation schemes.

In Section 4.1 we showed that the restriction $\mathrm{maxnr}(I) = p(|I|)$ for *any* polynomial p allows to solve an instance I of KNAPSACK in polynomial time. On the other hand, we can use the proof of Theorem 3.14 to reduce HAMILTON to the decision variant of the TSP by setting $r = 1$. The largest edge weight occurring in the instance I_G constructed in this reduction is at most 2, and hence $\mathrm{maxnr}(I_G) \le 2$ if we ignore the n names of the nodes. That is, the decision variant for the TSP for which all edge weights are at most 2 is still a difficult problem. This leads to the separation into strongly **NP**-complete and weakly **NP**-complete problems.

Definition 4.7 An **NP**-complete decision problem L is called *strongly* **NP***-complete* if there is a polynomial p so that $L_p = \{x : \ x \in L \text{ and } \mathrm{maxnr}(x) \le p(|x|)\}$ is still **NP**-complete. Is there no such polynomial, then we call L *weakly* **NP***-complete*.

Equivalent to this definition is: the **NP**-complete decision problem L is strongly **NP**-complete if there is no pseudo-polynomial algorithm for L unless **P** $=$ **NP**.

Example 4.8

- The decision problems HAMILTON and CLIQUE are strongly **NP**-complete, since no big numbers are used in them and therefore a polynomial of the form $p(n) = \Theta(n)$ suffices.
- It follows from our comments above that the decision variant of the TSP is strongly **NP**-complete. Since the constructed graphs only have edges of weight 1 or 2, the length of every path (u, v, w) along two edges is at least 3, i.e. at least as long as the direct edge $\{u, w\}$ between the two endpoints of the path. This implies that the triangle inequality is fulfilled. Hence, also the decision variant of the ΔTSP problem is strongly **NP**-complete.

- Knapsack is weakly **NP**-complete, as we saw in Section 4.1, because its optimization variant has a pseudo-polynomial algorithm that can easily be used also for the decision problem.

The next theorem shows that there is a tight relationship between strong **NP**-completeness and the impossibility of constructing an FPAS. An important ingredient in the proof is that the values of all feasible solutions are natural numbers.

Theorem 4.9 *Let Π be an optimization problem. If there is a polynomial $p(x_1, x_2)$ so that for all instances I, $\mathrm{OPT}(I) \le p(|I|, \mathrm{maxnr}(I))$, then it follows from the existence of an FPAS for Π that there is a pseudo-polynomial exact algorithm for Π.*

Proof. Let A be an FPAS for Π. Since the values of feasible solutions can only be natural numbers and we assume that $\mathrm{OPT}(I) \le p(|I|, \mathrm{maxnr}(I))$, it suffices to choose an ϵ for A that is so small that there cannot be a non-optimal solution with error at most ϵ and therefore A has to output an optimal solution.

We give now a pseudo-polynomial exact algorithm for Π. For the instance I set

$$\epsilon^* = \frac{1}{p(|I|, \mathrm{maxnr}(I)) + 1}$$

and start A on input I and ϵ^*.

We show that the output of A for this ϵ^* is an optimal solution for I: according to Definition 4.1 a feasible solution for I is found in time $O(poly(|I|, p(|I|, \mathrm{maxnr}(I))))$ with a relative error of

$$\epsilon_A(I, \epsilon^*) = \frac{|\mathrm{OPT}(I) - A(I, \epsilon^*)|}{\mathrm{OPT}(I)} \le \epsilon^* .$$

Hence,

$$|\mathrm{OPT}(I) - A(I, \epsilon^*)| \le \epsilon^* \cdot \mathrm{OPT}(I) = \frac{\mathrm{OPT}(I)}{p(|I|, \mathrm{maxnr}(I)) + 1} < 1 ,$$

because $\mathrm{OPT}(I) \le p(|I|, \mathrm{maxnr}(I))$. Since the values of all feasible solutions are natural numbers, it follows that $|\mathrm{OPT}(I) - A(I, \epsilon^*)| = 0$, and therefore $A(I, \epsilon^*) = \mathrm{OPT}(I)$. $\square$

This theorem reverses the approach for the development of an FPAS for Knapsack. There we gave an FPAS with the help of an exact pseudo-polynomial algorithm, and here we constructed out of an FPAS a pseudo-polynomial algorithm. In order to ensure that the proof is fully understood, one should try to find the place in it where it does not suffice for A to be a PAS.

If $\mathrm{maxnr}(I)$ is bounded from above by a polynomial, then this proof provides a polynomial runtime for an exact algorithm, which leads us to the following statement.

Corollary 4.10 *If there is an FPAS for an optimization variant of a strongly* **NP***-complete problem, then* $\mathbf{P} = \mathbf{NP}$.

That is, unless $\mathbf{P} = \mathbf{NP}$ there cannot be fully polynomial approximation schemes for Clique, graph coloring problems and many other optimization problems. Since the general TSP does not have an approximation algorithm with a constant approximation ratio (Theorem 3.14), also the general TSP does not have an FPAS or PAS.

But how about the ΔTSP, that can be approximated with the help of Christofides' algorithm with a ratio of 3/2? Unfortunately, we have seen in Example 4.8 that the decision variant of the metric TSP is stongly **NP**-complete, and therefore there cannot be an FPAS for the ΔTSP.

EUCLIDTSP is the restriction of the ΔTSP to geometric problems, i.e. graphs in which the distance is always given by a (rounded) Euclidean norm. A breakthrough result by Arora [Aro98] shows that there is a polynomial approximation scheme (i.e. a PAS) for EUCLIDTSP with a runtime of $O(n^3(\log n)^{O(1/\epsilon)})$. That is, 20 years after Christofides' algorithm it was possible to reduce the approximation ratio for EUCLIDTSP from $3/2$ to $1+\epsilon$ for any constant $\epsilon > 0$. A detailed description of the algorithm can be found in Chapter 13 of [MPS98].

However, the ΔTSP provably has no PAS unless $\mathbf{P} = \mathbf{NP}$ [ALM+98]. That is, it is provably more difficult than the EUCLIDTSP. But one of the big problems in this area is still whether there is an approximation algorithm for the ΔTSP with a ratio of below $3/2$.

References

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45:501–555, 1998.

[Aro98] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45:753–782, 1998.

[Hoc96] D.S. Hochbaum (ed.). *Approximation Algorithms for NP-Hard Problems.* PWS, Boston, 1996.

[IK75] O.H. Ibarra and C.E. Kim. Fast approximation for the knapsack and sum of subset problems. *Journal of the ACM*, 22:463–468, 1975.

[MPS98] E.W. Mayr, H.J. Prömel, and A. Steger. *Lectures on Proof Verification and Approximation Algorithms.* Springer, Berlin, 1998.