Chapter 9
Split and List

In this chapter we discuss several algorithms based on the following approach. There is a number of efficient algorithms for many problems in P. To apply these algorithms on hard problems, we (exponentially) enlarge the size of a hard problem and apply fast polynomial time algorithm on an input of exponential size. The common way to enlarge the problem is to split the input into parts, and for each part to enumerate (or list) all possible solutions to subproblems corresponding to the part. Then we combine solutions of subproblems to solutions of the input of the original problem by making use of a fast polynomial time algorithm.

9.1 Sort and Search

First let us recall that on a vector space $\mathbb{Q}^m$ over the rational numbers $\mathbb{Q}$ one can define a *lexicographical order* denoted by $\prec$. For vectors $\mathbf{x} = (x_1, x_2, \ldots, x_m)$, $\mathbf{y} = (y_1, y_2, \ldots, y_m) \in \mathbb{Q}^m$, we define that $\mathbf{x} \prec \mathbf{y}$ if and only if there is a $t \in \{1, 2, \ldots, m\}$ such that $x_i = y_i$ for all $i < t$ and $x_t < y_t$. For example, $(2,4,8,3) \prec (2,7,2,4)$. We also write $\mathbf{x} \preceq \mathbf{y}$ if $\mathbf{x} \prec \mathbf{y}$ or $\mathbf{x} = \mathbf{y}$.

Before proceeding with the Split & List technique, let us play a bit with the following "toy" problem. In the 2-TABLE problem, we are given 2 tables T_1 and T_2 each being an array of size $m \times k$, and a vector $\mathbf{s} \in \mathbb{Q}^m$. Each table consists of k vectors of $\mathbb{Q}^m$ in such a way that each vector is a column of the array. The question is, if the table contains an entry from the first column and an entry from the second column such that the sum of these two vectors is $\mathbf{s}$?

0 1 4 3	0 1 3 0
1 3 4 3	2 1 6 0
1 5 4 3	3 1 3 0

Fig. 9.1 An instance of the 2-TABLE problem with entries from $\mathbb{Q}^3$

F.V. Fomin, D. Kratsch, *Exact Exponential Algorithms*, Texts in Theoretical Computer Science. An EATCS Series, DOI 10.1007/978-3-642-16533-7_9,

An example of an instance of a 2-TABLE problem is given in Fig. 9.1. For vector $\mathbf{s} = \begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix}$, there are two solutions $(\begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix})$ and $(\begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix})$.

A trivial solution to the 2-TABLE problem would be to try all possible pairs of vectors. Each comparison takes $\mathcal{O}(m)$, and the number of pairs of vectors is $\mathcal{O}(k^2)$, which would result in running time $\mathcal{O}(mk^2)$. There is a smarter way to solve this problem.

Lemma 9.1. *The* 2-TABLE *problem for tables T_1 and T_2 of size $m \times k$ with entries from $\mathbb{Q}^m$ can be solved in time $\mathcal{O}(mk \log k)$.*

Proof. The vectors of the first table are sorted increasingly in lexicographic order and the vectors of the second table are sorted decreasingly in lexicographic order. Two vectors can be compared in time $\mathcal{O}(m)$. Consequently the sorting can be done in time $\mathcal{O}(mk \log k)$.

Now given sorted vector sequences $\mathbf{a}_1 \preceq \mathbf{a}_2 \preceq \cdots \preceq \mathbf{a}_k$ and $\mathbf{b}_k \preceq \mathbf{b}_{k-1} \preceq \cdots \preceq \mathbf{b}_1$, the algorithm finds out whether there are vectors $\mathbf{a}_i$ and $\mathbf{b}_j$ such that $\mathbf{a}_i + \mathbf{b}_j = \mathbf{s}$. More precisely algorithm `2-table`, described in Fig. 9.2, outputs all such pairs and its correctness is based on the following observation. If $\mathbf{a}_i + \mathbf{b}_j \prec \mathbf{c}$, then for every $l \geq i$, $\mathbf{a}_i + \mathbf{b}_l \prec \mathbf{c}$, and thus all vectors $\mathbf{b}_l$, $l \geq i$, can be eliminated from consideration. Similarly, if $\mathbf{c} \prec \mathbf{a}_i + \mathbf{b}_j$, then all vectors $\mathbf{a}_l$, $l \geq i$, are eliminated from consideration. The algorithm takes $\mathcal{O}(k)$ steps and the total running time, including sorting, is $\mathcal{O}(mk \log k)$. □

Algorithm `2-table`.
Input: Tables T_1 and T_2 of size $m \times k$ with columns/vectors $\mathbf{a}_i$ in T_1 and $\mathbf{b}_j$ in T_2, and vector $\mathbf{c}$.
Output: All pairs $(\mathbf{a}_i, \mathbf{b}_j)$ such that $\mathbf{a}_i + \mathbf{b}_j = \mathbf{c}$ and $\mathbf{a}_i \in T_1$, $\mathbf{b}_j \in T_2$.

```
i := 1; j := 1
while i ≤ k and j ≤ k do
    if a_i + b_j = c then
        return (a_i, b_j)
    if a_i + b_j ≺ c then
        i := i + 1
    if c ≺ a_i + b_j then
        j := j + 1
```

Fig. 9.2 Algorithm `2-table`

Let us remark that with a simple modification that outputs all pairs $\mathbf{a}_i + \mathbf{b}_j = \mathbf{c}$ and increments counters i and j, the algorithm can enumerate all solutions $(\mathbf{a_i}, \mathbf{b_j})$ within the same running time.

The solution of surprisingly many hard problems can be reduced to the solution of the 2-TABLE problem. The main idea of the approach is to partition an input of

a problem into two subproblems, solve them separately and find the solution to the original problem by combining the solutions to the subproblems. We consider three NP-hard problems.

Subset Sum. In the SUBSET SUM problem, we are given positive integers $a_1, a_2, \ldots, a_n$, and S. The task is to find a subset $I \subseteq \{1,2,\ldots n\}$ such that

$$\sum_{i \in I} a_i = S,$$

or to report that no such subset exists. For example, for $a_1 = 5$, $a_2 = 5$, $a_3 = 10$, $a_4 = 60$, $a_5 = 61$, and $S = 70$, the solution is $I = \{1,2,4\}$.

Theorem 9.2. *The* SUBSET SUM *problem can be solved in time* $\mathcal{O}(n2^{n/2})$.

Proof. We partition $\{a_1, a_2, \ldots, a_n\}$ into two sets $X = \{a_1, a_2, \ldots, a_{\lfloor n/2 \rfloor}\}$ and $Y = \{a_{\lfloor n/2 \rfloor + 1}, \ldots, a_n\}$. For each of these two sets, we compute the set of all possible subset sums. The total number of computed sums is at most $2^{n/2+1}$. Let I_X and I_Y be the sets of computed sums for X and Y respectively (let us remark that 0 belongs to both I_X and I_Y). Then there is a solution to the SUBSET SUM problem if and only if there is an $s_X \in I_X$ and an $s_Y \in I_Y$ such that $s_X + s_Y = S$. To find such s_X and s_Y, we reduce the problem to an instance of the 2-TABLE problem. We build an instance of the 2-TABLE. Table T_1 is formed by the elements of I_X and table T_2 is formed by the elements of I_Y. Both are arrays of size $m \times k$, where $m = 1$ and $k \leq 2^{n/2}$. Then by Lemma 9.1, we can find two elements, one from each table, whose sum is S (if they exist) in time $\mathcal{O}(2^{n/2} \log 2^{n/2}) = \mathcal{O}(n2^{n/2})$. □

Exact Satisfiability. In the EXACT SATISFIABILITY problem (XSAT), we are given a CNF-formula F with n variables and m clauses. The task is to find a satisfying assignment of F such that each clause contains exactly one true literal. For example, the CNF formula

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$$

is satisfied by the truth assignment $x_1 =$ true, $x_2 =$ false, and $x_3 =$ true, moreover, for this assignment, each clause is satisfied by exactly one literal.

While there are faster branching algorithms for XSAT, we find this example interesting because its comparison with SAT helps us to better understand which kind of properties are necessary to reduce a problem to the 2-TABLE problem.

Theorem 9.3. *The problem* XSAT *is solvable in time* $\mathcal{O}^*(2^{n/2})$.

Proof. Let F be an input of XSAT. Let its set of clauses be $\{c_1, c_2, \ldots, c_m\}$ and let its set of variables be $\{x_1, x_2, \ldots, x_n\}$. We split the variables into two sets $X = \{x_1, x_2, \ldots, x_{\lfloor n/2 \rfloor}\}$ and $Y = \{x_{\lfloor n/2 \rfloor + 1}, \ldots, x_n\}$. For every possible truth assignment f of the variables of X which assigns to each variable either the value true or false, we form its characteristic vector $\chi(f,X) \in \mathbb{Q}^m$. The ith coordinate of $\chi(f,X)$ is equal to the number of literals which evaluate to true in the clause c_i. Similarly, for

every possible truth assignment g of the variables of Y we form its characteristic vector $\chi(g,Y) \in \mathbb{Q}^m$. The jth coordinate of $\chi(g,Y)$ is equal to the number of literals which evaluate to true in the clause c_j.

Let us note that the input formula F is exactly satisfied if and only if there is an assignment f of X and an assignment g of Y such that $\chi(f,X)+\chi(g,Y) = (1,1,\ldots,1)$. We form two tables: table T_1 contains characteristic vectors of X and table T_2 contains characteristic vectors of Y. Each table has at most $2^{\lceil n/2 \rceil}$ columns. Thus we can again apply Lemma 9.1, and solve XSAT in time $\mathcal{O}^*(2^{n/2})$. □

Why can this approach not be used to solve SAT in time $\mathcal{O}^*(2^{n/2})$? This is because by constructing an instance of the 2-TABLE for an instance of SAT the same way as we did for XSAT, we have to find characteristic vectors such that $(1,1,\ldots,1) \preceq \chi(f,X)+\chi(g,Y)$. This is a real obstacle, because we cannot use Lemma 9.1 anymore: the argument "if $\mathbf{a}_i + \mathbf{b}_j \prec (1,1,\ldots,1)$, then for every $l \geq i$, $\mathbf{a}_i + \mathbf{b}_l \prec (1,1,\ldots,1)$" does not hold anymore. In the worst case (without having any ingenious idea) we have to try all possible pairs of vectors.

Knapsack. In the BINARY KNAPSACK problem, we are given a positive integer W and n items $s_1, s_2, \ldots, s_n$, each item has its value a_i and its weight w_i, which are positive integers. The task is to find a subset of items of maximum total value subject to the constraint that the total weight of these items is at most W.

To solve the BINARY KNAPSACK problem in time $\mathcal{O}^*(2^{n/2})$, we reduce its solution to the solution of the following MODIFIED 2-TABLE problem. We are given two tables T_1 and T_2 each one an array of size $1 \times k$ whose entries are positive integers, and an integer W. The task is to find one number from the first and one from the second table whose sum is at most W. An example is given in Fig. 9.3.

10	2	4	12	15	6	11	14

Fig. 9.3 In this example, for $W = 14$, the solution is the pair of integers $(2,11)$.

The problem MODIFIED 2-TABLE can be solved in time $\mathcal{O}(k \log k)$ with an algorithm similar to the one for `2-table`.

In the first step, the algorithms sorts the entries of T_1 in increasing order and the ones of T_2 in decreasing order. Let x be an entry in T_1 and y an entry in T_2. We observe the following. If $x+y \leq W$, then for all z appearing after y in T_2, we have $z < y$, and, consequently, $x+z \leq x+y$. Therefore, all such pairs (x,z) can be eliminated from consideration, as they cannot provide a better answer than (x,y). Similarly, if $x+y > W$ then for all z appearing after x in T_1, $z+y > W$, and thus all pairs (z,y), $z \geq x$, can be eliminated from consideration. Thus after sorting, which requires $\mathcal{O}(k \log k)$ time, one can find the required pair of numbers in $\mathcal{O}(k)$ steps. This observation is used to prove the following theorem.

Theorem 9.4. *The* BINARY KNAPSACK *problem is solvable in time* $\mathcal{O}^*(2^{n/2})$.

Proof. To solve the BINARY KNAPSACK problem, we split the set of items into two subsets $s_1, s_2, \ldots, s_{\lfloor n/2 \rfloor}$ and $s_{\lfloor n/2 \rfloor + 1}, \ldots, s_n$, and for each subset $I \subseteq \{1, 2, \ldots, \lfloor n/2 \rfloor\}$, we construct a couple $x_I = (A_I, W_I)$, where

$$A_I = \sum_{i \in I} a_i, \text{ and } W_I = \sum_{i \in I} w_i.$$

Thus we obtain a set X of couples and the cardinality of X is at most $2^{n/2}$. Similarly we construct the set Y which consists of all couples $y_J = (A_J, W_J)$, where $J \subseteq \{\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 1 + 1, \ldots, n\}$. Then the problem boils down to finding couples $x_I \in X$ and $y_J \in Y$ such that $A_I + A_J$ is maximum subject to the constraint $W_I + W_J \leq W$.

To reduce the problem to an instance of the MODIFIED 2-TABLE problem discussed above, we perform the following preprocessing: a couple (A_I, W_I) is removed from X (or Y) if there is a couple $(A_{I'}, W_{I'})$, $I \neq I'$, from the same set such that $A_{I'} \geq A_I$ and $W_{I'} \leq W_I$. The argument here is that the set of items with couple $(A_{I'}, W_{I'})$ has higher value and smaller weight, so we prefer $(A_{I'}, W_{I'})$ and can safely remove (A_I, W_I) from X. In the case of $(A_I, W_I) = (A_{I'}, W_{I'})$, we break ties arbitrarily. In other words, we remove couples dominated by some other couple.

This preprocessing is done in time $\mathcal{O}^*(n2^{n/2})$ in the following way for X and similarly for Y. First the items of the set X (or Y) are sorted in increasing order according to their weights. At the second step of the preprocessing we are given a list of couples sorted by increasing weights

$$(A_1, W_1), (A_2, W_2), \cdots, (A_k, W_k),$$

where $k \leq 2^{n/2}$ and for every $1 \leq i < j \leq k$, $W_i \leq W_j$. We put $A := A_1$ and move in the list from 1 to k performing the following operations: if $A_i > A$, we put $A := A_i$. Otherwise ($A_i \leq A$), we remove (A_i, W_i) from the list. This procedure takes $\mathcal{O}(k)$ steps and as the result of it we have produced a set of couples with no couple dominated by any other one.

Thus after preprocessing done for X and for Y, we have that $(A_I, W_I) \in X$ and $(A_J, W_J) \in Y$ have maximum sum $A_I + A_J$ subject to $W_I + W_J \leq W$ if and only if the sum $W_I + W_J$ is maximum subject to $W_I + W_J \leq W$. What remains is to construct the table of size $2 \times 2^{n/2}$ and use the algorithm for the MODIFIED 2-TABLE problem. This step requires time $\mathcal{O}^*(2^{n/2})$. This concludes the proof. □

A natural idea to improve the running time of all algorithms based on reductions to the k-TABLE problem, is to partition the original set into $k \geq 3$ subsets and reduce to the k-TABLE problem. However, it is not clear how to use this approach to obtain better overall running times. Consider the following k-TABLE problem: given k tables $T_1, T_2, \ldots T_k$ such that each table is an array of size $m \times k$ with entries from $\mathbb{R}^m$, the task is for a given vector $\mathbf{c}$, to find a set of vectors $(\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k)$, $\mathbf{c}_i \in T_i$, such that

$$\mathbf{c}_1 + \mathbf{c}_2 + \cdots + \mathbf{c}_k = \mathbf{c}.$$

We can solve the k-TABLE problem in time $\mathcal{O}(n^{k-1} + kn\log n)$ by recursively applying the algorithm for the 2-TABLE problem. Unfortunately we do not know any faster algorithm for this problem.

Thus if we split an instance of a hard problem, like XSAT, into k subsets, construct $2^{n/k}$ sets of vectors for each table, and use an algorithm for solving the k-TABLE problem, we obtain an algorithm of running time $\mathcal{O}^*(2^{(k-1)n/k})$.

However, the idea of reducing to a k-TABLE problem can be useful to reduce the space required by such algorithms. All algorithms discussed in this section keep tables of sizes $m \times 2^{n/2}$ and thus the space needed is $2^{n/2}$. Schroeppel and Shamir [202] used the k-TABLE problem to reduce the space requirement of such algorithms to $2^{n/4}$.

9.2 Maximum Cut

In this section we describe an algorithm due to Williams solving the MAXIMUM CUT problem. The algorithm is based on a fast way of finding triangles in a graph. This approach is based on fast square matrix multiplication. Let us recall, that the product of two $n \times n$ matrices can be computed in $\mathcal{O}(n^\omega)$ time, where $\omega < 2.376$ is the so-called square matrix multiplication exponent.

Maximum Cut. In the MAXIMUM CUT problem (Max-Cut), we are given an undirected graph $G = (V,E)$. The task is to find a set $X \subseteq V$ maximizing the value of $\mathrm{CUT}(X, V \setminus X)$, i.e. the number of edges with one endpoint in X and one endpoint in $V \setminus X$.

While a naive way of finding a triangle in a graph would be to try all possible triples of vertices, there is a faster algorithm for doing this job.

Theorem 9.5. *A triangle in a graph on n vertices can be found in time $\mathcal{O}(n^\omega)$ and in $\mathcal{O}(n^2)$ space.*

Proof. Let $A(G)$ be the adjacency matrix of G. It is easy to prove that in the kth power $(A(G))^k$ of $A(G)$ the entry $(A(G))^k[i,i]$ on the main diagonal of $(A(G))^k$ is equal to the number of walks of length k which start and end in vertex i. Every walk of length 3 which starts and ends at i must pass through 3 vertices, and thus is a triangle. We conclude that G contains a triangle if and only if $(A(G))^3$ has a non-zero entry on its main diagonal. The space required to compute the product of matrices is proportional to the size of $A(G)$, which is n^2. □

Theorem 9.6. *The* MAXIMUM CUT *problem on n-vertex graphs is solvable in time $\mathcal{O}^*(2^{\omega n/3}) = \mathcal{O}(1.7315^n)$, where $\omega < 2.376$ is the square matrix multiplication exponent.*

Proof. Let us assume that $G = (V,E)$ is a graph on n vertices and that n is divisible by 3. (If not we can add one or two isolated vertices which do not change the value of

the maximum cut and add a polynomial factor to the running time of the algorithm.) Let V_0, V_1, V_2 be an arbitrary partition of V into sets of sizes $n/3$.

We construct an auxiliary weighted directed graph $A(G)$ as follows. For every subset $X \subseteq V_i$, $0 \leq i \leq 2$, the graph $A(G)$ has a vertex X. Thus $A(G)$ has $3 \cdot 2^{n/3}$ vertices. The arcs of $A(G)$ are all possible pairs of the form (X,Y), where $X \subseteq V_i$, $Y \subseteq V_j$, and $j = i+1 \pmod 3$. Thus $A(G)$ has $3 \cdot 2^{2n/3}$ arcs. For every arc (X,Y) with $X \subseteq V_i$ and $Y \subseteq V_j$, $i \neq j$, we define its weight

$$w(X,Y) = CUT(X, V_i \setminus X) + \mathrm{CUT}(X, V_j \setminus Y) + CUT(Y, V_i \setminus X).$$

Claim. The following properties are equivalent

(i) There is $X \subseteq V$ such that $\mathrm{CUT}(X, V \setminus X) = t$.

(ii) The auxiliary graph $A(G)$ contains a directed triangle X_0, X_1, X_2, $X_i \subseteq V_i$, $0 \leq i \leq 2$, such that

$$t = w(X_0, X_1) + w(X_1, X_2) + w(X_2, X_0).$$

Proof (Proof of Claim). To prove $(i) \Rightarrow (ii)$, we put $X_i = X \cap V_i$, $0 \leq i \leq 2$. Then every edge e of G contributes 1 to the sum $w(X_0, X_1) + w(X_1, X_2) + w(X_2, X_0)$ if e is an edge between X and $V \setminus X$, and 0 otherwise. To prove $(ii) \Rightarrow (i)$, we put $X = X_0 \cup X_1 \cup X_2$. Then again, every edge is counted in $\mathrm{CUT}(X, V \setminus X)$ as many times as it is counted in $w(X_0, X_1) + w(X_1, X_2) + w(X_2, X_0)$. □

To find out whether the condition (ii) of the claim holds, we do the following. We try all possible values of $w(X_i, X_j)$, $j = i+1 \pmod 3$. Thus for every triple $W = (w_{01}, w_{12}, w_{20})$ such that $w = w_{01} + w_{12} + w_{20}$, we consider the subgraph $A(G,W)$ of $A(G)$ which contains only the arcs of weight w_{ij} from $X_i \subseteq V_i$ to $X_j \subseteq V_j$. For every value of t, the number of such triples is at most t^3. The subgraph $A(G,W)$ can be constructed in time $\mathcal{O}^*(2^{2n/3})$ by going through all arcs of $A(G)$. But then there exists a triple W satisfying (ii) if and only if the underlying undirected graph of $A(G,W)$ contains a triangle of weight W. By Theorem 9.5, verifying whether such a triangle exists can be done in time $\mathcal{O}^*(2^{\omega n/3})$. Thus for every value of t, we try all possible partitions of t, and for each such partition we construct the graph $A(G,W)$ and check whether it contains a triangle of weight t. The total running time is

$$\mathcal{O}^*(t \cdot t^3 (2^{\omega n/3} + 2^{2n/3})) = \mathcal{O}^*(2^{\omega n/3}).$$

Notes

The name Split and List for the technique is due to Ryan Williams, who used it in his PhD thesis [217]. The algorithms for SUBSET SUM and BINARY KNAPSACK are due to Horowitz and Sahni [117] and Schroeppel and Shamir [202]. Note that this is an early paper of Adi Shamir, one of the three inventors of the RSA public-key cryptosystem. The space requirements in these algorithms can be improved to $\mathcal{O}^*(2^{n/4})$ while keeping the same running time of $\mathcal{O}^*(2^{n/2})$ [202]. Howgrave-Graham and

Joux improve the algorithm of Schroeppel and Shamir for SUBSET SUM on random inputs i [118].

Fomin, Golovach, Kratochvil, Kratsch and Liedloff used the Split and List approach to list different types of (σ,ρ) dominating sets in graphs [82]. Klinz and Woeginger used this approach for computing power indices in voting games [131].

Williams [216] provides a variant of Theorem 9.6 for solving a more general counting version of WEIGHTED 2-CSP (a variant of constraint satisfaction with constraints of size at most 2). Williams' PhD thesis [217] contains further generalizations of this approach.

Theorem 9.5 is due to Itai and Rodeh [121]. A natural question concerning the proof of Theorem 9.6 is, whether partitioning into more than three parts would be useful. The real obstacle is the time spent to find a clique of size k in a graph. Despite many attempts, the following result of Nešetřil & Poljak was not improved for more than 25 years: The number of cliques of size $3k$ in an n-vertex graph can be found in time $\mathcal{O}(n^{\omega k})$ and space $\mathcal{O}(n^{2k})$ [163]. Eisenbrand and Grandoni [67] succeeded in improving the result of Nešetřil and Poljak for a $(3k+1)$-clique and a $(3k+2)$-clique for small values of k. In particular, they show how to find a cliques of size 4,5, and 7 in time $\mathcal{O}(n^{3.334})$ $\mathcal{O}(n^{4.220})$, and $\mathcal{O}(n^{5.714})$, respectively.

The first algorithm that performs a matrix multiplication faster than the standard Gaussian elimination procedure implying $\omega \leq \log_2 7 < 2.81$ is due to Strassen [210]. The proof that $\omega < 2.376$ is due to Coppersmith and Winograd [51].