Weighted Constraint Satisfaction Problem file format (wcsp)

Release 1.0.0

INRAE

Dec 05, 2023

CONTENTS

group `wcspformat`

It is a text format composed of a list of numerical and string terms separated by spaces. Instead of using names for making reference to variables, variable indexes are employed. The same for domain values. All indexes start at zero.

Cost functions can be defined in intention (see below) or in extension, by their list of tuples. A default cost value is defined per function in order to reduce the size of the list. Only tuples with a different cost value should be given (not mandatory). All the cost values must be positive. The arity of a cost function in extension may be equal to zero. In this case, there is no tuples and the default cost value is added to the cost of any solution. This can be used to represent a global lower bound constant of the problem.

The wcsp file format is composed of three parts: a problem header, the list of variable domain sizes, and the list of cost functions.

- Header definition for a given problem:

  ```
  <Problem name>
  <Number of variables (N)>
  <Maximum domain size>
  <Number of cost functions>
  <Initial global upper bound of the problem (UB)>
  ```

  The goal is to find an assignment of all the variables with minimum total cost, strictly lower than UB. Tuples with a cost greater than or equal to UB are forbidden (hard constraint).

- Definition of domain sizes

  ```
  <Domain size of variable with index 0>
  ...
  <Domain size of variable with index N - 1>
  ```

Note : domain values range from zero to *size-1*

Note : a negative domain size is interpreted as a variable with an interval domain in $[0, -size - 1]$

Warning : variables with interval domains are restricted to arithmetic and disjunctive cost functions in intention (see below)

- General definition of cost functions
  - Definition of a cost function in extension

    ```
    <Arity of the cost function>
    <Index of the first variable in the scope of the cost function>
    ...
    <Index of the last variable in the scope of the cost function>
    <Default cost value>
    <Number of tuples with a cost different than the default cost>
    ```

    followed by for every tuple with a cost different than the default cost:

    ```
    <Index of the value assigned to the first variable in the scope>
    ...
    ```

(continues on next page)

(continued from previous page)

```
<Index of the value assigned to the last variable in the scope>
<Cost of the tuple>
```

Note : Shared cost function: A cost function in extension can be shared by several cost functions with the same arity (and same domain sizes) but different scopes. In order to do that, the cost function to be shared must start by a negative scope size. Each shared cost function implicitly receives an occurrence number starting from 1 and incremented at each new shared definition. New cost functions in extension can reuse some previously defined shared cost functions in extension by using a negative number of tuples representing the occurrence number of the desired shared cost function. Note that default costs should be the same in the shared and new cost functions. Here is an example of 4 variables with domain size 4 and one AllDifferent hard constraint decomposed into 6 binary constraints.

- Shared CF used inside a small example in wcsp format:

```
AllDifferentDecomposedIntoBinaryConstraints 4 4 6 1
4 4 4 4
-2 0 1 0 4
0 0 1
1 1 1
2 2 1
3 3 1
2 0 2 0 -1
2 0 3 0 -1
2 1 2 0 -1
2 1 3 0 -1
2 2 3 0 -1
```

- Definition of a cost function in intension by replacing the default cost value by -1 and by giving its keyword name and its K parameters

```
<Arity of the cost function>
<Index of the first variable in the scope of the cost function>
...
<Index of the last variable in the scope of the cost function>
-1
<keyword>
<parameter1>
...
<parameterK>
```

Possible keywords of cost functions defined in intension followed by their specific parameters:

- >= *cst delta* to express soft binary constraint $x \geq y + cst$ with associated cost function $max((y + cst - x \leq delta)?(y + cst - x) : UB, 0)$
- > *cst delta* to express soft binary constraint $x > y + cst$ with associated cost function $max((y + cst + 1 - x \leq delta)?(y + cst + 1 - x) : UB, 0)$
- <= *cst delta* to express soft binary constraint $x \leq y + cst$ with associated cost function $max((x - cst - y \leq delta)?(x - cst - y) : UB, 0)$
- < *cst delta* to express soft binary constraint $x < y + cst$ with associated cost function $max((x - cst + 1 - y \leq delta)?(x - cst + 1 - y) : UB, 0)$

- = *cst delta* to express soft binary constraint $x = y + cst$ with associated cost function $(|y + cst - x| \leq delta)?|y + cst - x| : UB$
- disj *cstx csty penalty* to express soft binary disjunctive constraint $x \geq y + csty \vee y \geq x + cstx$ with associated cost function $(x \geq y + csty \vee y \geq x + cstx)?0 : penalty$
- sdisj *cstx csty xinfty yinfty costx costy* to express a special disjunctive constraint with three implicit hard constraints $x \leq xinfty$ and $y \leq yinfty$ and $x < xinfty \wedge y < yinfty \Rightarrow (x \geq y + csty \vee y \geq x + cstx)$ and an additional cost function $((x = xinfty)?costx : 0) + ((y = yinfty)?costy : 0)$
- Global cost functions using a dedicated propagator:
  - clique *1* (*nb_values* (*value*)*)* to express a hard clique cut to restrict the number of variables taking their value into a given set of values (per variable) to at most *1* occurrence for all the variables (warning! it assumes also a clique of binary constraints already exists to forbid any two variables using both the restricted values)
  - knapsack *capacity* (*weight*)* to express a reverse knapsack constraint (i.e., a linear constraint on 0/1 variables with >= operator) with capacity and weights are positive or negative integer coefficients (use negative numbers to express a linear constraint with <= operator)
  - knapsackc *capacity* (*weight*)* *nb_AMO* (*nb_variables* (*variable value*)*)* to express a reverse knapsack constraint (i.e., a linear constraint on 0/1 variables with >= operator) combined with a list of non-overlapping at-most-one constraints
  - knapsackp *capacity* (*nb_values* (*value weight*)*)* to express a reverse knapsack constraint with for each variable the list of values to select the item in the knapsack with their corresponding weight
  - knapsackv *capacity nb_triplets* (*variable value weight*)* to express a reverse knapsack constraint with a list of triplets variable, value, and its corresponding weight
  - wcsp *lb ub duplicatehard strongduality wcsp* to express a hard global constraint on the cost of an input weighted constraint satisfaction problem in wcsp format such that its valid solutions must have a cost value in [lb,ub[.
- Global cost functions using a flow-based propagator:
  - salldiff var|dec|decbi *cost* to express a soft alldifferent constraint with either variable-based (*var* keyword) or decomposition-based (*dec* and *decbi* keywords) cost semantic with a given *cost* per violation (*decbi* decomposes into a binary cost function complete network)
  - sgcc var|dec|wdec *cost nb_values* (*value lower_bound upper_bound* (*shortage_weight excess_weight*)?)* to express a soft global cardinality constraint with either variable-based (*var* keyword) or decomposition-based (*dec* keyword) cost semantic with a given *cost* per violation and for each value its lower and upper bound (if *wdec* then violation cost depends on each value shortage or excess weights)
  - ssame *cost list_size1 list_size2* (*variable_index*)* (*variable_index*)* to express a permutation constraint on two lists of variables of equal size (implicit variable-based cost semantic)
  - sregular var|edit *cost nb_states nb_initial_states* (*state*)* *nb_final_states* (*state*)* *nb_transitions* (*start_state symbol_value end_state*)* to express a soft regular constraint with either variable-based (*var* keyword) or edit distance-based (*edit* keyword) cost semantic with a given *cost* per violation followed by the definition of a deterministic finite automaton with number of states, list of initial and final states, and list of state transitions where symbols are domain values
- Global cost functions using a dynamic programming DAG-based propagator:
  - sregulardp var *cost nb_states nb_initial_states* (*state*)* *nb_final_states* (*state*)* *nb_transitions* (*start_state symbol_value end_state*)* to express a soft regular constraint with a variable-based (*var*

keyword) cost semantic with a given *cost* per violation followed by the definition of a deterministic finite automaton with number of states, list of initial and final states, and list of state transitions where symbols are domain values

- sgrammar|sgrammardp var|weight *cost nb_symbols nb_values start_symbol nb_rules* ((0 *terminal_symbol value*)|(1 *nonterminal_in nonterminal_out_left nonterminal_out_right*)|(2 *terminal_symbol value weight*)|(3 *nonterminal_in nonterminal_out_left nonterminal_out_right weight*))* to express a soft/weighted grammar in Chomsky normal form
- samong|samongdp var *cost lower_bound upper_bound nb_values* (*value*)* to express a soft among constraint to restrict the number of variables taking their value into a given set of values
- salldiffdp var *cost* to express a soft alldifferent constraint with variable-based (*var* keyword) cost semantic with a given *cost* per violation (decomposes into samongdp cost functions)
- sgccdp var *cost nb_values* (*value lower_bound upper_bound*)* to express a soft global cardinality constraint with variable-based (*var* keyword) cost semantic with a given *cost* per violation and for each value its lower and upper bound (decomposes into samongdp cost functions)
- max|smaxdp *defCost nbtuples* (*variable value cost*)* to express a weighted max cost function to find the maximum cost over a set of unary cost functions associated to a set of variables (by default, *defCost* if unspecified)
- MST|smstdp to express a spanning tree hard constraint where each variable is assigned to its parent variable index in order to build a spanning tree (the root being assigned to itself)

- Global cost functions using a cost function network-based propagator:
  - wregular *nb_states nb_initial_states* (*state* and cost)* *nb_final_states* (*state* and cost)* *nb_transitions* (*start_state symbol_value end_state cost*)* to express a weighted regular constraint with weights on initial states, final states, and transitions, followed by the definition of a deterministic finite automaton with number of states, list of initial and final states with their costs, and list of weighted state transitions where symbols are domain values
  - walldiff hard|lin|quad *cost* to express a soft alldifferent constraint as a set of wamong hard constraint (*hard* keyword) or decomposition-based (*lin* and *quad* keywords) cost semantic with a given *cost* per violation
  - wgcc hard|lin|quad *cost nb_values* (*value lower_bound upper_bound*)* to express a soft global cardinality constraint as either a hard constraint (*hard* keyword) or with decomposition-based (*lin* and *quad* keyword) cost semantic with a given *cost* per violation and for each value its lower and upper bound
  - wsame hard|lin|quad *cost* to express a permutation constraint on two lists of variables of equal size (implicitly concatenated in the scope) using implicit decomposition-based cost semantic
  - wsamegcc hard|lin|quad *cost nb_values* (*value lower_bound upper_bound*)* to express the combination of a soft global cardinality constraint and a permutation constraint
  - wamong hard|lin|quad *cost nb_values* (*value*)* *lower_bound upper_bound* to express a soft among constraint to restrict the number of variables taking their value into a given set of values
  - wvaramong hard *cost nb_values* (*value*)* to express a hard among constraint to restrict the number of variables taking their value into a given set of values to be equal to the last variable in the scope
  - woverlap hard|lin|quad *cost comparator righthandside* overlaps between two sequences of variables X, Y (i.e. set the fact that Xi and Yi take the same value (not equal to zero))
  - wsum hard|lin|quad *cost comparator righthandside* to express a soft sum constraint with unit coefficients to test if the sum of a set of variables matches with a given comparator and right-hand-side value

- wvarsum hard *cost comparator* to express a hard sum constraint to restrict the sum to be *comparator* to the value of the last variable in the scope
- wdiverse *distance* (*value*)* to express a hard diversity constraint using a dual encoding such that there is a given minimum Hamming distance to a given variable assignment
- whdiverse *distance* (*value*)* to express a hard diversity constraint using a hidden encoding such that there is a given minimum Hamming distance to a given variable assignment
- wtdiverse *distance* (*value*)* to express a hard diversity constraint using a ternary encoding such that there is a given minimum Hamming distance to a given variable assignment

  Let us note <> the comparator, K the right-hand-side value associated to the comparator, and Sum the result of the sum over the variables. For each comparator, the gap is defined according to the distance as follows:

  - if <> is == : gap = abs(K - Sum)
  - if <> is <= : gap = max(0,Sum - K)
  - if <> is < : gap = max(0,Sum - K - 1)
  - if <> is != : gap = 1 if Sum != K and gap = 0 otherwise
  - if <> is > : gap = max(0,K - Sum + 1);
  - if <> is >= : gap = max(0,K - Sum);

Warning : The decomposition of wsum and wvarsum may use an exponential size (sum of domain sizes).

Warning : *list_size1* and *list_size2* must be equal in *ssame*.

Warning : Cost functions defined in intention cannot be shared.

Note More about network-based global cost functions can be found on ./misc/doc/DecomposableGlobalCostFunctions.html

Examples:

- quadratic cost function $x0 * x1$ in extension with variable domains $\{0, 1\}$ (equivalent to a soft clause $\neg x0 \vee \neg x1$):

```
2 0 1 0 1 1 1 1
```

- simple arithmetic hard constraint $x1 < x2$:

```
2 1 2 -1 < 0 0
```

- hard temporal disjunction $x1 \geq x2 + 2 \vee x2 \geq x1 + 1$:

```
2 1 2 -1 disj 1 2 UB
```

- clique cut ({x0,x1,x2,x3}) on Boolean variables such that value 1 is used at most once:

```
4 0 1 2 3 -1 clique 1 1 1 1 1 1 1 1 1
```

- knapsack constraint ($2 * x0 + 3 * x1 + 4 * x2 + 5 * x3 >= 10$) on four Boolean 0/1 variables:

```
4 0 1 2 3 -1 knapsack 10 2 3 4 5
```

- knapsackc constraint ($2 * x0 + 3 * x1 + 4 * x2 + 5 * x3 >= 10$, $x1 + x2 <= 1$) on four Boolean 0/1 variables:

```
4 0 1 2 3 -1 knapsackc 10 2 3 4 5 1 2 1 1 2 1
```

- knapsackp constraint ($2 * (x0 = 0) + 3 * (x1 = 1) + 4 * (x2 = 2) + 5 * (x3 = 0 \vee x3 = 1) >= 10$) on four {0,1,2}-domain variables:

```
4 0 1 2 3 -1 knapsackp 10 1 0 2 1 1 3 1 2 4 2 0 5 1 5
```

- knapsackv constraint ($2 * (x0 = 0) + 3 * (x1 = 1) + 4 * (x2 = 2) + 5 * (x3 = 0 \vee x3 = 1) >= 10$) on four {0,1,2}-domain variables:

```
4 0 1 2 3 -1 knapsackv 10 5 0 0 2 1 1 3 2 2 4 3 0 5 3 1 5
```

- wcsp constraint ($3 <= 2 * x1 * x2 + 3 * x1 * x4 + 4 * x2 * x4 < 5$) on three Boolean 0/1 variables:

```
3 1 2 4 -1 wcsp 3 5 0 0 name 3 2 3 1000 2 2 2 2 0 1 0 1 1 1 2 2 0 2 0 1 1 1 3 2␣
↪1 2 0 1 1 1 4
```

- soft_alldifferent({x0,x1,x2,x3}):

```
4 0 1 2 3 -1 salldiff var 1
```

- soft_gcc({x1,x2,x3,x4}) with each value *v* from 1 to 4 only appearing at least v-1 and at most v+1 times:

```
4 1 2 3 4 -1 sgcc var 1 4 1 0 2 2 1 3 3 2 4 4 3 5
```

- soft_same({x0,x1,x2,x3},{x4,x5,x6,x7}):

```
8 0 1 2 3 4 5 6 7 -1 ssame 1 4 4 0 1 2 3 4 5 6 7
```

- soft_regular({x1,x2,x3,x4}) with DFA (3*)+(4*):

```
4 1 2 3 4 -1 sregular var 1 2 1 0 2 0 1 3 0 3 0 0 4 1 1 4 1
```

- soft_grammar({x0,x1,x2,x3}) with hard cost (1000) producing well-formed parenthesis expressions:

```
4 0 1 2 3 -1 sgrammardp var 1000 4 2 0 6 1 0 0 0 1 0 1 2 1 0 1 3 1 2 0 3 0 1 0␣
↪0 3 1
```

- soft_among({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{4}(x_i \in \{1,2\}) < 1$ or $\sum_{i=1}^{4}(x_i \in \{1,2\}) > 3$:

```
4 1 2 3 4 -1 samongdp var 1000 1 3 2 1 2
```

- soft max({x0,x1,x2,x3}) with cost equal to $\max_{i=0}^{3}((x_i! = i)?1000 : (4 - i))$:

```
4 0 1 2 3 -1 smaxdp 1000 4 0 0 4 1 1 3 2 2 2 3 3 1
```

- wregular({x0,x1,x2,x3}) with DFA (0(10)*2*):

```
4 0 1 2 3 -1 wregular 3 1 0 0 1 2 0 9 0 0 1 0 0 1 1 1 0 2 1 1 1 1 0 0 1 0 0 1 1␣
↪2 0 1 1 2 2 0 1 0 2 1 1 1 2 1
```

- wamong({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{4}(x_i \in \{1,2\}) < 1$ or $\sum_{i=1}^{4}(x_i \in \{1,2\}) > 3$:

```
4 1 2 3 4 -1 wamong hard 1000 2 1 2 1 3
```

- wvaramong({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{3}(x_i \in \{1,2\}) \neq x_4$:

```
4 1 2 3 4 -1 wvaramong hard 1000 2 1 2
```

- woverlap({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{2}(x_i = x_{i+2}) \geq 1$:

```
4 1 2 3 4 -1 woverlap hard 1000 < 1
```

- wsum({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{4}(x_i) \neq 4$:

```
4 1 2 3 4 -1 wsum hard 1000 == 4
```

- wvarsum({x1,x2,x3,x4}) with hard cost (1000) if $\sum_{i=1}^{3}(x_i) \neq x_4$:

```
4 1 2 3 4 -1 wvarsum hard 1000 ==
```

- wdiverse({x0,x1,x2,x3}) hard constraint on four variables with minimum Hamming distance of 2 to the value assignment (1,1,0,0):

```
4 0 1 2 3 -1 wdiverse 2 1 1 0 0
```

Latin Square 4 x 4 crisp CSP example in wcsp format:

```
latin4 16 4 8 1
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 0 1 2 3 -1 salldiff var 1
4 4 5 6 7 -1 salldiff var 1
4 8 9 10 11 -1 salldiff var 1
4 12 13 14 15 -1 salldiff var 1
4 0 4 8 12 -1 salldiff var 1
4 1 5 9 13 -1 salldiff var 1
4 2 6 10 14 -1 salldiff var 1
4 3 7 11 15 -1 salldiff var 1
```

4-queens binary weighted CSP example with random unary costs in wcsp format:

```
4-WQUEENS 4 4 10 5
4 4 4 4
2 0 1 0 10
0 0 5
0 1 5
1 0 5
1 1 5
1 2 5
2 1 5
2 2 5
2 3 5
3 2 5
3 3 5
2 0 2 0 8
0 0 5
0 2 5
1 1 5
1 3 5
2 0 5
2 2 5
```

(continues on next page)

(continued from previous page)

```
3 1 5
3 3 5
2 0 3 0 6
0 0 5
0 3 5
1 1 5
2 2 5
3 0 5
3 3 5
2 1 2 0 10
0 0 5
0 1 5
1 0 5
1 1 5
1 2 5
2 1 5
2 2 5
2 3 5
3 2 5
3 3 5
2 1 3 0 8
0 0 5
0 2 5
1 1 5
1 3 5
2 0 5
2 2 5
3 1 5
3 3 5
2 2 3 0 10
0 0 5
0 1 5
1 0 5
1 1 5
1 2 5
2 1 5
2 2 5
2 3 5
3 2 5
3 3 5
1 0 0 2
1 1
3 1
1 1 0 2
1 1
2 1
1 2 0 2
1 1
2 1
1 3 0 2
0 1
2 1
```