

# Tug of War Problem with *Answer Set Programming – clingo* An Encoding

Claudio Cesar de Sá

Independent Researcher

## Road map of this presentation:

1. About the ASP (clingo) – previous presentation (done)
2. Requisites – previous presentation (done)
3. Tug of War (well knowed problem from competitive programming sites and contests)
4. A modelling in ASP
5. A solution in clingo
6. Using Python's code inside of ASP
7. Conclusions

## Road map of this presentation:

1. About the ASP (clingo) – previous presentation (done)
2. Requisites – previous presentation (done)
3. Tug of War (well knowed problem from competitive programming sites and contests)
4. A modelling in ASP
5. A solution in clingo
6. Using Python's code inside of ASP
7. Conclusions

**Attention: some background in logic and declarative language is recommended!**

# Tug of War Problem



shutterstock.com · 1095074072

Figura: Practical application of this problem

# Tug of War

From: <https://www.geeksforgeeks.org/tug-of-war/>  
and/or <https://www.codechef.com/problems/C0319TSH>

Given a set of  $n$  integers, divide the set in two subsets of  $n/2$  sizes each such that the difference of the sum of two subsets is as minimum as possible. If  $n$  is even, then sizes of two subsets must be strictly  $n/2$  and if  $n$  is odd, then size of one subset must be  $(n - 1)/2$  and size of other subset must be  $(n + 1)/2$ .

# Examples

From: <https://www.geeksforgeeks.org/tug-of-war/>

- ▶ Example 1: let given set be  $\{3, 4, 5, -3, 100, 1, 89, 54, 23, 20\}$ , the size of set is 10. Output for this set should be  $\{4, 100, 1, 23, 20\}$  and  $\{3, 5, -3, 89, 54\}$ . Both output subsets are of size 5 and sum of elements in both subsets is same (148 and 148).
- ▶ Let us consider another example where  $n$  is odd. Let given set be  $\{23, 45, -34, 12, 0, 98, -99, 4, 189, -1, 4\}$ . The output subsets should be  $\{45, -34, 12, 98, -1\}$  and  $\{23, 0, -99, 4, 189, 4\}$ . The sums of elements in two subsets are 120 and 121 respectively.
- ▶ This problem is beauty: *easy to understand, hard to solve it!*

# Comments

- ▶ Again: all the combinations must be found!
- ▶ Input: a set of numbers, in our implementation an array, aiming a possible repetitions of these numbers.
- ▶ Output: two sets with the same size/cardinality, or with difference of one number for set A or B.
- ▶ Complexity: NP-Complete (all the combinations must be examined) – Set partition problem is NP complete – <https://www.geeksforgeeks.org/set-partition-is-np-complete/>
- ▶ Optimization: NP-Hard, due the minimum value of the absolute difference between the sum of two sets.

## Some comments and motivation:

- ▶ I solved it in Minizinc!
- ▶ Some approaches for this problem can be taken: Simulated Annealing, Ant Colony, Depth-First Search, meta-heuristics, ... etc.
- ▶ Dynamic Programming (DP) is the most suitable for contest programming
- ▶ The full code discussed here is found in:  
[https://github.com/claudiosa/CCS/tree/master/asp\\_Answer\\_Set\\_Programming/tug\\_of\\_war.lp](https://github.com/claudiosa/CCS/tree/master/asp_Answer_Set_Programming/tug_of_war.lp)
- ▶ I will be commenting the modelling in parts



## Modelling:

- ▶ The input values of this problem (an array and its size)

# Modelling:

- ▶ The input values of this problem (an array and its size)
- ▶ A help from Python here, it is discussed at the end this modelling

```
$ cat inp_2_tug_of_war.txt
#const n=11.
weights(1,23).  weights(2,45).  weights(3,-34).  weights(4,12).
weights(5,0).   weights(6,98).  weights(7,-99). weights(8,4).
weights(9,189). weights(10,-1). weights(11,4).
```

- ▶ Reminding our second case – example:  
 $\{23, 45, -34, 12, 0, 98, -99, 4, 189, -1, 4\}$
  - ▶ **Already converted in ground terms**
-

## Creating the possible sets as solutions:

```
% creating sets 01 and 02
{ set_01(I,W) } :- weights(I,W).
{ set_02(I,W) } :- weights(I,W).

%%% a CONSTRAINT here: these sets are disjunctives
:- set_01(I,_), I = 1..n, set_02(J,_), J = 1..n, I == J.
%% any set has the same element referenced by an index
```

The  $n$  value was defined previously!

Here is the trick of this modelling – think about it!

---

# Counting and summing the elements of each set:

Extensive use of **aggregate functions**: #count and #sum

```
%% counting elements of sets 01 and 02 by index
```

```
n_1(N) :- N = #count{I: set_01(I,W) }.
```

```
n_2(N) :- N = #count{I: set_02(I,W) }.
```

```
size(N) :- N = #count{I: weights(I,W)}.
```

```
%% summing all elements of sets 01 and 02
```

```
sum_set_01(X) :- X = #sum{W,I : I= 1..n, set_01(I,W)}.
```

```
sum_set_02(X) :- X = #sum{W,I : I= 1..n, set_02(I,W)}.
```

## About the size of each set:

Reminding:

```
n_1(X): X is the cardinality of set_1
n_2(Y): Y is the cardinality of set_2
size(N): N is the cardinality of original set
...
so
..
% constraint 2 ... set 1 and 2 .. are almost the same size
:- n_1(X), n_2(Y) , X - Y > 1 .   % not allowed IF |X|  > |Y|
:- n_1(X), n_2(Y) , Y - X > 1.    % not allowed IF |Y|  > |X|
% constraint 1 ... sum of size sets are the same
:- n_1(X), n_2(Y) , size(N) , (X+Y) != N.
```

Securely, this part can be improved !

---

## Preparing for the optimization:

```
%% the abs works fine
diff(Z) :- sum_set_01(X), sum_set_02(Y), Z = |Y - X|.
%% #abs(Y-X).
```

```
%% if "a perfect" balance -- no differences of weight
answer('y_YES') :- diff(0).
answer('n_NO') :- diff(Z), Z != 0.
```

```
%% A minimizations on this difference
#minimize{ Z : diff(Z) }.
```

---

## The outputs:

```
#show n_1/1.  
#show n_2/1.  
%#show size/1.  
  
%% sets obtained  
#show set_01/2.  
#show set_02/2.  
%% sum of each side  
#show sum_set_01/1.  
#show sum_set_02/1.  
#show diff/1.  
#show answer/1.
```

---

## An output:

```
$ clingo tug_of_war.lp inp_2_tug_of_war.txt
qclingo version 5.3.0
Reading from tug_of_war.lp ...
Solving...
Answer: 1
set_02(4,12) set_02(5,0) set_02(3,-34) set_02(7,-99)
set_02(10,-1)
set_01(1,23) set_01(2,45) set_01(6,98) set_01(8,4)
set_01(9,189) set_01(11,4)
sum_set_02(-122) sum_set_01(363) diff(485)
answer('n_N0') n_2(5) n_1(6)
Optimization: 485
Answer: 2
.....
```

Many lines omitted!

---



## An output:

Finally in the answer 12:

...

Answer: 12

```
set_02(2,45) set_02(4,12) set_02(6,98) set_02(3,-34)
set_02(10,-1)
```

```
set_01(1,23) set_01(5,0) set_01(8,4) set_01(9,189)
set_01(11,4) set_01(7,-99)
```

```
sum_set_02(120) sum_set_01(121) diff(1)
```

```
answer('n_N0') n_2(5) n_1(6)
```

```
Optimization: 1
```

```
OPTIMUM FOUND
```

```
Models      : 12
```

```
  Optimum   : yes
```

```
Optimization : 1
```

```
Calls       : 1
```

```
Time        : 1.181s (Solving: 0.58s 1st Model: 0.03s Unsat: 0.
```

```
CPU Time    : 1.181s
```

# Using Python's code inside of ASP

- ▶ A big help from Adam Smith (Postasco mailing list)
- ▶ How to deal with input and output values in ASP?

# Using Python's code inside of ASP

- ▶ A big help from Adam Smith (Postasco mailing list)
- ▶ How to deal with input and output values in ASP?
- ▶ Some snippets of code in Python!
- ▶ The code discussed here:

`convert_input_value_in_terms_Python.lp`

## An example:

```
#script (python)
def my_size(*terms):
    i=0
    for term in enumerate(terms):
        i+=1
    return i
#end.

#script (python)
def multiset( *terms ):
    result = []
    for i , term in enumerate(terms):
        """ List of Pairs """
        result.append( (i+1, term) )
    return result
#end.
...
```

## Using Python's functions embedded in ASP:

```
...  
weights(I,W) :- (I,W) = @multiset( 23, 45, -34, 12, 0 ).  
size(S) :- S = @my_size(1,2,3,4,9999999999).  
...
```

---

```
$ clingo convert_input_value_in_terms_Python.lp  
...  
Answer: 1  
size(5) weights(1,23) weights(2,45) weights(3,-34)  
weights(4,12) weights(5,0)  
SATISFIABLE  
Models          : 1  
...
```

---

# Conclusions:

- ▶ ASP is strongly declarative (roots from the logic to attack the representation and combinatorial problems)
- ▶ A methodology generate and test to developing
- ▶ ASP's workflow, modeling, grounding, solving (and optimizing)
- ▶ Here, we solved *the tug of war problem*. Easy to understand, but is is a combinatorial problem.
- ▶ Allows you to embed a Python coding in order to minimize the difficulties (☺) of input and output data
- ▶ An encoding in ASP is excellent exercise to keep your mind very active!
- ▶ Finally, a huge gratitude for the **potassco-users list**, always reactive for my silly doubts, where I had been learning much.

# Contact and comments (are must welcome 😊):

- ▶ <https://claudiocesars.wordpress.com/>
- ▶ This presentation and the code discussed:  
[https://github.com/claudiosa/CCS/tree/master/asp\\_Answer\\_Set\\_Programming](https://github.com/claudiosa/CCS/tree/master/asp_Answer_Set_Programming)
- ▶ There is a directory to Youtube!
- ▶ The full code discussed here is found in:  
[https://github.com/claudiosa/CCS/tree/master/asp\\_Answer\\_Set\\_Programming/tug\\_of\\_war.lp](https://github.com/claudiosa/CCS/tree/master/asp_Answer_Set_Programming/tug_of_war.lp)
- ▶ ✉: [ccs1664@gmail.com](mailto:ccs1664@gmail.com)
- ▶ This material has a partial support from WhatsTV Inc.  
<https://en.whatstv.com.br/>, here our gratitude!
- ▶ *Thank you so much!*