

# An output-sensitive Algorithm to partition a Sequence of Integers into Subsets with equal Sums

Alexander Büchel<sup>1,2</sup>    Ulrich Gilleßen<sup>2</sup>    Kurt-Ulrich Witt<sup>1,2</sup>

<sup>1</sup> *b-it Applied Science Institute, University Bonn-Rhein-Sieg, Sankt Augustin, Germany*

<sup>2</sup> *Research Group Discrete Mathematics and Optimization (ADIMO), University Bonn-Rhein-Sieg, Sankt Augustin, Germany*

received 18<sup>th</sup> July 2017, revised 18<sup>th</sup> July 2018, 27<sup>th</sup> Nov. 2018, accepted 4<sup>th</sup> Dec. 2018.

We present a polynomial time algorithm, which solves a nonstandard variation of the well-known PARTITION problem: Given positive integers  $n, k$  and  $t$  such that  $t \geq n$  and  $k \cdot t = \binom{n+1}{2}$ , the algorithm partitions the elements of the set  $I_n = \{1, \dots, n\}$  into  $k$  mutually disjoint subsets  $T_j$  such that  $\cup_{j=1}^k T_j = I_n$  and  $\sum_{x \in T_j} x = t$  for each  $j \in \{1, 2, \dots, k\}$ . The algorithm needs  $\mathcal{O}\left(n \cdot \left(\frac{n}{2k} + \log \frac{n(n+1)}{2k}\right)\right)$  steps to insert the  $n$  elements of  $I_n$  into the  $k$  sets  $T_j$

**Keywords:** Set partition problem, Cutting sticks problem

## 1 Introduction

For  $n \in \mathbb{N}$  let  $I_n = \{1, \dots, n\}$  be the set of integers from 1 to  $n$ , and  $\Delta_n = \frac{n(n+1)}{2}$  the sum of these elements. In this paper we consider a variant of the PARTITION problem and present a solution for a class of special instances of this variant. The general version of our variant is given by  $n, k, t_1, \dots, t_k \in \mathbb{N}$ , and the question is whether there exists  $k$  pairwise disjoint subsets  $T_j \subseteq I_n$  such that the elements of  $T_j$  add up to  $t_j$ , and the union of these sets equals  $I_n$ . We call such a collection of sets  $T_j$  a  $(t_1, t_2, \dots, t_k)$ -partition of  $I_n$ .

Fu and Hu (2015) show, that for  $k, l, t \in \mathbb{N}$  with  $0 < l \leq \Delta_n$  and  $(k-1)t + l + \Delta_{k-2} = \Delta_n$  a  $(t, t+1, \dots, t+k-2, l)$ -partition of  $I_n$  exists. Chen et al. (2015) prove, that a  $(t_1, \dots, t_k)$ -partition of  $I_n$  exists, if  $\sum_{j=1}^k t_j = \Delta_n$  and  $t_j \geq t_{j+1}$  for  $1 \leq j \leq k-1$  and  $t_{k-1} \geq n$  hold. In Büchel et al. (2016) we present a 0/1-linear program to solve partition problems.

In the special case, where  $t_j = t = \text{const}$  we call  $T_1, \dots, T_k$  a  $(k, t)$ -partition of  $I_n$ . Given  $n, k, t \in \mathbb{N}$  with  $t \geq n$  and  $\Delta_n = k \cdot t$  the decision problem reduces to the question, whether a  $(k, t)$  partition of  $I_n$  exists. Straight and Schillo (1979) show that for all  $k, t$  with  $\Delta_n = k \cdot t$  and  $t \geq n$  a partition of  $I_n$  exists. Ando et al. (1990) withdraw the condition  $\Delta_n = k \cdot t$  and prove that for positive integers  $n, k$  and  $t$ , the set  $I_n$  contains  $k$  disjoint subsets having a constant sum  $t$  if and only if  $k(2k-1) \leq k \cdot t \leq \Delta_n$ .

Where as the cited papers study for which  $k$ -tuples  $(t_1, \dots, t_k)$ -partitions of  $I_n$  exist, we are interested in efficient algorithms to determine partitions. In this paper we consider problem instances  $\Pi(n, k, t)$  with  $t \geq n$  and  $\Delta_n = k \cdot t$ . In Section 3 we introduce the recursive algorithm  $\Pi\text{Solve}$  which determines a partition for each instance  $\Pi(n, k, t)$ . Before, in Section 2 we present the so called meander algorithm which solves problem instances  $\Pi(n, k, t)$ , where  $n$  is even and  $2k$  is a divisor of  $n$  or where  $n$  is odd and  $2k$  divides  $n + 1$ , respectively. The reason is, that  $\Pi\text{Solve}$  can be stopped, when one of these conditions is reached, and the remaining partition can be determined directly by means of the meander algorithm. In Section 4 we analyze the run time complexity of  $\Pi\text{Solve}$ . Section 5 summarizes the paper and mentions some ideas to improve  $\Pi\text{Solve}$ .

Inputs for the algorithms are  $n$ ,  $k$  and  $t$ , hence these have length  $\mathcal{O}(\log n)$ . Since it is to be expected that the complexity to insert  $n$  elements into  $k$  sets is at least  $\mathcal{O}(n)$ , we will consider the complexity of the algorithms not depending on the size of the inputs, but output-sensitive, i.e. depending on  $n$  and  $k$ .

## 2 Meander Algorithm

For  $a \in \mathbb{N}_0$  and  $b \in \mathbb{N}$  we denote  $b|a$  if  $b$  is a divisor of  $a$ . Given the problem instance  $\Pi(n, k, t)$  the meander algorithm applies if  $n$  is even and  $2k|n$  or if  $n$  is odd and  $2k|n + 1$ , respectively. The algorithm distributes the elements of the set  $I_n$  into the subsets  $T_j$  such that these sets build a  $(k, t)$ -partition of  $I_n$ , i.e. the sets  $T_j$  fulfill the conditions

$$T_i \cap T_j = \emptyset, \quad 1 \leq i, j \leq k, \quad i \neq j \quad (1)$$

$$\bigcup_{j=1}^k T_j = I_n \quad (2)$$

$$\sum_{x \in T_j} x = t, \quad 1 \leq j \leq k \quad (3)$$

### 2.1 Case: $n$ even and $2k|n$

Figure 1 shows the part of the meander algorithm which solves problem instances  $\Pi(n, k, t)$  when  $n$  is even and  $2k$  divides  $n$ . To prove that the algorithm determines a correct  $(k, t)$ -partition of  $I_n$  we have to show that the partition fulfills the conditions above. Condition (1) is obviously fulfilled. We will verify (2) in Lemma 2.1 and (3) in Lemma 2.2.

Let

$$X_1(n, k) = \left\{ 2ki - (j - 1) \mid 1 \leq i \leq \frac{n}{2k}, 1 \leq j \leq k \right\} \quad (4)$$

$$X_2(n, k) = \left\{ 2k(i - 1) + j \mid 1 \leq i \leq \frac{n}{2k}, 1 \leq j \leq k \right\} \quad (5)$$

be the sets of elements of  $I_n$  which are distributed in assignment (I) or assignment (II), respectively.

**Lemma 2.1** *Let  $\Pi(n, k, t)$  be a problem instance such that  $n$  even and  $2k|n$ , then  $I_n = X_1(n, k) \cup X_2(n, k)$ .*

---

```

meandereven( $n, k, t$ );
input:  $n, k, t$  with  $n$  even,  $2k|n$ ,  $t \geq n$ , and  $\Delta_n = k \cdot t$ ;
output:  $(k, t)$ -partition  $T_j, 1 \leq j \leq k$ , of  $I_n$ ;
 $T_j := \emptyset, 1 \leq j \leq k$ ;
for  $j := 1$  to  $k$  do
  for  $i := 1$  to  $\frac{n}{2k}$  do
    (I)  $T_j := T_j \cup \{2ki - (j - 1)\}$ ;
    (II)  $T_j := T_j \cup \{2k(i - 1) + j\}$ ;
  endfor;
endfor;
end.

```

---

**Fig. 1: Meander Algorithm in case  $n$  even and  $2k|n$ .**

---

**Proof:** For each  $x \in I_n$  there exist unambiguously  $i, r$  such that

$$x = 2k(i - 1) + r, 1 \leq i \leq \frac{n}{2k}, 1 \leq r \leq 2k \quad (6)$$

We consider the two following sets of remainders  $r \in I_{2k}$ :  $R_1 = \{2k - (j - 1) \mid 1 \leq j \leq k\}$  and  $R_2 = \{j \mid 1 \leq j \leq k\}$ . Since  $r \in R_1$ , if  $k + 1 \leq r \leq 2k$ , it follows  $R_1 \cap R_2 = \emptyset$  and  $R_1 \cup R_2 = I_{2k}$ . Thus with respect to (6) we get either

$$x = 2k(i - 1) + 2k - (j - 1) = 2ki - (j - 1) \quad (7)$$

or

$$x = 2k(i - 1) + j \quad (8)$$

It follows  $x \in X_1(n, k) \cup X_2(n, k)$ . Hence we have shown  $I_n \subseteq X_1(n, k) \cup X_2(n, k)$ .

If  $x \in X_1(n, k)$ , then  $k + 1 \leq x \leq n$ , and if  $x \in X_2(n, k)$  then  $1 \leq x \leq n - k$ . Thus, if  $x \in X_1(n, k) \cup X_2(n, k)$ , we have  $1 \leq x \leq n$ , hence  $x \in I_n$  and thereby  $X_1(n, k) \cup X_2(n, k) \subseteq I_n$ .  $\square$

**Lemma 2.2** Let  $\Pi(n, k, t)$  be a problem instance with  $n$  even and  $2k|n$ , then the output  $T_j, 1 \leq j \leq k$ , of **meandereven**( $n, k, t$ ) fulfills condition (3).

**Proof:** For each  $j \in \{1, \dots, k\}$  we have:  $\sum_{x \in T_j} x =$

$$\sum_{i=1}^{\frac{n}{2k}} (2ki - (j - 1)) + \sum_{i=1}^{\frac{n}{2k}} (2k(i - 1) + j) = 2k \sum_{i=1}^{\frac{n}{2k}} (2i - 1) + \frac{n}{2k} = 2k \frac{n^2}{4k^2} + \frac{n}{2k} = \frac{n(n + 1)}{2k} = t$$

$\square$

**Theorem 2.1**  $\text{meandereven}(n, k, t)$

a) determines a correct partition of  $I_n$  for all problem instances  $\Pi(n, k, t)$  with  $n$  even and  $2k|n$ ,

and

b) needs  $\mathcal{O}(n)$  steps to insert the  $n$  elements of  $I_n$  into the sets  $T_j$ .

**Proof:** a) follows immediately from Lemmas 2.1 and 2.2, and b) is obvious.  $\square$

## 2.2 Case: $n$ odd and $2k|n+1$

To solve problem instances  $\Pi(n, k, t)$  with  $n$  odd and  $2k|n+1$  we adapt slightly the  $\text{meandereven}$ -algorithm (see Fig. 2). The correctness of the  $\text{meanderodd}$ -algorithm can be shown analogously to the proof of the correctness of the  $\text{meandereven}$ -algorithm. At this point we define the sets of elements assigned due to labels (I) and (II) in the  $\text{meanderodd}$ -algorithm as

$$X'_1(n, k) = \left\{ 2ki - j \mid 1 \leq i \leq \frac{n+1}{2k}, 1 \leq j \leq k \right\} \quad (9)$$

$$X'_2(n, k) = \left\{ 2k(i-1) + (j-1) \mid 1 \leq i \leq \frac{n+1}{2k}, 1 \leq j \leq k \right\} \quad (10)$$

---

```

meanderodd( $n, k, t$ );
input:   $n, k, t$  with  $n$  odd,  $2k|n+1$ ,  $t \geq n$ , and  $\Delta_n = k \cdot t$ ;
output: ( $k, t$ )-partition  $T_j, 1 \leq j \leq k$ , of  $I_n$ ;
   $T_j := \emptyset, 1 \leq j \leq k$ ;
  for  $j := 1$  to  $k$  do
    for  $i := 1$  to  $\frac{n}{2k}$  do
      (I)  $T_j := T_j \cup \{2ki - j\}$ ;
      (II)  $T_j := T_j \cup \{2k(i-1) + (j-1)\}$ ;
    endfor;
  endfor;
end.

```

---

**Fig. 2:** Meander Algorithm in case  $n$  odd and  $2k|n+1$ .

**Remark 2.1** In order to avoid a case distinction, we first assign the element 0 ( $i = 1, j = 1$ ) to set  $T_1$ . For this reason, in the following we assume that  $I_n$  contains the element 0, too.  $\square$

**Lemma 2.3** Let  $\Pi(n, k, t)$  be a problem instance such that  $n$  odd and  $2k|n+1$ , then  $I_n = X'_1(n, k) \cup X'_2(n, k)$ .

**Proof:** For each  $x \in I_n$  there exist unambiguously  $i, r$  such that

$$x = 2k(i - 1) + r, 1 \leq i \leq \frac{n+1}{2k}, 0 \leq r \leq 2k - 1 \quad (11)$$

We consider the sets of remainders  $r \in I_{2k-1}$ :  $R'_1 = \{2k - j \mid 1 \leq j \leq k\}$  and  $R'_2 = \{j - 1 \mid 1 \leq j \leq k\} = \{j \mid 0 \leq j \leq k - 1\}$ . Since  $r \in R'_1$ , if  $k \leq r \leq 2k - 1$ , it follows  $R_1 \cap R_2 = \emptyset$  and  $R_1 \cup R_2 = I_{2k-1}$ . Thus with respect to (11) we get

$$x = 2k(i - 1) + 2k - j = 2ki - j \quad (12)$$

or

$$x = 2k(i - 1) + (j - 1) \quad (13)$$

respectively. It follows  $x \in X'_1(n, k) \cup X'_2(n, k)$ . Hence we have shown  $I_n \subseteq X'_1(n, k) \cup X'_2(n, k)$ .

If  $x \in X'_1(n, k)$ , then  $k \leq x \leq n$ , and if  $x \in X'_2(n, k)$  then  $0 \leq x \leq n - k$ . Thus, if  $x \in X'_1(n, k) \cup X'_2(n, k)$ , we have  $0 \leq x \leq n$ , hence  $x \in I_n$  and thereby  $X'_1(n, k) \cup X'_2(n, k) \subseteq I_n$ .  $\square$

**Lemma 2.4** Let  $\Pi(n, k, t)$  be a problem instance with  $n$  odd and  $2k \mid n + 1$ , then the output  $T_j, 1 \leq j \leq k$ , of `meanderodd`( $n, k, t$ ) fullfills condition (3).

**Proof:** For each  $j \in \{1, \dots, k\}$  we have

$$\begin{aligned} \sum_{x \in T_j} x &= \sum_{i=1}^{\frac{n+1}{2k}} (2ki - j) + \sum_{i=1}^{\frac{n+1}{2k}} (2k(i - 1) + (j - 1)) \\ &= 2k \sum_{i=1}^{\frac{n}{2k}} (2i - 1) - \frac{n+1}{2k} = 2k \frac{(n+1)^2}{4k^2} - \frac{n+1}{2k} \\ &= \frac{n(n+1)}{2k} = t \end{aligned}$$

$\square$

**Theorem 2.2** `meanderodd`( $n, k, t$ )

- a) determines a correct partition of  $I_n$  for all problem instances  $\Pi(n, k, t)$  with  $n$  odd and  $2k \mid n + 1$ ,
- and
- b) needs  $\mathcal{O}(n)$  steps to insert the  $n$  elements of  $I_n$  into the sets  $T_j$ .

**Proof:** a) follows from Lemmas 2.3 and 2.4, and b) is obvious.  $\square$

### 3 The Algorithm `ISolve`

In this section we present the different cases which the `ISolve`-algorithm distinguishes using ideas similar to those used in Straight and Schillo (1979). The input to the algorithm are the integers  $n, k, t \in \mathbb{N}$  with  $t \geq n$  and  $\Delta_n = k \cdot t$ . The output is a  $(k, t)$ -partition  $T_j, 1 \leq j \leq k$ , of  $I_n$ , which fullfills condition (3). We prove that the algorithm works correctly in all cases.

### 3.1 Case: $2n > t$

In this case the algorithm makes a distinction between the cases  $t$  even and  $t$  odd.

#### 3.1.1 Case: $t$ even

The algorithm starts with filling  $\frac{2n-t}{2}$  sets as follows:

$$T_j = \{t - n + (j - 1), n - (j - 1)\}, 1 \leq j \leq \frac{2n - t}{2} \quad (14)$$

Obviously these sets are disjoint and fulfill condition (3). The union of these sets is the set  $\{t - n, \dots, \frac{t}{2} - 1, \frac{t}{2} + 1, \dots, n\}$ . Thus the elements of the set  $I_{t-n-1}$  and the element  $\frac{t}{2}$  remain, these have to be distributed into the empty  $k - \frac{2n-t}{2}$  sets. To do this, each of these sets is split into two subsets:

$$T_j = T_{j,1} \cup T_{j,2}, \frac{2n-t}{2} + 1 \leq j \leq k \quad (15)$$

The total number of these subsets is  $2(k - n) + t$ . The set  $T_{\frac{2n-t}{2}+1,1}$  is filled with the element  $\frac{t}{2}$ :

$$T_{\frac{2n-t}{2}+1,1} = \left\{ \frac{t}{2} \right\} \quad (16)$$

Thus it remains to distribute the elements of  $I_{t-n-1}$  into the  $2(k - n) + t - 1$  sets  $T_{\frac{2n-t}{2}+1,2}$  and  $T_{j,s}$ ,  $\frac{2n-t}{2} + 2 \leq j \leq k$ ,  $s \in \{1, 2\}$ , i.e. it remains to solve the problem instance  $\Pi(n', k', t')$  where

$$n' = t - n - 1 \quad (17)$$

$$k' = 2(k - n) + t - 1 \quad (18)$$

$$t' = \frac{t}{2} \quad (19)$$

We have to verify that this instance fulfills the input conditions

$$\Delta_{n'} = k' \cdot t' \quad (20)$$

and

$$t' \geq n' \quad (21)$$

Using (17) – (19) we get on one side

$$\Delta_{n'} = \frac{n'(n' + 1)}{2} = \frac{(t - n - 1)(t - n)}{2} = \Delta_n + \frac{t^2 - 2tn - t}{2} \quad (22)$$

and on the other side

$$k' \cdot t' = (2(k - n) + t - 1) \cdot \frac{t}{2} = k \cdot t + \frac{t^2 - 2tn - t}{2} \quad (23)$$

Since for our initial problem  $\Pi(n, k, t)$  the condition  $\Delta_n = k \cdot t$  holds, the verification of (20) follows immediately from (22) and (23).

From  $2n > t$  immediately follows  $\frac{t}{2} > t - n - 1$ . Using (17) and (19) condition (21) is verified, too.

Thus the algorithm can recursively continue to solve the initial problem by determining a solution for the instance  $\Pi(n', k', t')$ .

### 3.1.2 Case: $t$ odd

In this case the algorithm initially fills  $\frac{2n-t+1}{2}$  sets as follows:

$$T_j = \{t - n + (j - 1), n - (j - 1)\}, 1 \leq j \leq \frac{2n - t + 1}{2} \quad (24)$$

Obviously these sets are disjoint and fulfill condition (3). The union of these sets builds the set  $\{t - n, \dots, n\}$ . Thus the elements of the set  $I_{t-n-1}$  remain, these have to be distributed into the empty  $k - \frac{2n-t+1}{2}$  sets. Therefore, the instance  $\Pi(n', k', t')$  has to be solved, where

$$n' = t - n - 1 \quad (25)$$

$$k' = k - \frac{2n - t + 1}{2} \quad (26)$$

$$t' = t \quad (27)$$

To proof that this instance is feasible we have to verify, that the input conditions (20) and (21) are fulfilled in this case as well.

Using (25) – (27) we get on one side

$$\Delta_{n'} = \frac{n'(n' + 1)}{2} = \frac{(t - n - 1)(t - n)}{2} = \Delta_n + \frac{t^2 - 2tn - t}{2} \quad (28)$$

and on the other side

$$k' \cdot t' = \left(k - \frac{2n - t + 1}{2}\right) \cdot t = k \cdot t + \frac{t^2 - 2tn - t}{2} \quad (29)$$

Since  $\Delta_n = k \cdot t$  the verification of (20) follows immediately from (28) and (29).

From  $2n > t$  it follows  $n > t - n - 1$ . From this we get by means of the input condition  $t \geq n$  and the definitions (25) and (27):  $t' = t \geq n > t - n - 1 = n'$ , i.e. condition (21) is fulfilled.

### 3.2 Case: $2n \leq t$

In this case each set  $T_j$  is split into two disjoint subsets:  $T_j = T_{j,1} \cup T_{j,2}$ ,  $1 \leq j \leq k$ . The sets  $T_{j,1}$  will be filled as follows:

$$T_{j,1} = \{n - 2k + j, n - (j - 1)\} \quad (30)$$

Hence the elements  $n - 2k + 1, \dots, n$  are already distributed, and the two elements in each of these sets add up to

$$n - (i - 1) + n - 2k + i = 2(n - k) + 1 \quad (31)$$

It remains to partition the elements of  $I_{n-2k}$  into the sets  $T_{j,2}$  such that the sum of elements in each  $T_{j,2}$  equals  $t - (2(n - k) + 1)$ . Thus it remains to solve the problem instance  $\Pi(n', k', t')$  with

$$n' = n - 2k \quad (32)$$

$$k' = k \quad (33)$$

$$t' = t - 2(n - k) - 1 \quad (34)$$

As well as in the former cases we have to assure, that the input conditions (20) and (21) are fulfilled. On the one side we have

$$\Delta_{n'} = \frac{(n-2k)(n-2k+1)}{2} = \Delta_n + 2k^2 - k - 2kn \quad (35)$$

and on the other side

$$k' \cdot t' = k \cdot (t - 2(n-k) - 1) = k \cdot t - 2kn + 2k^2 - k \quad (36)$$

(20) follows immediately from (35) and (36).

From  $t \geq 2n$  it follows  $n+1 \geq 4k$ . By subtraction we get  $t-n-1 \geq 2n-4k$  and from this and definitions (32) and (34)  $t' = t - 2n + 2k - 1 \geq n - 2k = n'$ , i.e. condition (21) is verified.

The considerations so far lead to the algorithm  $\Pi\text{Solve}$  shown in Figure 3, and we proved that it works correctly in all cases.

## 4 Complexity

In this section we analyse the worst case run time complexity of the  $\Pi\text{Solve}$ -Algorithm. The algorithm consists of four subalgorithms related to the cases we distinguish: (I)  $2k|n$  or  $2k|n+1$ , (II)  $t \geq 2n$ , (III)  $t < 2n$  and  $t$  even, (IV)  $t < 2n$  and  $t$  odd. We abbreviate these cases by  $m$  (meander),  $s$  (smaller),  $ge$  (greater even), and  $go$  (greater odd), respectively. Then the run  $\Pi\text{Solve}(n, k, t)$  can be represented by a sequence  $\rho'(n, k, t) \in \{m, s, ge, go\}^+$ .

**Example 4.1 a)** Let  $n = 1337$ . The list of runs for all partitions of  $I_{1337}$  is:

$$\begin{aligned} \rho'(1337, 3, 298151) &= m \\ \rho'(1337, 7, 127779) &= s^{94} go m \\ \rho'(1337, 21, 42593) &= s^{30} go s ge m \\ \rho'(1337, 191, 4683) &= ss go m \\ \rho'(1337, 223, 4011) &= m \\ \rho'(1337, 573, 1561) &= go m \\ \rho'(1337, 669, 1337) &= m \end{aligned}$$

**b)** Let  $n = 9999$ , then we have

$$\begin{aligned} \rho'(9999, 4444, 11250) &= ge s^3 ge^4 go m \\ \rho'(9999, 4040, 12375) &= go s^4 go s^4 go s ge m \\ \rho'(9999, 3960, 12625) &= go s^3 ge go s^8 go m \\ \rho'(9999, 3333, 15000) &= ge^3 go m \\ \rho'(9999, 12, 4166250) &= s^{415} go s ge^2 m \end{aligned}$$



---

```

ΠSolve( $n, k, t$ );
input:  $n, k, t$  with  $t \geq n$ , and  $\Delta_n = k \cdot t$ ;
output:  $(k, t)$ -partition  $T_j, 1 \leq j \leq k$ , of  $I_n$ ;

(I) case  $2k|n$ 
    then fill  $\{T_j\}_{1 \leq j \leq k}$  by meandereven( $n, k, t$ )
case  $2k|n+1$ 
    then fill  $\{T_j\}_{1 \leq j \leq k}$  by meanderodd( $n, k, t$ )

(II) case  $t \geq 2n$ 
    then for  $1 \leq j \leq k$  do  $T_{j,1} = \{n - 2k + j, n - (j - 1)\}$  endfor;
    fill  $\{T_{j,2}\}_{1 \leq j \leq k}$  by  $\Pi\text{Solve}(n - 2k, k, t - 2(n - k) - 1)$ ;
    for  $1 \leq j \leq k$  do  $T_j = T_{j,1} \cup T_{j,2}$  endfor

(III) case  $t < 2n$  and  $t$  even
    then for  $1 \leq j \leq \frac{2n-t}{2}$  do  $T_j = \{t - n + (j - 1), n - (j - 1)\}$  endfor;
     $T_{\frac{2n-t}{2}+1,1} = \{\frac{t}{2}\}$ ;
    fill  $T_{\frac{2n-t}{2}+1,2}, \{T_{j,1}\}_{\frac{2n-t}{2}+2 \leq j \leq k}$  and  $\{T_{j,2}\}_{\frac{2n-t}{2}+2 \leq j \leq k}$ 
    by  $\Pi\text{Solve}(t - n - 1, 2(k - n) + t - 1, \frac{t}{2})$ ;
    for  $\frac{2n-t}{2} + 1 \leq j \leq k$  do  $T_j = T_{\frac{2n-t}{2}+j,1} \cup T_{\frac{2n-t}{2}+j,2}$  endfor

(IV) case  $t < 2n$  and  $t$  odd
    then for  $1 \leq j \leq \frac{2n-t+1}{2}$  do  $T_j = \{t - n + (j - 1), n - (j - 1)\}$  endfor;
    fill  $\{T_j\}_{\frac{2n-t+1}{2}+1 \leq j \leq k}$  by  $\Pi\text{Solve}(t - n - 1, k - \frac{2n-t+1}{2}, t)$ 
end.

```

---

Fig. 3: Algorithm  $\Pi\text{Solve}$ .

Let  $\alpha$  be a non empty sequence over  $\Omega' = \{m, s, ge, go\}$ , then  $first(\alpha)$  is the first and  $last(\alpha)$  the last symbol of  $\alpha \in \Omega'^+$ , and  $head(\alpha)$  is the sequence without the last symbol.  $|w|_a$  is the number of occurrences of symbol  $a \in \Omega'$  in the sequence  $w \in \Omega'^*$ .

Obviously we have

**Lemma 4.1** *Let  $\Pi(n, k, t)$  be a problem instance, then  $last(\rho'(n, k, t)) = m$  and  $m$  is not a member of  $head(\rho'(n, k, t))$ .*  $\square$

Thus, we may neglect the last symbol of  $\rho'(n, k, t)$  and denote  $\rho(n, k, t) = head(\rho'(n, k, t))$ . As well we do not need the alphabet  $\Omega'$ , because  $\rho(n, k, t) \in \{s, ge, go\}^*$ . We denote this alphabet by  $\Omega$ .

Next we show, that the last call before the recursion stops with the  $m$ -case cannot be  $s$ .

**Lemma 4.2** *Let  $\Pi(n, k, t)$  be a problem instance. If  $|\rho(n, k, t)| \geq 1$ , then  $last(\rho(n, k, t)) \neq s$ .*

**Proof:** We assume  $last(\rho(n, k, t)) = s$ . Let  $\Pi(\nu, \kappa, \tau)$  be the problem instance before the last  $s$ -call. Then by (32) and (33) after the  $s$ -call we have  $\nu' = \nu - 2\kappa$  and  $\kappa' = \kappa$ . Since the next call is  $m$  it has to be  $2\kappa'|\nu'|$  or  $2\kappa'|\nu'| + 1$ , thus we have  $2\kappa|\nu - 2\kappa$  or  $2\kappa|\nu - 2\kappa + 1$ . It follows  $2\kappa|\nu$  or  $2\kappa|\nu + 1$ . Hence the instance  $\Pi(\nu, \kappa, \tau)$  would have been solved by an  $m$ -call, a contradiction to our assumption  $last(\rho(n, k, t)) = s$ .  $\square$

**Corollary 4.1** *If  $|\rho(n, k, t)| \geq 1$ , then  $last(\rho(n, k, t)) \in \{ge, go\}$ .*  $\square$

#### 4.1 Case: $2n > t$ and $t$ odd

From  $2n > t$  we can conclude  $t > 2(t - n - 1)$ . Using (25) and (27) we get  $t' > 2n'$ . This leads to

**Lemma 4.3** *Let  $\Pi(n, k, t)$  be a problem instance with  $2n > t$ ,  $t$  odd and  $\rho'(n, k, t) = \alpha go\beta$ ,  $\alpha \in \Omega^*$ ,  $\beta \in \Omega'^+$ , then*

*a)  $first(\beta) = m$ , if  $|\beta| = 1$ ,*

*b)  $first(\beta) = s$ , if  $|\beta| \geq 2$ .*  $\square$

Thus, after the case  $go$  the recursion ends by call of the meander algorithm or the recursion continues with the  $s$  case either.

**Corollary 4.2** *Let  $\Pi(n, k, t)$  be a problem instance with  $2n > t$  and  $t$  odd, then*

$$|\rho(n, k, t)|_s \geq |\rho(n, k, t)|_{go}. \quad (37)$$

#### 4.2 Case: $2n > t$ and $t$ even

From (19) it follows immediately

$$|\rho(n, k, t)|_{ge} \leq \log t = \log \frac{n(n+1)}{2k} \quad (38)$$

### 4.3 Case: $2n \leq t$

In this case if the algorithm performs the instance  $\Pi(n, k, t)$ , then the next instance to solve may be  $\Pi(n', k, t')$  with  $n' = n - 2k$  and  $t' = t - 2(n - k) - 1$  (cf. Subsection 3.2, equations (32) and (34), respectively). By  $n^{(\ell)}$  and  $t^{(\ell)}$  we denote the value of  $n$  and  $t$  in the  $\ell^{\text{th}}$  recursion call in the case  $2n^{(\ell)} \leq t^{(\ell)}$ . Thus we have  $n^{(0)} = n$ ,  $n^{(1)} = n' = n - 2k$  and  $t^{(0)} = t$ ,  $t^{(1)} = t' = t - 2(n - k) - 1$ , for example. By induction we get

$$n^{(\ell)} = n - 2k \cdot \ell \quad (39)$$

$$\begin{aligned} t^{(\ell)} &= t - 2n \cdot \ell + 2k \cdot \ell^2 - \ell \\ &= t - (2(n - k \cdot \ell) + 1) \cdot \ell \end{aligned} \quad (40)$$

Now we determine the order of the maximum value of  $\ell$  guaranteeing the condition  $2n^{(\ell)} \leq t^{(\ell)}$ . Using (39) and (40) we get

$$0 \leq t^{(\ell)} - 2n^{(\ell)} \quad (41)$$

$$= t - (2(n - k \cdot \ell) + 1) \cdot \ell - 2(n - 2k \cdot \ell) \quad (42)$$

To determine  $\ell$  we solve the quadratic equation

$$0 = \ell^2 + \frac{4k - 2n - 1}{2k} \cdot \ell + \frac{t - 2n}{2k} \quad (43)$$

which has the solutions

$$\ell_{1,2} = -\frac{4k - 2n - 1}{4k} \pm \sqrt{\left(\frac{4k - 2n - 1}{4k}\right)^2 - \frac{t - 2n}{2k}} \quad (44)$$

$$(45)$$

$$= -\frac{4k - 2n - 1}{4k} \pm \frac{4k - 1}{4k} \quad (46)$$

i.e.

$$\ell_1 = \frac{n}{2k}, \quad \ell_2 = \frac{n+1}{2k} - 2 \quad (47)$$

Finally we get

$$\ell \leq \frac{n}{2k} \quad (48)$$

Thus, we have just proven

**Lemma 4.4** *Let  $\Pi(n, k, t)$  be a problem instance. If  $\rho(n, k, t) = s^\ell x$  with  $x \in \{ge, go\}$ , then  $\ell \leq \frac{n}{2k}$ .  $\square$*

Corollary 4.2, inequality (38) and Lemma 4.4 lead to

**Theorem 4.1** *Let  $\Pi(n, k, t)$  be a problem instance.*

*a) Then the recursion depth of  $\Pi\text{Solve}(n, k, t)$  is  $\mathcal{O}\left(\frac{n}{2k} + \log \frac{n(n+1)}{2k}\right)$ .*

*b) Since the complexity of operations the algorithm performs in each recursion call (assigning elements of  $I_n$  to some set  $T_j$ , arithmetic comparisons and operations) is  $\mathcal{O}(n)$  it follows that  $\Pi\text{Solve}$  needs*

$$\mathcal{O}\left(n \cdot \left(\frac{n}{2k} + \log \frac{n(n+1)}{2k}\right)\right) \quad (49)$$

*steps to insert the  $n$  elements of  $I_n$  into the  $k$  sets  $T_j$ .* □

## 5 Conclusion

In Section 3 we present the recursive algorithm  $\Pi\text{Solve}$  which solves following special PARTITION problems  $\Pi(n, k, t)$ : Given  $n, k, t \in \mathbb{N}$  with  $t \geq n$  and  $\Delta_n = k \cdot t$ , then the algorithm partitions the set  $I_n = \{1, \dots, n\}$  into  $k$  mutually disjoint sets such that the elements in each set add up to  $t$ . The recursion can be stopped, if  $n$  is even and  $2k$  is a divisor of  $n$  or if  $n$  is odd and  $2k$  is a divisor of  $n + 1$ , respectively, because in these cases the meander algorithms presented in Section 2 can be applied, which directly determines a partition.

We prove that the algorithm works correctly and needs

$$\mathcal{O}\left(n \cdot \left(\frac{n}{2k} + \log \frac{n(n+1)}{2k}\right)\right) \quad (50)$$

steps to assign the elements of  $I_n$  to the  $k$  subsets  $T_j$  for each problem instance  $\Pi(n, k, t)$ . Taking into account that the algorithm for the inputs  $n$  and  $k$  determines an output consisting of  $k$  sets to which the elements of  $I_n$  are to be distributed so that all constraints are met,  $\Pi\text{Solve}$  is a polynomial output-sensitive time algorithm.

In Jagadish (2015) an approximation algorithm for the cutting sticks-problem is presented. Because the cutting sticks-problem can be transformed into an equivalent partitioning problem our algorithms can be applied to the corresponding cutting sticks-problems.

Further research may investigate whether ideas from the previous chapters and cited papers can be used to improve the efficiency of the  $\Pi\text{Solve}$ -algorithm. In Büchel et al. (2016), Büchel et al. (2017a) and Büchel et al. (2017b) we present efficient solutions for problem instances  $\Pi(n, k, t)$ , where  $n = q \cdot k$ ,  $q, k$  odd;  $n = m^2 - 1$ ,  $m \geq 3$ ;  $n = p - 1$ ,  $n = p$ ,  $n = 2p$ ,  $p \in \mathbb{P}$ , where  $\mathbb{P}$  is the set of prime numbers. Thus we may augment the  $\Pi\text{Solve}$ -algorithm by related conditions to stop further recursion calls.

## Acknowledgements

We would like to thank Arkadiusz Zarychta, a member of the ADIMO group as well, who created a tool by means of which we are able to test the algorithm and to analyse experimentally its performance.

Furthermore we would like to thank the reviewers for their valuable comments leading to improvements of the presentations.

## References

- K. Ando, S. Gervacio, and M. Kano. *Disjoint Subsets of Integers having a constant Sum*. Discrete Mathematics 82, 7-11, 1990.
- A. Büchel, U. Gilleßen, and K.-U. Witt. *Betrachtungen zum Cutting sticks-Problem (in German)*. Technical Report 01-2016, Department of Computer Science, Bonn-Rhein-Sieg University of Applied Sciences, 2016.
- A. Büchel, U. Gilleßen, and K.-U. Witt. *Ansätze für effiziente Lösungen von Cutting sticks-Problemen und deren Charakterisierung (in German)*. Technical Report in preparation, Department of Computer Science, Bonn-Rhein-Sieg University of Applied Sciences, 2017a.
- A. Büchel, U. Gilleßen, and K.-U. Witt. *Effiziente Lösungen von Spezialfällen des Cutting sticks-Problems (in German)*. Technical Report in preparation, Department of Computer Science, Bonn-Rhein-Sieg University of Applied Sciences, 2017b.
- F.-L. Chen, H.-L. Fu, Y. Wang, and J. Zhou. *Partition of a Set of Integers into Subsets with prescribed Sums*, pages 629 – 638. Taiwanese Journal of Mathematics 9, 2015.
- H.-L. Fu and W.-H. Hu. *A Special Partition of the Set  $I_n$* . Taiwanese Journal of Mathematics 9, 2015.
- M. Jagadish. *An Approximation Algorithm for the Cutting-Sticks Problem*, pages 170 – 174. Information Processing Letters 115, 2015.
- H. J. Straight and P. Schillo. *On the Problem of Partitioning  $\{1, \dots, n\}$  into Subsets having equal Sums*, pages 229 – 231. Proceedings of the American Mathematical Society 74(2), 1979.