4. The Subset Sum Problem

Given a set $N = \{1,\dots,n\}$ of n items with positive integer weights $w_1,\dots,w_n$ and a capacity c, the *subset sum problem* (SSP) is to find a subset of N such that the corresponding total weight is maximized without exceeding the capacity c. Recall the formal definition as introduced in Section 1.2:

$$\text{(SSP)} \qquad \text{maximize} \quad \sum_{j=1}^{n} w_j x_j \tag{4.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \le c, \tag{4.2}$$

$$x_j \in \{0,1\}, \quad j = 1,\dots,n. \tag{4.3}$$

We will assume that assumptions (1.14) to (1.16) as defined for (KP) also hold for (SSP). This means that every item j fits into the knapsack, i.e.,

$$w_j \le c, \quad j = 1,\dots,n, \tag{4.4}$$

and that the overall weight sum of the items exceeds c, i.e.,

$$\sum_{j=1}^{n} w_j > c. \tag{4.5}$$

Without loss of generality we assume that all weights are positive, i.e.,

$$w_j > 0, \quad j = 1,\dots,n, \tag{4.6}$$

since otherwise we may use the transformation from Section 1.4 to achieve this assumption.

We will often identify the items with their corresponding weights. (SSP) can be considered as a special case of the knapsack problem arising when the profit and the weight associated with each item are identical. (SSP) has numerous applications: Solutions of subset problems can be used for designing better lower bounds for scheduling problems (see Guéret and Prins [200] and Hoogeveen et al. [237]).

H. Kellerer et al., *Knapsack Problems*

The constraints of 0-1 integer programs could be tightened by solving subset sum problems with some additional constraints (see e.g. Dietrich and Escudero [105] and Escudero, Martello and Toth [134]), and it appears as subproblem in numerous combinatorial problems (see e.g. Pisinger [386]).

Several authors considered the idea of applying the subset sum problem as a subprocedure in a bin packing algorithm. This means that items are packed into bins filling one bin at a time, each as much as possible. Recently, Caprara and Pferschy [68] managed to show that the resulting heuristic has a worst-case performance ratio between 1.6067... and 1.6210.... Variants of this approach were considered in Caprara and Pferschy [67].

(SSP) formulated as a *decision problem* asks whether there exists a subset of N such that the corresponding weights add up exactly to the capacity c. To distinguish it from the optimization problem we denote it as SSP-DECISION (see also Appendix A). The decision problem is of particular interest in cryptography since SSP-DECISION with unique solutions corresponds to a secret message to be transmitted. We will go into details with cryptosystems based on subset sum problems in Section 15.6.

Although (SSP) is a special case of (KP) it is still $\mathcal{NP}$-hard [164] as will be shown in Appendix A. Clearly, (SSP) can be solved to optimality in pseudopolynomial time by the dynamic programming algorithms described in Section 2.3. But due to the simple structure of (SSP) adapted algorithms can have much better behavior. For (SSP) all upper bounds based on some kind of continuous relaxation as they will be described in Section 5.1.1, give the trivial bound $U = c$. Thus, although (SSP) in principle can be solved by every branch-and-bound algorithm for (KP), the lack of tight bounds may imply an unacceptably large computational effort. Therefore, it is worth constructing algorithms designed specially for (SSP). Also, owing to the lack of tight bounds, it may be necessary to apply heuristic techniques to obtain a reasonable solution within a limited time.

In this chapter exact and approximation algorithms for the (SSP) are investigated. We will start the treatment of (SSP) in Section 4.1 dealing with different dynamic programming algorithms: If the coefficients are not too large, dynamic programming algorithms are generally able to solve subset sum problems even when the decision problem SSP-DECISION has no solution. In Section 4.2.1 upper bounds tighter than the trivial bound $U = c$ are considered. In Section 4.2 we will present some hybrid algorithms which combine branch-and-bound with dynamic programming. Section 4.3 shows how large-sized instances can be solved by defining a core problem, and Section 4.4 gives a computational comparison of the exact algorithms presented. Section 4.5 deals with polynomial time approximation schemes for (SSP). In Section 4.6 we will present a new *FPTAS* for (SSP) which is superior to all other approximation schemes and runs in $O(\min\{n \cdot 1/\varepsilon, n + 1/\varepsilon^2 \log(1/\varepsilon)\})$ time and requires only $O(n + 1/\varepsilon)$ space. We will close this chapter with a study on the computational behavior of the new *FPTAS*.