Recover origin of pair differences - non perfect rulers (Draft)

Håkan Kjellerstrand (hakank@gmail.com, http://hakank.org/)

Abstract

This paper describes a method to recover the origin (generating array/list) of a list of pair differences that was generated from the origin list using constraint modelling. There are three different variants:: a) the origin and the differences has no duplicates, b) the origin has some duplicates and the difference list indicates this with a single difference of 0 (zero), and c) the origin has duplicates and the difference list contains all the possible ((n*(n-1) div 2) difference pairs. To recover the exact origin list is not possible, but here we show how to recover a list that is similar in structural ways, a shifted list. We also discuss perfect rulers and their relation to the set of the possible (solvable) difference lists: “non perfect rulers”.

Constraint modelling model

All results/findings in this report were found by experiments using the Picat model http://hakank.org/picat/linear_combination.pi .There is also a PEQNP model with some of the experiments: http://hakank.org/peqnp/linear_combinations.py .

Origin and description of the problem

This paper and the corresponding model was inspired by the following tweet from Simone Sturniolo (https://twitter.com/SturnioloSimone/status/1214588457135329280):

Suppose there's a certain set of numbers - x_1, x_2, ..., x_n. You don't know n. Suppose you only know certain linear combinations of them, L_ij = x_i-x_j, but you DON'T know i or j. What's an algorithm to find as many of the xs as possible just from looking at the Ls? Ideas?

and a follow up tweet:

There are four numbers. I can tell you that 1, 3, and 6 are differences between pairs of numbers chosen among them (you don't know which pairs). Can you tell me what the numbers are? Something like that, except you don't know it's four.

To summarize the problem: **Given a list of pair differences try to recover the list that generated that list. The challenge is that the values in the generating list are unknown and the length of the generating list is unknown.**

Note: This description in this paper is the one that I (hakank) has interpreted from the tweets above. Perhaps the model - with all its assumptions and symmetry breakings etc - is not that useful.

Picat model

The model that this paper describes is the Picat model http://hakank.org/picat/linear_combinations.pi (the name is perhaps not the best but it has been stuck since it's used in difference tweets etc).

Picat - http://picat-lang.org/ - is a multi-paradigm language which supports

- logic programming (Picat is written in B-Prolog, C, and Picat)
- constraint modelling using different kind of solver: constraint programming, SAT, MIP, and SMT (z3 and cvc4),
- planning
- tabling
- functions (though not a fully functional programming language)
- imperative constructs such as for loops, while loops, reassignments

For more about Picat, see the book "Constraint Solving and Planning with Picat" by Neng-Fa Zhou, Håkan Kjellerstrand, Jonathan Fruhman, Springer, 2015. It is freely downloadable from http://picat-lang.org/picatbook2015.html .

PEQNP model

During the development of this Picat model, an experimental PEQNP model was also implemented with some of the features of the Picat model: http://hakank.org/peqnp/linear_combinations.py

For more about PEQNP see https://github.com/maxtuno/PEQNP/ .

Dictionary of terms used

This paper use quite a few different types of lists. Here are the central types of lists used in this report:

- **List difference**: Given a list L, the list difference is the list `[L[2]-L[1],..L[N]-L[N-1]]`.

- **Pair differences**: Given a list L of length Len, the pair differences is the list of length (Len*(Len-1) div 2): `[abs(L[I]-L[J]) : I in 1..Len, J in I+1..Len].`
- **Difference list**: This is the input of a problem instance, this is a list of **pair differences** generated from an unknown list, the **origin list**.
- **Origin list**: The unknown list that generates the input to a problem instance, the difference list.
- **Shifted list**: Given a list L of length Len, the **shifted list** L' is the conversion `L'= [L[I]-L[1]+1 for I in 1..Len]`. A shifted list always starts with `L'[1]=1` and ends with `L'[Len] = max(L)-L[1]+1`. Thus the shifted list L' is a "structural similar" - perhaps one could call this **normalized** - version of list L.
- **Solution**: The solution of a problem with the input of a difference list D is a recovered **shifted version** of the **origin list**. For a given difference list there might be (many) different solutions, where some of these solutions has symmetries in their list differences.

Sorry if it gets a bit messy with different type of differences and differences of list differences etc. Hopefully the examples will clarify these concepts.

Example

Here is a simple example: Given the difference list [1,2,3], one solution would be [1,2,4], and another solution would be [1,3,4]. However, the same difference list could have been generated by [11,12,13] or [99,100,101] etc.. Thus, **it is not possible to guarantee that the solution is the exact origin list, just that it is a structural similar list**.

Another example is the difference list [4,9,11,13,15,16,20,24,29,31,33,35,44] which - as it happens - was generated by the origin list [5,14,18,34,38,49]. This list is structural similar to the **shifted list** [1,10,14,30,34,45], i.e. a list that starts with 1 and all numbers are subtracted by 5 (the first element in the list). In fact there is a (huge) family of origins with the same difference list.

There are 2 different solutions to this problem instance:

- [1,12,16,32,36,45]
- [1,10,14,30,34,45]

Note that the second solution is exactly the same as the shifted origin list.

Why these two solutions? These two solutions might not make much sense, but if we instead look at the **list differences** ([x[2]-x[1], x[3]-x[2],...,x[n]-x[n-1]]), we see that that these differences are the reverse of each other:

- [1,12,16,32,36,45] has the list difference of [11,4,16,4,9]

- [1,10,14,30,34,45] has the list difference of [9,4,16,4,11], the reverse.

Now, since we are talking about "structural" solutions, the origin could have been any list
- [1,10,14,30,34,45] + n or
- [1,12,16,32,36,45] + n

where n is any random integer (>= 0).

Here are some examples of possible origin lists :

[1,10,14,30,34,45]
[2,11,15,31,35,46]
[3,12,16,32,36,47]
[4,13,17,33,37,48]
[5,14,18,34,38,49]
[6,15,19,35,39,50]
[7,16,20,36,40,51]
[8,17,21,37,41,52]
[9,18,22,38,42,53]
[10,19,23,39,43,54]
[11,20,24,40,44,55]
….

Or
[1,12,16,32,36,45]
[2,13,17,33,37,46]
[3,14,18,34,38,47]
[4,15,19,35,39,48]
[5,16,20,36,40,49]
[6,17,21,37,41,50]
[7,18,22,38,42,51]
[8,19,23,39,43,52]
[9,20,24,40,44,53]
[10,21,25,41,45,54]
[11,22,26,42,46,55]
….

Assumptions and constraints: base model

In this paper there are certain assumptions and these assumptions are converted to constraints in the constraint model.

- **Input**: A list - difference list - of distinct numbers sorted in increasing order - representing the absolute values of the pair differences generated from an **unknown** origin list of **unknown length**. For the base variant, we assume that there are no duplicates in the origin list and there are thus no duplicates in the difference list. This input list is called "diffs" or `Diffs` in examples or code segments.

- The largest value in the solution is 1 + the maximum value of the difference list (= the last value of the difference list). The reason for this is that if M is the maximum value in the difference list then 1+M is the largest possible value in the solution.

- The **solution** is a list in strict increasing order with a domain of integers from 1 to max(Diff)+1.

- The solution always starts with 1.

In Picat, these constraints are modelled as follows (with simplifications with the full model). Some of the constraints are discussed later in the paper.

```
X = new_list(N), % create a list of N decision variables
X :: 1..max(Diff)+1, % the domain of the decision variables is
1..max(Diff)+1
X[1] #= 1,  % first element is always 1

all_distinct(X), % the values variables are distinct

increasing_strict(X), % the values are sorted in strict increasing
order


% There is a symmetry two solutions has the same list differences
% (X[2]-X[1],...X[N]-X[N-1]), but one is the reversal of the other.
X[2]-X[1] #>= X[N]-X[N-1],

% From the observation that
%   the sum of the differences of the solutions
% is equal to
%   the sum of the list of the differences + the first element
%   in (the sorted list) Diffs
sum([X[I]-X[I-1] : I in 2..N]) #= sum([Diffs[I]-Diffs[I-1] : I in
2..Diffs.len]) + Diffs[1],
```

Now we come to the more interesting constraints regarding the connection between the difference list and the solution list (called `X`):

- For all the elements D in the difference list Diffs, there must be a pair `(X[I], X[I], I < J)` where `abs(X[I]-X[J]) = D`.
- There are no other pair difference in the solution X than those that are in the difference list.

To summarize: The difference list (sorted and with duplicates removed) and the pair differences of the solution (sorted and with duplicates removed) must be identical.

These two constraints are modelled as followed in Picat:

```
%
% All the differences from the pairs in the list X must be in
% the difference list Diff
%
  Ts = [],
  foreach(I in 1..N, J in I+1..N)
     T #= abs(X[I]-X[J]),
     T :: Diffs,
     Ts := Ts ++ [T]
  end,

%
% Ensure that we cover all the differences (in Diffs)
%
IJs = [],
foreach(D in Diffs)
   % Create the indices I,J where I < J
   I :: 1..N,
   J :: 1..N,
   I #< J, % symmetry breaking

   element(I,X,Ix), % Ix = X[I]
   element(J,X,Jx), % Jx = X[J]
   % Ensure that this difference is covered
   D #= abs(Ix-Jx),
   IJs := IJs ++ [I,J] % add to the variables to be solved
end,

% Solve
Vars = X ++ IJs ++ Ts,
solve($[ffd,updown],Vars).
% End of model
```

Later we will loosen some of these constraints, e.g.

- There might be a 0 (zero) in the difference list, indicating that the solution list has at least one duplicate
- There might be duplicates (including 0s) in the difference list, indicating that the solution list also has duplicate values. In this later case, the number of occurrences of duplicates in both lists must match.

Is it possible to find the real origin list?

In general it is not possible to restore the origin list that generated the differences, just a solution that satisfies the constraints above. Or at least: there is no way to know that the origin is found. The solution given by this model is **structural similar** to the “normalized” version of the origin list, the **shifted list** as was mentioned above.

Not all difference lists are possible

The example that Simone Sturniolo showed in the tweet (see above), namely [1, 3, 6], is in fact not a possible difference list. Here is the reasoning.

a) A (symmetry) assumption about the solution (origin list) is that it always starts with 1.

b) Since there is a difference of 6 in the difference list, then the origin list must also have 7 (since 7-1 = 6). The origin list thus have the elements [1,7].

c) The number left to take care of the difference of 3: There must be a two numbers, X and Y, such that X-Y = 3. Since the existing list of difference are 1 and 7 there are two possibilities:

   1) Try 7-Y = 3: give Y = 4 which is not in the difference list so this is not possible.
   2) Try X-1 = 3: give X = 4 which is not in the difference list so this is not possible.

   Any X and Y for which X-Y = 3 must give at least one number X or Y not in the difference.

QED: The difference list [1,3,6] is not possible.

Here is a list of all possible difference lists of length 3 (“diffs”) with a domain of 1..10 having a valid solution (“sol”). There are 120 possible lists, but only 20 of them are valid difference lists (looking for the first possible solution).

```
[diffs = [1,2,3],sol = [1,3,4]]
[diffs = [1,3,4],sol = [1,4,5]]
[diffs = [1,4,5],sol = [1,5,6]]
[diffs = [1,5,6],sol = [1,6,7]]
[diffs = [1,6,7],sol = [1,7,8]]
[diffs = [1,7,8],sol = [1,8,9]]
[diffs = [1,8,9],sol = [1,9,10]]
[diffs = [1,9,10],sol = [1,10,11]]
[diffs = [2,3,5],sol = [1,4,6]]
[diffs = [2,4,6],sol = [1,5,7]]
[diffs = [2,5,7],sol = [1,6,8]]
[diffs = [2,6,8],sol = [1,7,9]]
[diffs = [2,7,9],sol = [1,8,10]]
[diffs = [2,8,10],sol = [1,9,11]]
[diffs = [3,4,7],sol = [1,5,8]]
[diffs = [3,5,8],sol = [1,6,9]]
[diffs = [3,6,9],sol = [1,7,10]]
[diffs = [3,7,10],sol = [1,8,11]]
[diffs = [4,5,9],sol = [1,6,10]]
[diffs = [4,6,10],sol = [1,7,11]]
```

Note: For the difference list [1,2,3] there is also the solution [1,2,3,4] but it is not counted since there were a solution with a shorter length.

Here are some statistics for length 3 (and 4) difference lists with a certain domain (1..Max) which shows the number of total possible lists (of length 3 and 4 with domain 1..Max), the number of valid difference lists and the percentage of possible difference lists divided by the total number of possible lists.

```
Len  Max  #lists    #difflists %(diff/poss)
-----------------------------------------
 3   3     1     1   1.0000
 3   4     4     2   0.5000
 3   5     10    4   0.4000
 3   6     20    6   0.3000
 3   7     35    9   0.2571
 3   8     56    12   0.2143
 3   9     84   16   0.1905
 3  10     120  20   0.1667
 3  11     165  25   0.1515
 3  12     220  30   0.1364
 3  13     286  36   0.1259
 3  14     364  42   0.1154
 3  15     455  49   0.1077
 3  16     560  56   0.1000
 3  17     680  64   0.0941
 3  18     816  72   0.0882
 3  19     969  81   0.0836
 3  20   1140   90   0.0789
 3  30   4060   210   0.0517
 3  40   9880   380   0.0385
 3  50  19600   600   0.0306
 3 100 161700  2450   0.0152
```

For lists and possible difference lists of length 4

```
Len  Max  #lists    #difflists %(diff/poss)
-------------------------------------------
 4   4     1    1   1.0000
 4   5     5    3   0.6000
 4   6     15   4   0.2667
 4   7     35   7   0.2000
 4   8     70   10   0.1429
 4   9     126  13   0.1032
 4  10     210  17   0.0810
 4  11     330  22   0.0667
 4  12     495  26   0.0525
 4  13    715   32   0.0448
 4  14   1001   38   0.0380
 4  15   1365   44   0.0322
 4  16   1820   51   0.0280
 4  17   2380   59   0.0248
 4  18   3060   66   0.0216
 4  19   3876   75   0.0193
 4  20   4845   84   0.0173
 4  30  27405   200   0.0073
 4  40  91390   367   0.0040
 4  50 230300   584   0.0025
 4 100 3921225  2417   0.0006
```

The probability of finding a valid difference list decreases when the max value increases; perhaps not really surprising.

Length of the solution

In the problem statements there is no information about the length of the origin list: the length of the solution must be calculated and then tried.

The model finds a solution with a shortest possible length by looping through the possible lengths and if it finds a solution it is the shortest solution. Note that there might be many possible solutions of the same length.

What is the possible **minimum length** of a solution given a specific difference list? Let us first observe that the number of pair differences (including duplicates) from a list L with the length Len is

NumberOfPairs = (Len(Len-1)) div 2*

However, we don't have the length of the origin list. What we do have is the pair differences (the input list of the problem) although the duplicates are removed so we can only state some minimum length:

*MinLen = floor(1+sqrt(1+8*NumberOfPairs)/2),*

Given an origin list L with length Len, the model normally find a solution of length Len and many times even shorter.

Thus, the loop will start with the N = MinLen and increase until a solution is found.

An example:

The following origin list
L = [16,22,27,28,36,37,41,50,60,63,64,73,78,84,87,91,93,94]
with Len=18

yield these 69 (distinct) pair differences

[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,41,42,43,44,45,46,47,48,50,51,52,53,54,55,56,57,58,59,60,62,63,64,65,66,67,68,69,71,72,75,77,78]

The minimum solution length (N) is thus

N = floor(1+sqrt(1+8*69)/2) = 12

The loop thus starts with N=12 and try to find a solution of this length, if not it increase N by 1 and continues, . Here is the output of the run:

```
n = 12
n = 13
n = 14
n = 15
x = [1,2,4,10,15,17,22,38,46,47,60,69,70,73,79]
```

I.e. it tries N=12, 13 and 14 without any luck, but then find a solution with length 15. Note that the origin list was of length 18 so here we found a is a shorter solution.

Finding the exact (mathematical) mechanism why there are solutions of shorter lengths in some cases and not in another is an open question. One can observe that the “extremal” case where the difference lists of 1..<some M> give large “compressions”.

Number of solutions - perfect and non-perfect rulers

Many - probably most - random instances has just 2 solutions. However, that are certain “extremal” cases where there are many solutions, the instances where the differences are all numbers from 1 to Len.

Here is the number of solutions found for Len = 1..21, i.e. where the difference lists are all the numbers 1..Len, i.e. for perfect rulers.

N	#solutions for diffs=1..N
1	1
2	1
3	1
4	2
5	3
6	4

```
7    2
8   12
9    8
10    4
11   38
12   30
13   14
14    6
15  130
16   80
17   32
18   12
19  500
20  326
21  150
```

Using OEIS (On-line Encyclopedia of Integer Sequences, http://oeis.org/) it can be found that the integer sequence 1,1,2,3,4,2,12,8,4,38,30,14,6,130,80,32,12,500,326,150 is - probably not surprisingly - the OEIS sequence A103300 "Number of perfect rulers with length n (n>=0)", (https://oeis.org/A103300).

A perfect ruler (https://en.wikipedia.org/wiki/Perfect_ruler) is a sub-problem of the problem we study, i.e. the case where the differences are a list of all numbers from 1..Len, and without duplicates. For more about perfect rulers, see: http://mathworld.wolfram.com/PerfectRuler.html

A related problem is Golomb ruler: https://en.wikipedia.org/wiki/Golomb_ruler which requires that the solution is optimal; this is not handled in this model (the minimization mode is not really the same as Golomb ruler). Perhaps this will be explored in the future. (And somewhere in this Wiki post might be some more good ideas to check out: https://oeis.org/wiki/User:Peter_Luschny/PerfectRulers .)

Since the set of perfect rulers is just a subset of the problem domain, the target sequences we study might then be called **non-perfect rulers**.

Invariant: sum of the differences

For some problem instances there might be a huge number of solutions. For example the problem that is generated by the origin list [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15] has 130 solutions, e.g.

    x = [1,2,6,9,11,13,15]
    x = [1,2,4,7,11,14,15]

x = [1,3,5,6,8,14,15]
x = [1,3,5,7,10,14,15]
x = [1,2,5,6,8,13,15]

Interestingly, the sum of the list differences are all 14.

Another example: The number of solutions for the generator array of 1..20 is 326, and the sum of list differences are all 19, i.e. one less than the number of elements.

A generator of [1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39] also give 326 solutions, with the sum of list differences of 38.

The following is thus an invariant: The sum of the list differences in the difference list is the same as the sum of the list differences in the solution.

Invariant: sum of list differences

It also seems that the sum of list differences is an invariant: the sum of the list differences in the solution is the same as the sum of the list differences of the generator array,

For example, the generator array of [16,41,43,54,56,62,66,67,84,95] has the sum of list differences of 79, which is the same as for the 2 solutions:

```
     l = [16,41,43,54,56,62,66,67,84,95]
     l_len = 10
     l_shifted = [1,26,28,39,41,47,51,52,69,80]
     l_diffs = [25,2,11,2,6,4,1,17,11]
     l_diff_sum = 79
     diffs =
[1,2,4,5,6,8,10,11,12,13,15,17,18,19,21,22,23,24,25,26,27,28,29,30,33
,38,39,40,41,43,46,50,51,52,54,68,79]
     diffLen = 37
     mode = all
     n = 9
     n = 10
     x = [1,12,29,30,34,40,42,53,55,80]
     x_shifted = [1,12,29,30,34,40,42,53,55,80]
     x_diffs = [11,17,1,4,6,2,11,2,25]
     sum_x_diffs = 79
     check =
[1,2,4,5,6,8,10,11,12,13,15,17,18,19,21,22,23,24,25,26,27,28,29,30,33
,38,39,40,41,43,46,50,51,52,54,68,79]
```

```
      x = [1,26,28,39,41,47,51,52,69,80]
      x_shifted = [1,26,28,39,41,47,51,52,69,80]
      x_diffs = [25,2,11,2,6,4,1,17,11]
      sum_x_diffs = 79
      check =
[1,2,4,5,6,8,10,11,12,13,15,17,18,19,21,22,23,24,25,26,27,28,29,30,33
,38,39,40,41,43,46,50,51,52,54,68,79]
```

Relation between sum of differences of generating array, the differences and the solution

Another invariant is the relation between the sum of the differences of the generating array (which as we saw above is the same as the sum of differences of the solution) and the sum of the differences of the differences array (i.e. the input array of the problem).

The invariant (and constraint) is:
the sum of the differences of the origin list == the sum of the differences of the difference array + the first element in the difference list

For example:

```
      l = [27,29,39,44,50,52,61,64,67,86]
      l_len = 10
      l_diffs = [2,10,5,6,2,9,3,3,19]
      l_diff_sum = 59
      diffs =
[2,3,5,6,8,9,10,11,12,13,14,15,17,19,20,21,22,23,25,28,32,34,35,36,37
,38,40,42,47,57,59]
      diffs_diff_sum = 57
      difference_of_diff_sums = 2

      i.e. l_diff_sum = diffs_diff_sum + diffs[1] = 57 + 2 = 59
      which is the same as
            x_diff_sum + diffs[1] = 59
```

This is coded in Picat as:

```
sum([X[I]-X[I-1] : I in 2..N]) #= sum([Diffs[I]-Diffs[I-1] : I in
2..Diffs.len]) + Diffs[1],
```

Allowing duplicates, part I

The above experiments has been on origin lists with strictly distinct numbers, Now we will loopen this restriction and do some experiments where duplicates are allowed in the origin list, which means that the difference list contains a 0. Not that we yet don't allow any duplicates in the difference list: all duplicates created by the origin list are summed up by the single 0 difference in the difference list.

Below in an example where the origin list ([2,4,7,7,8,12,17,18,20,21,23,23,25,29,30,36,37,37,38,38]) has a couple of duplicates:

- 7, 23, 37, and 38

Note that there are only one 0 which "collects" all these four duplicates. As a result, the solution will also only contain one duplicate, here 37. When we don't allow duplicates, the length of the solutions is often near the length of the generating list (apart from some extreme cases where it's much shorter).

With duplicates the solution list is always much shorter.

```
   l = [2,4,7,7,8,12,17,18,20,21,23,23,25,29,30,36,37,37,38,38]
   l_len = 20
   diffs =
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36]
   n = 9
   n = 10
   n = 11
   x = [1,2,4,7,14,21,28,32,36,37,37]
   x_shifted = [1,2,4,7,14,21,28,32,36,37,37]
   x_diffs = [1,2,3,7,7,7,4,4,1,0]
   sum_x_diffs = 36
   check =
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36]
```

A little more extreme example is the generating list of 10 pairs of duplicates:

[1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10]

with the difference list

[0,1,2,3,4,5,6,7,8,9]

The solution compresses down to the problem of a difference list of 0,1..9, and is just 6 elements with one duplicate: [1,2,5,8,10,10].

The Picat code handling this is the condition of a flag `AllowDuplicates = true`: were the constraint that the solution X must be in strict increasing order is replaced by the constraint that X must be sorted in (non-strict) increasing order.

```
if AllowDuplicates != false, Diffs[1] == 0 then
   increasing(X)
else
   increasing_strict(X)
end,
```

Allowing duplicates, part II

In this experiment we don't remove any duplicates in the difference list. Compare with the previous experiment, the single 0 in the difference list (which indicated all duplicates) is now replaced by all the duplicates of the pairs differences.

The number of difference in this variant will thus be the full value of

NumDiffs = (Len*(Len-1)) div 2

where Len is the length of the origin list. E.g. for Len=20 then NumDiffs = 20*19 div 2 = 190.

Here is an example of this experiment:

- Origin list: [7,10,11,12,12,12,13,15,15,17]
  Note that we have a couple of duplicates: 12,12,12 and 15,15

- Difference list is thus:
  [0,0,0,0,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,5,5,5,5,5,5,5,5,6,6,7,8,8,10]

- Solution:
  [1,4,5,6,6,6,7,9,9,11]
  and it has the same differences as above:
  [0,0,0,0,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,5,5,5,5,5,5,5,5,6,6,7,8,8,10]

Note that the solution has as many duplicates as the generating list, though probably not exactly the same numbers, here 6,6,6 and 9,9.

The additional constraint in the model is that the list of differences of the solution (X) must have exactly the same number of occurrences as in the difference list, i.e.

- 4 0s
- 6 1s
- 8 2s
- etc.

The Picat code for this is the following constraint which is active if the `AllowDuplicates` is set to `Experimental`:

```
foreach(T in 0..Max)
  sum([abs(X[I]-X[J])#=T : I in 1..N, J in I+1..N]) #= count(T,Diffs)
End,
```

This constraint is called *global cardinality count*, but in this case, the explicit loop seems to be faster. Note that this extra constraint might slow down the model.

Lengths of difference list and the solution lists

This is another experiments on “extremal” difference lists, i.e. the perfect rulers with difference lists that contains all the integers from 1..Len (for some length Len).

The normal procedure of the experiments is that we try to find one solution (or all solutions) of the shortest possible length. However, there are solutions beyond the shortest solution length. This is what we tests here.

The number of solutions for a certain length (len) of the difference lists and the number of solutions for certain length of solutions:

```
[len = 1,counts = [1]
[len = 2,counts = [1]
[len = 3,counts = [2,1]
[len = 4,counts = [3,1]
[len = 5,counts = [4,4,1]
[len = 6,counts = [2,9,5,1]
[len = 7,counts = [12,14,6,1]
[len = 8,counts = [8,27,20,7,1]
[len = 9,counts = [4,40,48,27,8,1]
```

```
[len = 10,counts = [38,90,75,35,9,1]
```

For example, for the difference list length of 9, here is a count of the number of solutions for different N:

```
len = 9
diffs = [1,2,3,4,5,6,7,8,9]
lens = [5,10]
5 = 4
6 = 40
7 = 48
8 = 27
9 = 8
10 = 1
counts = [len = 9,counts = [4,40,48,27,8,1]]
```

I.e. there were 4 solutions of length 4 (the shortest length), 40 solutions of length 6 etc. Note That there is only one single solution of length N+1, and there are N-1 solutions of length N.

All these sequences are clumped together in the OEIS sequence https://oeis.org/A103294 *Triangle T, read by rows: T(n,k) = number of complete rulers with length n and k segments (n >= 0, k >= 0).*

Again, note that this experiment is only on complete rulers, which is only a very special case of the problem we study, i.e. recovering a list based on its differences. These difference lists often are non-contiguous and thus not “extremal” (perfect rulers).

The **total number of solutions** for a certain difference length (1..10):
1,1,3,4,9,17,33,63,128,248

This is the (expected) OEIS list https://oeis.org/A103295: *Number of complete rulers with length n.*

Experiments in the Picat model

In the Picat model http://hakank.org/picat/linear_combinations.pi there are quite a few experiments. This final section discuss the experiments and the code.

go/0: Generate all shortest solutions from a given difference list

This takes a difference list and then searching for the shortest solution. Here is a short and commented version of this experiment.

- `L = generate_problem(5,20),`
  Generate a origin list with at most 5 elements (duplicates are removed hence it can be 3 or 4 elements). The maximum value in the list is 20.

- `Diffs = diffs(L)`
  Create a list of the pair differences given the origin list L.

-
  ```
  Mode = all,
  AllowDuplicates = false,
  do_experiment(L, Diffs, Mode, AllowDuplicates)
  ```
 Run the experiment.

The `do_experiment/4` predicate is the workhorse of an experiment. It runs the experiment, print out solutions: it checks that is really a solution by comparing the input difference list with the difference list generated by the solution . The parameters of this predicate are

- `L`: the origin list
- `Diffs`: The difference list
- `Mode`: The "mode" of the experiment. The different modes are:
  `first`: Find a solution of the shortest length.
  `all`: Find all the solutions of the shortest length
  `minimize`: (experimental) search for a solution of shortest length with the smallest sum of list differences. Note: This has not been discussed in this paper.

This first experiment:

- Randomize a short origin list and create the difference list
- Search for a shortest solution.

Example:

```
l = [5,12,13,15,19]
l_len = 5
l_shifted = [1,8,9,11,15]
l_diffs = [7,1,2,4]
l_diff_sum = 14
diffs = [1,2,3,4,6,7,8,10,14]
diffLen = 9
diffs_diff_sum = 13
difference_of_diff_sums = 1
differences_of_diffs_sums2 = 0
missing_from_diffs = [5,9]
num_missing_from_diffs = 2
mode = all
```

```
n = 5
x = [1,5,7,8,15]
x_diffs = [4,2,1,7]
sum_x_diffs = 14
check = [1,2,3,4,6,7,8,10,14]

x = [1,8,9,11,15]
x_diffs = [7,1,2,4]
sum_x_diffs = 14
check = [1,2,3,4,6,7,8,10,14]

num_solutions = 2
```

go2/0: Generate solutions for a difference list with duplicates, first version

This is same same as go/0 with the exception that it allows that there are duplicates in the origin list, and thus the difference list contains a (single) 0 that counts for all the duplicates in the orgin list. The flag `AllowDuplicates`changes the duplication behaviour of both the generated origin list as well as the constraints behaves.

This experiment just search for one solution (`Mode = first`).

```
L = generate_problem2(20,40),

AllowDuplicates = true,
% AllowDuplicates = experimental, % see comment below
Mode = first,
if AllowDuplicates == experimental then
   Diffs = diffs2(L)
else
   Diffs = diffs(L)
end,

do_experiment(L, Diffs, Mode, AllowDuplicates).
```

Here is a sample run:

```
l = [2,3,5,5,6,7,9,9,10,10,14,15,22,24,33,37,37,39,40,40]
l_len = 20
l_shifted = [1,2,4,4,5,6,8,8,9,9,13,14,21,23,32,36,36,38,39,39]
l_diffs = [1,2,0,1,1,2,0,1,0,4,1,7,2,9,4,0,2,1,0]
l_diff_sum = 38
```

```
diffs =
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38]
diffLen = 39
diffs_diff_sum = 38
difference_of_diff_sums = 0
differences_of_diffs_sums2 = 0
missing_from_diffs = [39]
num_missing_from_diffs = 1
mode = first
n = 9
n = 10
n = 11
n = 12
x = [1,2,6,16,20,27,30,33,36,38,39,39]
x_diffs = [1,4,10,4,7,3,3,3,2,1,0]
sum_x_diffs = 38
check =
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38]
```

go3/0: Generate solutions for a difference list with duplicates, second version

This experiment is almost the same as the one above (go2/0) but it works with the full list of pair duplicates. The main change is the flat `AllowDuplicate` is set to `experimental` (instead of `true`). Note: To force duplicates use a quite low max value, e.g length*2.

```
L = generate_problem2(10,20),

Mode = first,
if AllowDuplicates == experimental then
   Diffs = diffs2(L)
else
   Diffs = diffs(L)
end,

do_experiment(L, Diffs, Mode, AllowDuplicates).
```

Here is a sample run. Note the duplicates in the origin list (15 and 16) which generates two 0s in the difference list. The extra requirement for this version is that the solution then also must be constructed so it has two 0s in its pair difference list.

```
l = [4,5,6,12,13,15,15,16,16,17]
l_len = 10
l_shifted = [1,2,3,9,10,12,12,13,13,14]
l_diffs = [1,1,6,1,2,0,1,0,1]
l_diff_sum = 13
diffs =
[0,0,1,1,1,1,1,1,1,1,1,2,2,2,2,2,3,3,3,3,4,4,4,5,6,7,7,8,8,9,9,9,10,1
0,10,10,11,11,11,11,11,12,12,12,13]
diffLen = 45
diffs_diff_sum = 13
difference_of_diff_sums = 0
differences_of_diffs_sums2 = 0
missing_from_diffs =
[14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36
,37,38,39,40,41,42,43,44,45]
num_missing_from_diffs = 32
mode = first
n = 10
x = [1,2,2,3,3,5,6,12,13,14]
x_diffs = [1,0,1,0,2,1,6,1,1]
sum_x_diffs = 13
check =
[0,0,1,1,1,1,1,1,1,1,1,2,2,2,2,2,3,3,3,3,4,4,4,5,6,7,7,8,8,9,9,9,10,1
0,10,10,11,11,11,11,11,12,12,12,13]
```

go4/0: Simple test of difference lists without a known origin list

The above experiments was constructed by first generating a origin list that was the basis of the difference list which - in turn - was the input to the experimental run.

In go/4 we can experiment with difference lists without having any origin list, for example to check if there is a solution. Since the `run_experiment/4` predicate required an origin list we have to program this experiment on our own, though - to be honest - much of this was copy/pasted from `run_experiment/4`.

Here we check if there is any solution to the input of the difference list `[1,3,6]`. As we will see, there is no solutions (cf discussion in the section *Not all difference lists are possible*).

```
Diffs =  [1,3,6], % Not possible
println(diffs=Diffs),
Len = Diffs.len,
println(diffsLen=Diffs.len),

MinLen = floor(1+sqrt(1+8*Len)/2), % DiffLen = (N*(N-1)) div 2
Mode = first,
AllowDuplicates = false,
Found = false,
foreach(N in MinLen..MinLen*2, Found == false)
  println(n=N),
  if linear_comb(Diffs,N,Mode,AllowDuplicates, Sol), Sol \= [] then
    if Mode == all then
      foreach(XX in Sol)
         print_solution(Diffs,XX,AllowDuplicates)
     end,
     println(num_solutions=Sol.len)
    else
     print_solution(Diffs,Sol,AllowDuplicates)
    end,
    Found := true
  End
end,
nl.
```

The output of this experiment is quite simple:

```
diffs = [1,3,6]
diffsLen = 3
n = 3
n = 4
n = 5
n = 6
```

It just states that there are no solution for any origin length.

go5/0: Count the number of possible solutions for some “perfect rulers”

This experiment is the base for the section *Lengths of difference list and the solution lists* above. The code is not much different from what we have seen, just a different way of handling solutions.

go6/0: What possible difference lists are there?

This experiment is the code for the results in the section *Not all difference lists are possible* and as for go5/0 it’s mostly about house keeping the solutions.