# C++ Library of toulbar2

Release 1.0.0

**INRAE** 

## **CONTENTS**

1	Introduction	1
2	Main types and constants	2
3	WeightedCSP class	5
4	WeightedCSPSolver class	23
5	MultiCFN class	28
6	ToulBar2 class	33
7	Miscellaneous functions	48
In	dex	49

### **CHAPTER**

## ONE

## **INTRODUCTION**

toulbar2 is an open-source C++ solver for cost function networks.

See: Class Diagram.

## **MAIN TYPES AND CONSTANTS**

#### The main types are:

- ::Value : domain value
- ::Cost : cost value (exact type depends on compilation flag)
- ::Long : large integer (long long int)
- ::TProb : probability value (exact type depends on compilation flag)
- ::TLogProb : log probability value (exact type depends on compilation flag)
- ::Double : large float (long double)
- ::tValue : a short Value type for tuples
- ::Tuple : vector of tValues to encode tuples

#### **1** Note

Compilation flag for Cost is: INT\_COST (int), LONGLONG\_COST (long long), or PARETOPAIR\_COST (see ::ParetoPair)

#### 1 Note

Compilation flag for TProb is: DOUBLE\_PROB or LONGDOUBLE\_PROB

#### **1** Note

Compilation flag for T(Log)Prob is: DOUBLE\_PROB or LONGDOUBLE\_PROB

#### Warning

PARETOPAIR\_COST is fragile.

#### **Variables**

```
const string IMPLICIT_VAR_TAG = "#"
     Special character value at the beginning of a variable's name to identify implicit variables (i.e., variables which
     are not decision variables)
const string HIDDEN_VAR_TAG = "^"
     Special character value at the beginning of a variable's name to identify hidden variables like diverse extra
     variables corresponding to the current sequence of diverse solutions found so far.
const string HIDDEN_VAR_TAG_HVE = "^c"
const string HIDDEN_VAR_TAG_HVE_PRE = "^!"
const Value MAX_VAL = (std::numeric_limits < Value >::max() / 2)
     Maximum domain value.
const Value WRONG_VAL = std::numeric_limits<Value>::max()
     Forbidden domain value.
const Value MIN_VAL = -(std::numeric limits<Value>::max() / 2)
     Minimum domain value.
const Value MAX_DOMAIN_SIZE = 10000
     Maximum domain size Deprecated Should use WCSP::getMaxDomainSize instead.
const int NARYPROJECTIONSIZE = 3
const unsigned int NARYPROJECTION2MAXDOMSIZE = 100
const unsigned int NARYPROJECTION3MAXDOMSIZE = 30
const Long NARYPROJECTIONPRODDOMSIZE = 10000
const Long NARYDECONNECTSIZE = 4
const int MAX_BRANCH_SIZE = 1000000
const\ ptrdiff\_t\ \textbf{CHOICE\_POINT\_LIMIT} = SIZE\_MAX
const ptrdiff_t OPEN_NODE_LIMIT = SIZE_MAX
const int STORE_SIZE = 16
```

```
const int \texttt{MAX\_ELIM\_BIN} = 10000000000
```

const int  $MAX\_ARITY = 1000$ 

const Long  $MAX_NB_TUPLES = 100000LL$ 

Maximum number of tuples in n-ary cost functions.

const int LARGE\_NB\_VARS = 10000

const int **DECIMAL\_POINT** = 3

const bool NARY2CLAUSE = true

const int  $MAX\_EAC\_ITER = 10000$ 

**CHAPTER** 

#### THREE

#### WEIGHTEDCSP CLASS

#### class WeightedCSP

Abstract class WeightedCSP representing a weighted constraint satisfaction problem

- problem lower and upper bounds
- list of variables with their finite domains (either represented by an enumerated list of values, or by a single interval)
- list of cost functions (created before and during search by variable elimination of variables with small degree)
- local consistency propagation (variable-based propagation) including cluster tree decomposition caching (separator-based cache)

### **1** Note

Variables are referenced by their lexicographic index number (as returned by eg Weighted-CSP::makeEnumeratedVariable)

#### **1** Note

Cost functions are referenced by their lexicographic index number (as returned by eg Weighted-CSP::postBinaryConstraint)

#### **Public Functions**

```
inline virtual ~WeightedCSP()
virtual int getIndex() const = 0
    instantiation occurrence number of current WCSP object
virtual string getName() const = 0
    get WCSP problem name (defaults to filename with no extension)
virtual void setName(const string &problem) = 0
    set WCSP problem name
virtual void *getSolver() const = 0
    special hook to access solver information
```

virtual Cost **getLb()** const = 0

gets internal dual lower bound

virtual Cost getUb() const = 0

gets internal primal upper bound

virtual Double **getDPrimalBound()** const = 0

gets problem primal bound as a Double representing a decimal cost (upper resp. lower bound for minimization resp. maximization)

virtual Double **getDDualBound()** const = 0

gets problem dual bound as a Double representing a decimal cost (lower resp. upper bound for minimization resp. maximization)

virtual Double **getDLb()** const = 0

gets problem lower bound as a Double representing a decimal cost

virtual Double **getDUb()** const = 0

gets problem upper bound as a Double representing a decimal cost

virtual void **updateDUb**(Double newDUb) = 0

sets problem upper bound as a Double representing a decimal cost

virtual void **updateUb**(Cost newUb) = 0

sets initial problem upper bound and each time a new solution is found

virtual void **enforceUb()** = 0

enforces problem upper bound when exploring an alternative search node

virtual void **increaseLb**(Cost addLb) = 0

increases problem lower bound thanks to eg soft local consistencies

**Parameters** 

addLb – increment value to be added to the problem lower bound

virtual void **decreaseLb**(Cost shift) = 0

shift problem optimum toward negative costs

#### **Parameters**

**shift** – positive shifting value to be subtracted to the problem optimum when printing the solutions

virtual Cost **getNegativeLb()** const = 0

gets constant term used to subtract to the problem optimum when printing the solutions

virtual Cost **finiteUb**() const = 0

computes the worst-case assignment finite cost (sum of maximum finite cost over all cost functions plus one)



#### Warning

current problem should be completely loaded and propagated before calling this function

#### Returns

the worst-case assignment finite cost

virtual void **setInfiniteCost()** = 0

updates infinite costs in all cost functions accordingly to the problem global lower and upper bounds

#### Warning

to be used in preprocessing only

virtual bool **isfinite()** const = 0

returns true if any complete assignment using current domains is a valid tuple with finite cost (i.e., cost strictly less than the problem upper bound minus the lower bound)

virtual bool **enumerated**(int varIndex) const = 0

true if the variable has an enumerated domain

virtual string **getName**(int varIndex) const = 0



#### 1 Note

by default, variables names are integers, starting at zero

virtual unsigned int **getVarIndex**(const string &s) const = 0

return variable index from its name, or numberOfVariables() if not found

virtual Value **getInf**(int varIndex) const = 0

minimum current domain value

virtual Value **getSup**(int varIndex) const = 0

maximum current domain value

virtual Value **getValue**(int varIndex) const = 0

current assigned value



#### Warning

undefined if not assigned yet

virtual unsigned int **getDomainSize**(int varIndex) const = 0

current domain size

virtual vector<Value> getEnumDomain(int varIndex) = 0

gets current domain values in an array

virtual bool **getEnumDomain**(int varIndex, Value \*array) = 0

#### **Deprecated**

virtual vector<pair<Value, Cost>> getEnumDomainAndCost(int varIndex) = 0

gets current domain values and unary costs in an array

virtual bool **getEnumDomainAndCost**(int varIndex, ValueCost \*array) = 0

#### **Deprecated**

```
virtual unsigned int getDomainInitSize(int varIndex) const = 0
     gets initial domain size (warning! assumes EnumeratedVariable)
virtual Value toValue(int varIndex, unsigned int idx) = 0
     gets value from index (warning! assumes EnumeratedVariable)
virtual unsigned int toIndex(int varIndex, Value value) = 0
     gets index from value (warning! assumes EnumeratedVariable)
virtual unsigned int toIndex(int varIndex, const string &valueName) = 0
     gets index from value name (warning! assumes EnumeratedVariable with value names)
virtual int getDACOrder(int varIndex) const = 0
     index of the variable in the DAC variable ordering
virtual bool assigned(int varIndex) const = 0
virtual bool unassigned(int varIndex) const = 0
virtual bool canbe(int varIndex, Value v) const = 0
virtual bool cannotbe(int varIndex, Value v) const = 0
virtual Value nextValue(int varIndex, Value v) const = 0
     first value after v in the current domain or v if there is no value
virtual void increase(int varIndex, Value newInf) = 0
     changes domain lower bound
virtual void decrease(int varIndex, Value newSup) = 0
     changes domain upper bound
virtual void assign(int varIndex, Value newValue) = 0
     assigns a variable and immediately propagates this assignment
virtual void remove(int varIndex, Value remValue) = 0
     removes a domain value (valid if done for an enumerated variable or on its domain bounds)
virtual void assignLS(vector<int> &varIndexes, vector<Value> &newValues, bool force = false) = 0
     assigns a set of variables at once and propagates (used by Local Search methods such as Large Neighbor-
     hood Search)
         Parameters
              • varIndexes – vector of variable indexes as returned by makeXXXVariable
              • newValues – vector of values to be assigned to the corresponding variables
              • force – boolean if true then apply assignLS even if the variable is already assigned Note
                this function is equivalent but faster than a sequence of assign.
virtual void assignLS(int *varIndexes, Value *newValues, unsigned int size, bool dopropagate, bool force =
                       false = 0
```

## Incremental Search) Parameters

virtual void **deconnect**(vector<int> &varIndexes) = 0

**varIndexes** – vector of variable indexes as returned by makeXXXVariable

deconnects a set of variables from the rest of the problem and assigns them to their support value (used by

```
virtual Cost getUnaryCost (int varIndex, Value v) const = 0
     unary cost associated to a domain value
virtual Cost getMaxUnaryCost(int varIndex) const = 0
     maximum unary cost in the domain
virtual Value getMaxUnaryCostValue(int varIndex) const = 0
     a value having the maximum unary cost in the domain
virtual Value getSupport (int varIndex) const = 0
     NC/EAC unary support value.
virtual Value getBestValue(int varIndex) const = 0
     hint for some value ordering heuristics (only used by RDS)
virtual void setBestValue(int varIndex, Value v) = 0
     hint for some value ordering heuristics (only used by RDS)
virtual bool getIsPartOfOptimalSolution() = 0
     special flag used for debugging purposes only
virtual void setIsPartOfOptimalSolution(bool v) = 0
     special flag used for debugging purposes only
virtual int getDegree(int varIndex) const = 0
     approximate degree of a variable (ie number of active cost functions, see Variable elimination)
virtual int getTrueDegree(int varIndex) const = 0
     degree of a variable
virtual Long getWeightedDegree(int varIndex) const = 0
     weighted degree heuristic
virtual void resetWeightedDegree() = 0
     initialize weighted degree heuristic
virtual void resetTightness() = 0
     initialize constraint tightness used by some heuristics (including weighted degree)
virtual void resetTightnessAndWeightedDegree() = 0
     initialize tightness and weighted degree heuristics
virtual void preprocessing() = 0
     applies various preprocessing techniques to simplify the current problem
virtual void sortConstraints() = 0
     sorts the list of cost functions associated to each variable based on smallest problem variable indexes
```



must be called after creating all the cost functions and before solving the problem

## **▲** Warning

side-effect: updates DAC order according to an existing variable elimination order

```
virtual void whenContradiction() = 0
     after a contradiction, resets propagation queues
virtual void deactivatePropagate() = 0
     forbids propagate calls
virtual bool isactivatePropagate() = 0
     are propagate calls authorized?
virtual void reactivatePropagate() = 0
     re-authorizes propagate calls
virtual void propagate (bool fromscratch = false) = 0
     (if authorized) propagates until a fix point is reached (or throws a contradiction). If fromscratch is true then
     propagates every cost function at least once.
virtual bool verify() = 0
     checks the propagation fix point is reached
virtual unsigned int numberOfVariables() const = 0
     number of created variables
virtual unsigned int numberOfUnassignedVariables() const = 0
     current number of unassigned variables (warning! linear time complexity)
virtual unsigned int numberOfConstraints() const = 0
     initial number of cost functions (before variable elimination)
virtual unsigned int numberOfConnectedConstraints() const = 0
     current number of cost functions (warning! linear time complexity)
virtual unsigned int numberOfConnectedBinaryConstraints() const = 0
     current number of binary cost functions
virtual unsigned int numberOfConnectedKnapsackConstraints() const = 0
     current number of knapsack cost functions
virtual unsigned int medianDomainSize() const = 0
     median current domain size of variables
virtual unsigned int medianDegree() const = 0
     median current degree of variables
virtual unsigned int medianArity() const = 0
     median arity of current cost functions
virtual unsigned int getMaxDomainSize() const = 0
     maximum initial domain size found in all variables
virtual unsigned int getMaxCurrentDomainSize() const = 0
     maximum current domain size found in all variables
virtual unsigned int getDomainSizeSum() const = 0
     total sum of current domain sizes
virtual void cartProd(BigInteger &cartesianProduct) = 0
     Cartesian product of current domain sizes.
```

#### **Parameters**

cartesianProduct – result obtained by the GNU Multiple Precision Arithmetic Library **GMP** 

virtual Long getNbDEE() const = 0

number of value removals due to dead-end elimination

virtual int makeEnumeratedVariable(string n, Value iinf, Value isup) = 0

create an enumerated variable with its domain bounds

virtual int makeEnumeratedVariable(string n, vector<Value> &dom) = 0

create an enumerated variable with its domain values

virtual void **addValueName**(int xIndex, const string &valuename) = 0

add next value name



#### Warning

should be called on Enumerated Variable object as many times as its number of initial domain values

virtual const string &getValueName(int xIndex, Value value) = 0

return the name associated to a value as defined by addValueName or an empty string if no name found

virtual int makeIntervalVariable(string n, Value iinf, Value isup) = 0

create an interval variable with its domain bounds

virtual void **postNullaryConstraint**(Double cost) = 0

add a zero-arity cost function with floating-point cost

virtual void **postUnaryConstraint**(int xIndex, vector<Double> &costs, bool incremental = false) = 0

add a unary cost function with floating-point costs (if incremental is true then it disappears upon backtrack)

virtual int postBinaryConstraint(int xIndex, int yIndex, vector<Double> &costs, bool incremental = false) = 0

add a binary cost function with floating-point costs (if incremental is true then it disappears upon backtrack)

virtual int postTernaryConstraint(int xIndex, int yIndex, int zIndex, vector<Double> &costs, bool incremental = false) = 0

add a ternary cost function with floating-point costs (if incremental is true then it disappears upon backtrack)

virtual void **postNullaryConstraint**(Cost cost) = 0

virtual void **postUnary**(int xIndex, vector<Cost> &costs) = 0

Deprecated Please use the postUnaryConstraint method instead

virtual void **postUnaryConstraint**(int xIndex, vector<Cost> &costs) = 0

virtual void **postIncrementalUnaryConstraint**(int xIndex, vector<Cost> &costs) = 0

virtual int **postBinaryConstraint** (int xIndex, int yIndex, vector<Cost> &costs) = 0

virtual int postIncrementalBinaryConstraint(int xIndex, int yIndex, vector<Cost> &costs) = 0

virtual int **postTernaryConstraint** (int xIndex, int yIndex, int zIndex, vector<Cost> &costs) = 0

virtual int **postIncrementalTernaryConstraint** (int xIndex, int yIndex, int zIndex, vector<Cost> &costs) = 0

virtual int **postNaryConstraintBegin**(vector<int> scope, Cost defval, Long nbtuples = 0, bool forcenary = !NARY2CLAUSE) = 0



#### 🛕 Warning

must call WeightedCSP::postNaryConstraintEnd after giving cost tuples

virtual int **postNaryConstraintBegin**(int \*scope, int arity, Cost defval, Long nbtuples = 0, bool forcenary = !NARY2CLAUSE) = 0

#### **Deprecated**

virtual void **postNaryConstraintTuple** (int ctrindex, vector<Value> &tuple, Cost cost) = 0

virtual void **postNaryConstraintTuple**(int ctrindex, Value \*tuple, int arity, Cost cost) = 0

#### **Deprecated**

virtual void **postNaryConstraintEnd**(int ctrindex) = 0



#### Warning

must call WeightedCSP::sortConstraints after all cost functions have been posted (see Weighted-CSP::sortConstraints)

virtual int **postUnary**(int xIndex, Value \*d, int dsize, Cost penalty) = 0

**Deprecated** Please use the postUnaryConstraint method instead

virtual int **postUnaryConstraint**(int xIndex, Value \*d, int dsize, Cost penalty) = 0

virtual int **postSupxyc** (int xIndex, int yIndex, Value cst, Value deltamax =  $MAX_VAL - MIN_VAL$ ) = 0

virtual int **postDisjunction**(int xIndex, int yIndex, Value cstx, Value csty, Cost penalty) = 0

virtual int **postSpecialDisjunction**(int xIndex, int yIndex, Value cstx, Value csty, Value xinfty, Value yinfty, Cost costx, Cost costy) = 0

virtual int postAllDifferentConstraint(vector<int> scope, const string & arguments) = 0

virtual int postAllDifferentConstraint(int \*scopeIndex, int arity, istream &file) = 0

#### **Deprecated**

virtual int **postCliqueConstraint**(vector<int> scope, const string & arguments) = 0

virtual int **postCliqueConstraint** (int \*scopeIndex, int arity, istream &file) = 0

#### **Deprecated**

virtual int **postKnapsackConstraint** (vector<int> scope, const string & arguments, bool isclique = false, int kp = 0, bool conflict = false, Tuple wcnf = {}) = 0

create a knapsack constraint (also known as generalized linear constraint) e.g.  $c1 * (v_0 == a_0) + c2 * (v_1 == a_0) + [...] >= c$ 

The constraint parameters are given as a string:

- the first argument is the capacity
- kp = 0 (knapsack keyword in wcsp format): list of weights (positive or negative integers) associated to the scope variables (must be 0/1 variables)
- kp = 1 (knapsackp keyword in wcsp format): for each variable in the scope, the number of values with a nonzero weight, followed by the list of pairs of value and corresponding nonzero weight
- kp = 2 (knapsacky keyword in wcsp format): number of triplets, followed by the list of triplets of variable index in the scope, value, and its corresponding weight



#### 1 Note

The knapsack constraint assumes a greater than or equal operator, use negative weights and capacity if the less than or equal operator is needed.

#### **Parameters**

- **scope** variable indices of the constraint scope.
- **arguments** constraint parameters (capacity, coefficients) as a string.
- isclique if true, then arguments contains a fixed capacity (must be 1) and for each variable in the scope, the number of values, followed by the list of values having a unit weight (clique keywork in wcsp format, warning: it assumes a less than or equal operator)
- **kp** see above
- conflict if true, then kp must be 0 and arguments is extended with a list of nonoverlapping at-most-one constraints given by the number of AMO constraints, followed by, for each AMO constraint, the number of pairs, followed by the list of pairs of variable index in the scope and value (knapsackc keyword in wcsp format)
- wcnf if not empty, then it contains a vector of -1/1 coefficients describing a hard clause (for each Boolean variable in the scope, it must be true if the associated coefficient is 1 or its opposite must be true if -1)

virtual int postKnapsackConstraint(int \*scopeIndex, int arity, istream &file, bool isclique = false, int kp = 0, bool conflict = false, Tuple wcnf =  $\{\}$ ) = 0

#### **Deprecated**

virtual int postWeightedCSPConstraint(vector<int> scope, WeightedCSP \*problem, WeightedCSP \*negproblem, Cost lb = MIN COST, Cost ub = MAX COST, bool duplicateHard = false, bool strongDuality = false) = 0

create a hard constraint such that the input cost function network (problem) must have its optimum cost in [lb,ub[ interval.



#### Warning

The input scope must contain all variables in problem in the same order.



#### Warning

if duplicateHard is true it assumes any forbidden tuple in the original input problem is also forbidden by another constraint in the main model (you must duplicate any hard constraints in your input model into the main model).

#### Warning

if strongDuality is true then it assumes the propagation is complete when all channeling variables in the scope are assigned and the semantic of the constraint enforces that the optimum on the remaining variables is between lb and ub.

virtual int postGlobalConstraint(int \*scopeIndex, int arity, const string &gcname, istream &file, int \*constrcounter = NULL, bool mult = true) = 0

Deprecated Please use the postWxxx methods instead

virtual void **postGlobalFunction**(vector<int> scope, const string &gcname, const string &arguments) = 0 generic function to post any global cost function

virtual int **postWAmong** (vector<int> scope, const string &semantics, const string &propagator, Cost baseCost, const vector<Value> &values, int lb, int ub) = 0

post a soft among cost function

post a soft weighted among cost function

#### **Parameters**

- scopeIndex an array of variable indexes as returned by Weighted-CSP::makeEnumeratedVariable
- **arity** the size of the array
- semantics the semantics of the global cost function: "var" or "hard" or "lin" or "quad" (network-based propagator only)—
- **propagator** the propagation method (only "DAG" or "network")
- **baseCost** the scaling factor of the violation
- values a vector of values to be restricted
- 1b a fixed lower bound for the number variables to be assigned to the values in *values*
- **ub** a fixed upper bound for the number variables to be assigned to the values in *values*

virtual int postWAmong (int \*scopeIndex, int arity, const string &semantics, const string &propagator, Cost baseCost, const vector<Value> &values, int lb, int ub) = 0

#### **Deprecated**

virtual void postWAmong (int \*scopeIndex, int arity, string semantics, Cost baseCost, Value \*values, int nbValues, int lb, int ub) = 0

#### **Deprecated**

virtual void **postWVarAmong** (vector<int> scope, const string &semantics, Cost baseCost, vector<Value> &values) = 0

post a weighted among cost function with the number of values encoded as the last variable with index *varIndex* (*network-based* propagator only)

virtual void **postWVarAmong** (int \*scopeIndex, int arity, const string &semantics, Cost baseCost, Value \*values, int nbValues) = 0

#### **Deprecated**

virtual int **postWRegular** (vector<int> scope, const string &semantics, const string &propagator, Cost baseCost, int nbStates, const vector<WeightedObjInt> &initial\_States, const vector<WeightedObjInt> &accepting\_States, const vector<DFATransition> &Wtransitions) = 0

post a soft or weighted regular cost function

post a soft weighted regular cost function



#### Warning

Weights are ignored in the current implementation of DAG and flow-based propagators

#### **Parameters**

- scopeIndex an array of variable indexes as returned Weighted-CSP::makeEnumeratedVariable
- arity the size of the array
- semantics the semantics of the soft global cost function: "var" or "edit" (flow-based propagator) or — "var" (DAG-based propagator)— (unused parameter for network-based propagator)
- **propagator** the propagation method ("flow", "DAG", "network")
- baseCost the scaling factor of the violation ("flow", "DAG")
- **nbStates** the number of the states in the corresponding DFA. The states are indexed as 0, 1, ..., nbStates-1
- initial\_States a vector of WeightedObjInt specifying the starting states with weight
- accepting\_States a vector of WeightedObjInt specifying the final states
- Wtransitions a vector of (weighted) transitions

virtual int **postWRegular** (int \*scopeIndex, int arity, const string &semantics, const string &propagator, Cost baseCost, int nbStates, const vector<WeightedObjInt> &initial\_States, const vector<WeightedObjInt> &accepting\_States, const vector<DFATransition> &Wtransitions) = 0

#### **Deprecated**

virtual void **postWRegular** (int \*scopeIndex, int arity, int nbStates, vector<pair<int, Cost>> initial\_States, vector<pair<int, Cost>> accepting\_States, int \*\*Wtransitions, vector<Cost> transitionsCosts) = 0

**Deprecated** post a weighted regular cost function decomposed as a cost function network

virtual int postWAllDiff(vector<int> scope, const string &semantics, const string &propagator, Cost baseCost) = 0

post a soft all different cost function

- scopeIndex an array of variable indexes as returned by Weighted-CSP::makeEnumeratedVariable
- arity the size of the array
- **semantics** the semantics of the global cost function: for flow-based propagator: "var" or "dec" or "decbi" (decomposed into a binary cost function complete network), for DAG-based propagator: "var", for network-based propagator: "hard" or "lin" or "quad" (decomposed based on wamong)
- **propagator** the propagation method ("flow", "DAG", "network")
- **baseCost** the scaling factor of the violation post a soft all different cost function

virtual int **postWAllDiff**(int \*scopeIndex, int arity, const string &semantics, const string &propagator, Cost baseCost) = 0

#### **Deprecated**

virtual void **postWallDiff**(int \*scopeIndex, int arity, string semantics, Cost baseCost) = 0

**Deprecated** post a soft all different cost function decomposed as a cost function network

virtual int **postWGcc**(int \*scopeIndex, int arity, const string &semantics, const string &propagator, Cost baseCost, const vector<BoundedObjValue> &values) = 0

post a soft global cardinality cost function

#### **Parameters**

- scopeIndex an array of variable indexes as returned by Weighted-CSP::makeEnumeratedVariable
- arity the size of the array
- **semantics** the semantics of the global cost function: "var" (DAG-based propagator only) or "var" or "dec" or "wdec" (flow-based propagator only) or "hard" or "lin" or "quad" (network-based propagator only)—
- **propagator** the propagation method ("flow", "DAG", "network")
- baseCost the scaling factor of the violation
- **values** a vector of BoundedObjValue, specifying the lower and upper bounds of each value, restricting the number of variables can be assigned to them

virtual void **postWGcc**(int \*scopeIndex, int arity, string semantics, Cost baseCost, Value \*values, int nbValues, int \*lb, int \*ub) = 0

**Deprecated** post a soft global cardinality cost function decomposed as a cost function network

virtual int **postWSame** (int \*scopeIndexG1, int arityG1, int \*scopeIndexG2, int arityG2, const string &semantics, const string &propagator, Cost baseCost) = 0

post a soft same cost function (a group of variables being a permutation of another group with the same size)

- scopeIndexG1 an array of the first group of variable indexes as returned by Weighted-CSP::makeEnumeratedVariable
- arityG1 the size of *scopeIndexG1*
- scopeIndexG2 an array of the second group of variable indexes as returned by WeightedCSP::makeEnumeratedVariable
- **arityG2** the size of *scopeIndexG2*

- **semantics** the semantics of the global cost function: "var" or "hard" or "lin" or "quad" (network-based propagator only)—
- **propagator** the propagation method ("flow" or "network")
- **baseCost** the scaling factor of the violation.

virtual void **postWSame**(int \*scopeIndex, int arity, string semantics, Cost baseCost) = 0

**Deprecated** post a soft same cost function

virtual void **postWSameGcc** (int \*scopeIndex, int arity, string semantics, Cost baseCost, Value \*values, int nbValues, int \*lb, int \*ub) = 0

post a combination of a same and gcc cost function decomposed as a cost function network

virtual int **postWGrammarCNF** (int \*scopeIndex, int arity, const string &semantics, const string &propagator, Cost baseCost, int nbSymbols, int startSymbol, const vector<CFGProductionRule> WRuleToTerminal) = 0

post a soft/weighted grammar cost function with the dynamic programming propagator and grammar in Chomsky normal form

#### **Parameters**

- scopeIndex an array of the first group of variable indexes as returned by Weighted-CSP::makeEnumeratedVariable
- arity the size of scopeIndex
- semantics the semantics of the global cost function: "var" or "weight"
- **propagator** the propagation method ("DAG" only)
- baseCost the scaling factor of the violation
- **nbSymbols** the number of symbols in the corresponding grammar. Symbols are indexed as 0, 1, ..., nbSymbols-1
- **startSymbol** the index of the starting symbol
- **WRuleToTerminal** a vector of :: *CFGProductionRule*. Note that:
  - if *order* in *CFGProductionRule* is set to 0, it is classified as A -> v, where A is the index of the terminal symbol and v is the value.
  - if *order* in *CFGProductionRule* is set to 1, it is classified as A -> BC, where A,B,C the index of the nonterminal symbols.
  - if order in CFGProductionRule is set to 2, it is classified as weighted A -> v, where A is the index of the terminal symbol and v is the value.
  - if order in CFGProductionRule is set to 3, it is classified as weighted A -> BC, where A,B,C the index of the nonterminal symbols.
  - if *order* in *CFGProductionRule* is set to values greater than 3, it is ignored.

virtual int **postMST** (int \*scopeIndex, int arity, const string &semantics, const string &propagator, Cost baseCost) = 0

post a Spanning Tree hard constraint

- **scopeIndex** an array of variable indexes as returned by *Weighted-CSP::makeEnumeratedVariable*
- arity the size of scopeIndex

- **semantics** the semantics of the global cost function: "hard"
- **propagator** the propagation method ("DAG" only)
- baseCost unused in the current implementation (MAX\_COST)

virtual int **postMaxWeight** (int \*scopeIndex, int arity, const string &semantics, const string &propagator, Cost baseCost, const vector<WeightedVarValPair> weightFunction) = 0

post a weighted max cost function (maximum cost of a set of unary cost functions associated to a set of variables)

#### **Parameters**

- scopeIndex an array of variable indexes as returned by Weighted-CSP::makeEnumeratedVariable
- arity the size of *scopeIndex*
- **semantics** the semantics of the global cost function: "val"
- **propagator** the propagation method ("DAG" only)
- **baseCost** if a variable-value pair does not exist in *weightFunction*, its weight will be mapped to baseCost.
- **weightFunction** a vector of WeightedVarValPair containing a mapping from variable-value pairs to their weights.

virtual void **postWSum**(int \*scopeIndex, int arity, string semantics, Cost baseCost, string comparator, int rightRes) = 0

post a soft linear constraint with unit coefficients

#### **Parameters**

- scopeIndex an array of variable indexes as returned by Weighted-CSP::makeEnumeratedVariable
- arity the size of *scopeIndex*
- **semantics** the semantics of the global cost function: "hard" or "lin" or "quad" (network-based propagator only)
- **propagator** the propagation method ("network" only)
- baseCost the scaling factor of the violation
- **comparator** the comparison operator of the linear constraint ("==", "!=", "<", "<=", ">,", ">=")
- **rightRes** right-hand side value of the linear constraint

virtual void **postWvarSum**(int \*scopeIndex, int arity, string semantics, Cost baseCost, string comparator) = 0 post a soft linear constraint with unit coefficients and last variable as right-hand side

virtual void **postWOverlap**(int \*scopeIndex, int arity, string semantics, Cost baseCost, string comparator, int rightRes) = 0

post a soft overlap cost function (a group of variables being point-wise equivalent — and not equal to zero — to another group with the same size)

- scopeIndex an array of variable indexes as returned by Weighted-CSP::makeEnumeratedVariable
- **arity** the size of *scopeIndex* (should be an even value)

- semantics the semantics of the global cost function: "hard" or "lin" or "quad" (networkbased propagator only)
- **propagator** the propagation method ("network" only)
- baseCost the scaling factor of the violation.
- comparator the point-wise comparison operator applied to the number of equivalent variables ("==", "!=", "<", "<=", ">,", ">=")
- rightRes right-hand side value of the comparison

virtual void **postWDivConstraint**(vector<int> scope, unsigned int distance, vector<Value> &values, int method = 0) = 0

post a diversity Hamming distance constraint between a list of variables and a given fixed assignment



#### 1 Note

depending on the decomposition method, it adds dual and/or hidden variables

#### **Parameters**

- scope vector  $\alpha f$ variable indexes Weightedreturned by CSP::makeEnumeratedVariable
- distance the Hamming distance minimum bound
- **values** a vector of values (same size as scope)
- **method** the network decomposition method (0:Dual, 1:Hidden, 2:Ternary)

#### virtual vector<vector<int>> \*getListSuccessors() = 0

generating additional variables vector created when berge decomposition are included in the WCSP

#### virtual vector<int> getBergeDecElimOrder() = 0

return an elimination order compatible with Berge acyclic decomposition of global decomposable cost functions (if possible keep reverse of previous DAC order)

#### virtual void **setDACOrder**(vector<int> &elimVarOrder) = 0

change DAC order and propagate from scratch

#### virtual bool isKnapsack() = 0

true if there are knapsack constraints defined in the problem

#### virtual bool isGlobal() = 0

true if there are soft global constraints defined in the problem

#### virtual Cost **read\_wcsp**(const char \*fileName) = 0

load problem in all format supported by toulbar2. Returns the UB known to the solver before solving (file and command line).

#### virtual void **read\_legacy**(const char \*fileName) = 0

load problem in wcsp legacy format

#### virtual void **read\_uai2008**(const char \*fileName) = 0

load problem in UAI 2008 format (see http://graphmod.ics.uci.edu/uai08/FileFormat and http://www.cs. huji.ac.il/project/UAI10/fileFormat.php)

#### **A** Warning

UAI10 evidence file format not recognized by toulbar2 as it does not allow multiple evidence (you should remove the first value in the file)

virtual void **read\_random**(int n, int m, vector<int> &p, int seed, bool forceSubModular = false, string globalname = "") = 0

create a random WCSP with n variables, domain size m, array p where the first element is a percentage of tuples with a nonzero cost and next elements are the number of random cost functions for each different arity (starting with arity two), random seed, a flag to have a percentage (last element in the array p) of the binary cost functions being permutated submodular, and a string to use a specific global cost function instead of random cost functions in extension

virtual void **read\_wcnf**(const char \*fileName) = 0

load problem in (w)cnf format (see http://www.maxsat.udl.cat/08/index.php?disp=requirements)

virtual void **read\_qpbo**(const char \*fileName) = 0

load quadratic pseudo-Boolean optimization problem in unconstrained quadratic programming text format (first text line with n, number of variables and m, number of triplets, followed by the m triplets (x,y,cost) describing the sparse symmetric nXn cost matrix with variable indexes such that  $x \le y$  and any positive or negative real numbers for costs)

virtual void **read\_opb**(const char \*fileName) = 0

load pseudo-Boolean optimization problem

virtual void **read\_lp**(const char \*fileName) = 0

load integer linear programming problem

virtual const vector<Value> getSolution() = 0

after solving the problem, return the optimal solution (warning! do not use it if doing solution counting or if there is no solution, see *WeightedCSPSolver::solve* output for that)

virtual Double **getSolutionValue()** const = 0

returns current best solution cost or MAX\_COST if no solution found

virtual Cost getSolutionCost() const = 0

returns current best solution cost or MAX\_COST if no solution found

virtual const vector<Value> getSolution(Cost \*cost\_ptr) = 0

#### **Deprecated**

returns current best solution and its cost

virtual vector<pair<Double, vector<Value>>> **getSolutions()** const = 0

returns all solutions found

virtual void **initSolutionCost()** = 0

invalidate best solution by changing its cost to MAX\_COST

virtual void **setSolution**(Cost cost, TAssign \*sol = NULL) = 0

set best solution from current assigned values or from a given assignment (for BTD-like methods)

virtual void **printSolution()** = 0

prints current best solution on standard output (using variable and value names if cfn format and Toul-Bar2::showSolution>1)

```
virtual void printSolution(ostream &os) = 0
     prints current best solution (using variable and value names if cfn format and ToulBar2::writeSolution>1)
virtual void printSolution(FILE *f) = 0
     prints current best solution (using variable and value names if cfn format and ToulBar2::writeSolution>1)
virtual void print(ostream &os) = 0
     print current domains and active cost functions (see Output messages, verbosity options and debugging)
virtual void dump (ostream &os, bool original = true) = 0
     output the current WCSP into a file in wcsp format
         Parameters
              • os – output file
              • original – if true then keeps all variables with their original domain size else uses unas-
                signed variables and current domains recoding variable indexes
virtual void dump_CFN (ostream &os, bool original = true) = 0
     output the current WCSP into a file in wcsp format
         Parameters
              • os – output file
              • original – if true then keeps all variables with their original domain size else uses unas-
                signed variables and current domains recoding variable indexes
virtual Cost decimalToCost (const string &decimalToken, const unsigned int lineNumber) const = 0
virtual Cost DoubletoCost (const Double &c) const = 0
virtual Double Cost2ADCost (const Cost &c) const = 0
     converts an integer cost from a lower or upper bound of the whole problem into a real value
virtual Double Cost2RDCost (const Cost &c) const = 0
     converts an integer cost from a local cost function into a real value
virtual Cost Prob2Cost (TProb p) const = 0
virtual TProb Cost2Prob(Cost c) const = 0
virtual TLogProb Cost2LogProb(Cost c) const = 0
virtual Cost LogProb2Cost (TLogProb p) const = 0
virtual Cost LogSumExp(Cost c1, Cost c2) const = 0
virtual TLogProb LogSumExp(TLogProb logc1, Cost c2) const = 0
virtual TLogProb LogSumExp (TLogProb logc1, TLogProb logc2) const = 0
virtual void setLb(Cost newLb) = 0
virtual void setUb(Cost newUb) = 0
```

virtual void restoreSolution(Cluster \*c = NULL) = 0

virtual void **buildTreeDecomposition()** = 0

```
virtual TreeDecomposition *getTreeDec() = 0
virtual vector<Variable*> &getDivVariables() = 0
    returns all variables on which a diversity request exists
virtual void initDivVariables() = 0
    initializes diversity variables with all decision variables in the problem
virtual void iniSingleton() = 0
virtual void updateSingleton() = 0
virtual void removeSingleton() = 0
virtual void printVACStat() = 0

Public Static Functions
static WeightedCSP *makeWeightedCSP(Cost upperBound, void *solver = NULL)
Weighted CSP factory.
```

**CHAPTER** 

#### **FOUR**

#### WEIGHTEDCSPSOLVER CLASS

#### class WeightedCSPSolver

Abstract class WeightedCSPSolver representing a WCSP solver

- link to a WeightedCSP
- generic complete solving method configurable through global variables (see *ToulBar2* class and command line options)
- · optimal solution available after problem solving
- elementary decision operations on domains of variables
- statistics information (number of nodes and backtracks)
- problem file format reader (multiple formats, see Weighted Constraint Satisfaction Problem file format (wcsp))
- solution checker (output the cost of a given solution)

#### **Public Functions**

```
inline virtual ~WeightedCSPSolver()
virtual WeightedCSP *getWCSP() = 0
     access to its associated Weighted CSP
virtual Long getNbNodes() const = 0
     number of search nodes (see WeightedCSPSolver::increase, WeightedCSPSolver::decrease, WeightedCSP-
     Solver::assign, WeightedCSPSolver::remove)
virtual Long getNbBacktracks() const = 0
     number of backtracks
virtual void increase(int varIndex, Value value, bool reverse = false) = 0
     changes domain lower bound and propagates
virtual void decrease(int varIndex, Value value, bool reverse = false) = 0
     changes domain upper bound and propagates
virtual void assign(int varIndex, Value value, bool reverse = false) = 0
     assigns a variable and propagates
virtual void remove (int varIndex, Value value, bool reverse = false) = 0
     removes a domain value and propagates (valid if done for an enumerated variable or on its domain bounds)
```

```
virtual Cost read_wcsp(const char *fileName) = 0
     reads a Cost function network from a file (format as indicated by ToulBar2:: global variables)
virtual void read_random(int n, int m, vector<int> &p, int seed, bool forceSubModular = false, string
                           globalname = "") = 0
     create a random WCSP, see WeightedCSP::read_random
virtual bool solve(bool first = true) = 0
     simplifies and solves to optimality the problem
```



#### Warning

after solving, the current problem has been modified by various preprocessing techniques



#### Warning

DO NOT READ VALUES OF ASSIGNED VARIABLES USING WeightedCSP::getValue (temporally wrong assignments due to variable elimination in preprocessing) BUT USE Weighted CSP-Solver::getSolution INSTEAD

#### Returns

false if there is no solution found

virtual void **beginSolve**(Cost ub) = 0 virtual Cost **preprocessing**(Cost ub) = 0 virtual void **recursiveSolve**(Cost lb = MIN\_COST) = 0 virtual void **recursiveSolveLDS**(int discrepancy) = 0 virtual pair<Cost, Cost> hybridSolve() = 0 virtual void **endSolve**(bool isSolution, Cost cost, bool isComplete) = 0 virtual Cost **narycsp**(string cmd, vector<Value> &solution) = 0 solves the current problem using INCOP local search solver by Bertrand Neveu



#### 1 Note

side-effects: updates current problem upper bound and propagates, best solution saved (using WCSP::setBestValue)



#### Warning

cannot solve problems with global cost functions

- cmd command line argument for narycsp INCOP local search solver (cmd format: lowerbound randomseed nbiterations method nbmoves neighborhoodchoice neighborhoodchoice2 minnbneighbors maxnbneighbors neighborhoodchoice3 autotuning tracemode)
- solution best solution assignment found (MUST BE INITIALIZED WITH A DE-FAULT COMPLETE ASSIGNMENT)

#### Returns

best solution cost found

virtual Cost **pils**(string cmd, vector<Value> &solution) = 0

solves the current problem using PILS local search @ François Beuvin, David Simoncini, Sebastien Verel



#### Warning

cannot solve problems with non-binary cost functions

#### **Parameters**

cmd - command line argument for PILS local search solver (cmd format: nbruns perturb\_mode perturb\_strength flatMaxIter nbEvalHC nbEvalMax strengthMin strengthMax incrFactor decrFactor)

#### Returns

best solution cost found

virtual Cost **1rBCD**(string cmd, vector<Value> &solution) = 0

solves the current problem using LR-BCD local search @ Valentin Durante, George Katsirelos, Thomas Schiex



#### Warning

cannot solve problems with non-binary cost functions

#### **Parameters**

cmd - command line argument for LR-BCD local search solver (cmd format: maxiter rank nbroundings)

#### Returns

best solution cost found

virtual bool **solve\_symmax2sat**(int n, int m, int \*posx, int \*posy, double \*cost, int \*sol) = 0

quadratic unconstrained pseudo-Boolean optimization Maximize  $h' \times W \times h$  where W is expressed by all its non-zero half squared matrix costs (can be positive or negative, with  $\forall i, posx[i] < posy[i]$ )



#### See also

::solvesymmax2sat\_ for Fortran call

1 Note

costs for  $posx \neq posy$  are multiplied by 2 by this method

1 Note

by convention:  $h = 1 \equiv x = 0$  and  $h = -1 \equiv x = 1$ 

Warning

does not allow infinite costs (no forbidden assignments, unconstrained optimization)

#### Returns

true if at least one solution has been found (array sol being filled with the best solution)

virtual void **dump\_wcsp**(const char \*fileName, bool original = true, ProblemFormat format = WCSP FORMAT) = 0

output current problem in a file

See also

WeightedCSP::dump

virtual void **read\_solution**(const char \*fileName, bool updateValueHeuristic = true) = 0 read a solution from a file

virtual void **parse\_solution**(const char \*certificate, bool updateValueHeuristic = true) = 0 read a solution from a string (see *ToulBar2* option -x)

virtual const vector<Value> getSolution() = 0

after solving the problem, return the optimal solution (warning! do not use it if doing solution counting or if there is no solution, see WeightedCSPSolver::solve output for that)

virtual Double getSolutionValue() const = 0

after solving the problem, return the optimal solution value (can be an arbitrary real cost in minimization or preference in maximization, see CFN format) (warning! do not use it if doing solution counting or if there is no solution, see WeightedCSPSolver::solve output for that)

virtual Cost getSolutionCost() const = 0

after solving the problem, return the optimal solution nonnegative integer cost (warning! do not use it if doing solution counting or if there is no solution, see WeightedCSPSolver::solve output for that)

virtual Cost **getSolution**(vector<Value> &solution) const = 0

#### **Deprecated**

after solving the problem, add the optimal solution in the input/output vector and returns its optimum cost (warning! do not use it if doing solution counting or if there is no solution, see WeightedCSPSolver::solve output for that)

virtual vector<pair<Double, vector<Value>>> **getSolutions()** const = 0

after solving the problem, return all solutions found with their corresponding value

virtual Double **getDDualBound()** const = 0

after (partially) solving the problem (possibly interrupted before the search is complete), return a global problem dual bound as a Double representing a decimal cost (lower resp. upper bound for minimization resp. maximization)

virtual set<int> getUnassignedVars() const = 0

virtual int numberOfUnassignedVariables() const = 0

#### **Public Static Functions**

static WeightedCSPSolver \*makeWeightedCSPSolver (Cost initUpperBound, WeightedCSP \*wcsp = NULL) WeightedCSP Solver factory.

**CHAPTER** 

**FIVE** 

#### **MULTICFN CLASS**

#### class MultiCFN

store a combination of several cost function networks. one new class: multiwcsp to create a wcsp as the linear combination of wcsp's given as input makeWeightedCSPSolver: possibility to give a wcsp as input, which will be used as the instance solved by the solver (works only with the base Solver class, otherwise, a new weightedCSP is created withinin the solver object) adding a function for the python API for weightedCSP to read an instance independently from the solver adding the multiwcsp object and methods in the python API modification of the python API in makeWeightedCSPSolver to take a weightedCSP as input modification of the python API to create a weightedCSP object without solvers modification of the python API to read an instance file directly from a weightedCSP object

#### **Public Types**

typedef std::map<std::string, std::string> Solution

#### **Public Functions**

#### MultiCFN()

default constructor

MultiCFN(std::vector<WCSP\*> &wcsps, std::vector<Double> &weights)

constructor: build a multicfn combining wcsps given as input

#### **Parameters**

- wcsps a vector of wcsp objects to combine with weighted sums
- weights a list of weights for each wcsp

#### ~MultiCFN()

destructor (delete all constraints)

void push\_back(WCSP \*wcsp, Double weight = 1.0)

add a wcsp to the network, create the variables if they do not exist, the wcsp is stored internally, the original wcsp will not be referenced

- wcsp the wcsp to add
- weight the weight of the wcsp in the objective function (sum of the cost functions)

#### void setWeight(unsigned int wcsp\_index, Double weight)

set the weight of the cost functions of one of the network

#### **Parameters**

- wcsp\_index the index of the network to modify
- weight the new weight

#### Double **getWeight** (unsigned int wcsp\_index)

get the wieght of a network

#### **Parameters**

wcsp\_index - the index of the network (by adding order)

#### **Returns**

the weight assigned to the network

#### unsigned int nbNetworks()

number of networks loaded in the combiner

#### Returns

the number of network

#### unsigned int nbVariables()

number of variables in the problem

#### Returns

the number of variables

#### unsigned int nbCostFunctions()

number of cost functions in the problem

#### Returns

the number of cost functions

#### std::string getNetworkName(unsigned int index)

get the name of one of the network added to the multiwcsp

#### **Parameters**

index – the index of the network

#### Returns

the name associated to the network

#### int getVariableIndex(std::string name)

get the index of one of the variable added to the multiwesp

#### **Parameters**

**name** – the name of the variable

#### Returns

the index associated to the variable or -1 if not found

#### unsigned int **nbValues** (unsigned int index)

number of values in the domain of the variable

#### **Parameters**

**index** – the index of the variable (must be lower than *nbVariables()*)

#### Returns

the number of values in the domain of the variable

```
std::vector<unsigned int> getScope(unsigned int index)
     scope of the cost function
          Parameters
              index – the index of the cost function (must be lower than nbCostFunctions())
         Returns
              the list of variable indexes in the scope of the cost function
mcriteria::CostFunction::Type getType(unsigned int index)
     type of the cost function
         Parameters
              index – the index of the cost function (must be lower than nbCostFunctions())
         Returns
              the type of the cost function
Double getCost (unsigned int index, std::vector<unsigned int> &tuple)
     return the cost of a given tuple
         Parameters
              • index – the index of the cost function (must be lower than nbCostFunctions())
              • tuple – the tuple
unsigned int getDecimalPoint()
     return the precision used in the combined wcsp (max of the decimalPoint of the wcsp given as input)
Double getUnitCost()
     get the wcsp's unit cost precision as double
void print(std::ostream &os)
     print the cfn
     os the stream to print to
```

make a wcsp from the convex combination of all the wcsps

#### **Parameters**

- **vars** the optional set of variable indexes to extract the induced graph (if missing or empty then no restriction)
- **scopes** the optional list of allowed scopes to extract the partial graph (if missing or empty then no restriction)
- **constrs** the optional list of allowed cost function indexes (same index as in cfn dump file) to extract the partial graph (if missing or empty then no restriction)

 $\label{eq:const} $$ void \ \textbf{makeWeightedCSP} * wcsp, const set<unsigned int> \& vars = \{\}, const vector<set<unsigned int> \& constrained int> \& c$ 

fill a wcsp with the convex combination of all the wcsps already added

- wcsp the weighted csp to be filled
- **vars** the optional set of variable indexes to extract the induced graph (if missing or empty then no restriction)

- **scopes** the optional list of allowed scopes to extract the partial graph (if missing or empty then no restriction)
- **constrs** the optional list of allowed cost function indexes (same index as in cfn dump file) to extract the partial graph (if missing or empty then no restriction)

unsigned int **tupleToIndex**(const std::vector<mcriteria::Var\*> &variables, const std::vector<unsigned int> &tuple)

convert a tuple to a cost index, rightmost value indexed first

#### **Parameters**

- variables the list of variables from the tuple
- tuple the tuple: value indexes for each variable

#### Returns

the index corresponding to the tuple

#### Solution getSolution()

get the solution of the created wcsp after being solved

#### Returns

the solution as a dictionary of variable names/value names

#### Pre

the wcsp must have been solved and not been deleted

#### std::vector<Double> getSolutionValues()

get the objective values of the different cost function networks from the created wcsp after being solved

#### Returns

the objective values as a dictionary of variable names/value names

#### Pre

the wcsp must have been solved and not been deleted

#### std::vector<Double> computeSolutionValues(Solution &solution)

compute the values of an existing solution

#### **Parameters**

**solution** – the solution given

#### Returns

the costs of the solution

void outputNetSolutionCosts(size\_t index, Solution &solution)

#### Solution convertToSolution(std::vector<Value> &solution)

convert a solution returned by ToulBar2 to a dictionary with variable names and values as labels

#### **Parameters**

**solution** – the solution given bu *ToulBar2* 

#### Returns

the solution as a dictionary

#### void **setNoiseActivation**(bool activation)

activate the noise parameter to add uniform random noise to each cost in every cost function

#### **Parameters**

**activation** – true activates the noise

void **setNoiseLevel**(Double min\_level, Double max\_level) set the min and max value for uniform noise perturbation of the costs

#### **Public Members**

std::vector<mcriteria::Var> var

std::map<std::string, int> var\_index

std::vector<mcriteria::CostFunction\*> cost\_function

std::map<std::string, unsigned int> cost\_function\_index

#### **TOULBAR2 CLASS**

#### class ToulBar2

It contains all toulbar2 global variables encapsulated as static class members of this class.

Each variable may correspond to some command-line option of toulbar2 executable.

#### **Public Static Functions**

```
static inline Double getCostMultiplier()
```

static inline Cost getCostMultiplierInt()

static inline void **setCostMultiplier**(Double mult)

#### **Public Static Attributes**

#### static string **version** = Toulbar\_VERSION

toulbar2 version number

#### static int verbose

verbosity level (-1:no output, 0: new solutions found, 1: choice points, 2: current domains, 3: basic EPTs, 4: active cost functions, 5: detailed cost functions, 6: more EPTs, 7: detailed EPTs) (command line option -v)

#### static bool FullEAC

VAC-integrality/Full-EAC variable ordering heuristic (command line option -vacint and optionally -A)

#### static bool VACthreshold

automatic threshold cost value selection for VAC during search (command line option -vacthr)

static int nbTimesIsVAC

static int nbTimesIsVACitThresholdMoreThanOne

static bool RASPS

#### static int useRASPS

VAC-based upper bound probing heuristic (0: no rasps, 1: rasps using DFS, >1: using LDS with bounded discrepancy + 1) (command line option -raspslds or -rasps)

#### static bool RASPSreset

reset weighted degree variable ordering heuristic after doing upper bound probing (command line option -raspsini)

#### static int RASPSangle

automatic threshold cost value selection for probing heuristic (command line option -raspsdeg)

#### static Long RASPSnbBacktracks

number of backtracks of VAC-based upper bound probing heuristic (command line option -rasps)

#### static int RASPSnbStrictACVariables

#### static Cost RASPSlastitThreshold

#### static bool RASPSsaveitThresholds

static vector<pair<Cost, Double>> RASPSitThresholds

#### static int debug

debug mode(0: no debug, 1: current search depth and statics on nogoods for BTD, 2: idem plus some information on heuristics, 3: idem plus save problem at each node if verbose >= 1) (command line option -Z)

#### static string externalUB

initial upper bound in CFN format

#### static int showSolutions

shows each solution found (0: nothing, 1: value indexes, 2: value names, 3: variable&value names) (command line option -s)

#### static bool showHidden

shows hidden variables for each solution found (command line option -s with a negative value)

#### static int writeSolution

writes each solution found (0: nothing, 1: value indexes, 2: value names, 3: variable&value names) (command line option -w)

#### static FILE \*solutionFile

#### static long solutionFileRewindPos

#### static Long allSolutions

finds at most a given number of solutions with a cost strictly lower than the initial upper bound and stops (or counts the number of zero-cost satisfiable solutions in conjunction with BTD) (command line option -a)

#### static int dumpWCSP

saves the problem in wcsp (0: do not save, 1: original or 2: after preprocessing) or cfn (3: original or 4: after preprocessing) format (command line option -z)

#### static bool dumpOriginalAfterPreprocessing

saves the problem with initial domains after preprocessing (used in conjunction with dumpWCSP)

#### static bool approximateCountingBTD

approximate zero-cost satisfiable solution counting using BTD (command line options -D and -a and -B=1)

#### static bool binaryBranching

tree search using binary branching instead of n-ary branching for enumerated domains (command line option -b)

#### static int dichotomicBranching

tree search using dichotomic branching if current domain size is strictly greater than Toul-Bar2::dichotomicBranchingSize (0: no dichotomic branching, 1: splitting in the middle of domain range, 2: splitting in the middle of sorted unary costs) (command line option -d)

#### static unsigned int dichotomicBranchingSize

dichotomic branching threshold (related to command line option -d)

#### static bool sortDomains

sorts domains in preprocessing based on increasing unary costs (command line option -sortd)



#### Warning

Works only for binary WCSPs.

static map<int, ValueCost\*> sortedDomains

#### static bool solutionBasedPhaseSaving

solution-based phase saving value heuristic (command line option -solr)

#### static Double bisupport

value heuristic in bi-objective optimization when the second objective is encapsulated by a bounding constraint (command line option -bisupport)

#### static int elimDegree

boosting search with variable elimination of small degree (0: no variable elimination, 1: linked to at most

one binary cost function, 2: linked to at most two binary cost functions, 3: linked to at most one ternary cost function and two scope-included cost functions) (command line option -e)

#### static int elimDegree\_preprocessing

in preprocessing, generic variable elimination of degree less than or equal to a given value (0: no variable elimination) (command line option -p)

static int elimDegree\_

static int elimDegree\_preprocessing\_

#### static int elimSpaceMaxMB

maximum space size for generic variable elimination (in MegaByte) (related to command line option -p)

#### static int minsumDiffusion

in preprocessing, applies Min Sum Diffusion algorithm a given number of iterations (command line option -M)

#### static int preprocessTernaryRPC

in preprocessing, simulates restricted path consistency by adding ternary cost functions on most-promising triangles of binary cost functions (maximum space size in MegaByte) (command line option -t)

#### static int hve

hidden variable encoding into a binary WCSP

#### static int pwc

pairwise consistency by dual encoding into a binary WCSP

#### static bool pwcMinimalDualGraph

minimizes dual intersection graph by removing redundant edges

#### static int preprocessFunctional

in preprocessing, applies variable elimination of 0: no variable, 1: functional, or 2: bijective variables, or 3:functional before and after PWC (command line option -f)

#### static bool costfuncSeparate

in preprocessing, applies pairwise decomposition of non-binary cost functions (command line option -dec)

#### static int preprocessNary

in preprocessing, projects n-ary cost functions on all their scope-included binary cost functions if n is lower than a given value (0: no projection) (command line option -n)

#### static bool QueueComplexity

ensures optimal worst-case time complexity of DAC and EAC (command line option -o)

#### static bool Static\_variable\_ordering

tree search using a static variable ordering heuristic (same order as DAC) (command line option -svo)

#### static bool lastConflict

tree search using binary branching with last conflict backjumping variable ordering heuristic (command line options -c and -b)

#### static int weightedDegree

weighted degree variable ordering heuristic if the number of cost functions is less than a given value (command line option -q)

#### static int weightedTightness

in preprocessing, initializes weighted degrees associated to cost functions by their 1: average or 2: median costs (command line options -m and -q)

#### static int constrOrdering

in preprocessing, sorts constraints based on 0: do not sort, 1: lexicographic ordering, 2: decreasing DAC ordering, 3: decreasing constraint tightness, 4: DAC then tightness, 5: tightness then DAC, 6: random order, or the opposite order if using a negative value (command line option -sortc)

#### static bool MSTDAC

maximum spanning tree DAC ordering (command line option -mst)

#### static int **DEE**

soft neighborhood substitutability, a.k.a., dead-end elimination (0: no elimination, 1: restricted form during search, 2: full in preprocessing and restricted during search, 3: full always, 4: full in preprocessing) (command line option -dee)

static int DEE\_

#### static int nbDecisionVars

tree search by branching only on the first variables having a lexicographic order position below a given value, assuming the remaining variables are completely assigned by this first group of variables (0: branch on all variables) (command line option -var)

#### static int 1ds

iterative limited discrepancy search (0: no LDS), use a negative value to stop the search after the given absolute number of discrepancies has been explored (command line option -l)

#### static bool limited

#### static Long restart

randomly breaks ties in variable ordering heuristics and Luby restarts until a given number of search nodes (command line option -L)

#### static Long backtrackLimit

limit on the number of backtracks (command line option -bt)

#### static externalevent setvalue

```
static externalevent setmin
static externalevent setmax
static externalevent removevalue
static externalcostevent setminobj
static externalsolution newsolution
static Pedigree *pedigree
static Haplotype *haplotype
static string map_file
static bool cfn
static bool gz
static bool bz2
static bool xz
static bool bayesian
static int uai
static int resolution
     defines the number of digits that should be representable in UAI/OPB/QPBO/WBO formats (command line
     option -precision)
static bool resolution_Update
static TProb errorg
static TLogProb NormFactor
static int foundersprob_class
     Allele frequencies of founders
       • 0: equal frequencies
```

• 1: probs depending on the frequencies found in the problem

• otherwise: read probability distribution from command line

# static vector<TProb> allelefreqdistrib static bool consecutiveAllele

#### static int pedigreeCorrectionMode

#### static int pedigreePenalty

static bool generation

#### static int vac

enforces VAC at each search node having a search depth less than the absolute value of a given value (0: no VAC, 1: VAC in preprocessing, >1: VAC during search up to a given search depth), if given a negative value then VAC is not performed inside depth-first search of hybrid best-first search method (command line option -A and possibly -hbfs)

#### static int vac\_prev

#### static string costThresholdS

threshold cost value for VAC in CFN format (command line option -T)

#### static string costThresholdPreS

in preprocessing, threshold cost value for VAC in CFN format (command line option -P)

#### static Cost costThreshold

threshold cost value for VAC (command line option -T)

#### static Cost costThresholdPre

in preprocessing, threshold cost value for VAC (command line option -P)

#### static Double trwsAccuracy

in preprocessing, enforces TRW-S until a given accuracy is reached (command line option -trws)

#### static bool trwsOrder

replaces DAC order by Kolmogorov's TRW-S order (command line option —trws-order)

#### static unsigned int trwsNIter

enforces at most n iterations of TRW-S (command line option —trws-n-iters)

#### static unsigned int trwsNIterNoChange

stops TRW-S when n iterations did not change the lower bound (command line option —trws-n-iters-no-change)

```
static unsigned int trwsNIterComputeUb
     computes an upper bound every n steps in TRW-S (command line option —trws-n-iters-compute-ub)
static Double costMultiplier
     multiplies all costs internally by this number when loading a problem in WCSP format (command line
     option -C)
static Cost costMultiplier_
static unsigned int decimalPoint
static string deltaUbS
     stops search if the absolute optimality gap reduces below a given value in CFN format (command line
     option -agap)
static Cost deltaUb
static Cost deltaUbAbsolute
     stops search if the absolute optimality gap reduces below a given value (command line option -agap)
static Double deltaUbRelativeGap
     stops search if the relative optimality gap reduces below a given value (command line option -rgap)
static int singletonConsistency
     in preprocessing, performs singleton soft local consistency (command line option -S)
static int vacValueHeuristic
     VAC-based and Knapsack value (and variable) ordering heuristics (command line options -V and -A)
static BEP *bep
static LcLevelType LcLevel
     soft local consistency level (0: NC, 1: AC, 2: DAC, 3: FDAC, 4: EDAC) (command line option -k)
static LcLevelType LcLevel_prev
static int maxEACIter
     maximum number of iterations in EDAC before switching to FDAC
static bool wcnf
static bool qpbo
```

### $static\ Double\ \textbf{qpboQuadraticCoefMultiplier}$

defines coefficient multiplier for quadratic terms in QPBO format (command line option -qpmult)

#### static bool opb

#### static bool cardinality

decomposes cardinality equality constraints into a network of binary and ternary constraints (using wsum and wvarsum)

#### static bool 1p

#### static int addAMOConstraints

automatically detects and adds at-most-one constraints to existing knapsack constraints

#### static bool addAMOConstraints

automatically detects and adds at-most-one constraints to existing knapsack constraints

#### static int knapsackDP

solves exactly knapsack constraints using dynamic programming (at every search node or less often)

#### static bool VAClin

solves exactly knapsack constraints using dynamic programming (at every search node or less often)

#### static unsigned int divNbSol

upper bound on the number of diverse solutions (0: no diverse solution) (keep it small as it controls model size)

#### static unsigned int divBound

minimum Hamming distance between diverse solutions (command line options -div and -a)

#### static unsigned int divWidth

adds a global MDD constraint with a given maximum relaxed width for finding diverse solutions (command line option -mdd)

#### static unsigned int divMethod

diversity encoding method (0: Dual, 1: Hidden, 2: Ternary, 3: Knapsack) (command line option -divm)

#### static unsigned int divRelax

MDD relaxation heuristic (0: random, 1: high diversity, 2: small diversity, 3: high unary costs) (command line option -mddh)

#### static char \*varOrder

variable elimination order for DAC, BTD, and VNS methods (0: lexicographic ordering, -1: maximum cardinality search ordering, -2: minimum degree ordering, -3: minimum fill-in ordering, -4: maximum spanning tree ordering, -5: reverse Cuthill-Mckee ordering, -6: approximate minimum degree ordering, -7: same as 0, 8: lexicographic ordering using variable names, string: variable ordering filename) (command line option -O)

#### static int btdMode

tree search exploiting tree/path decomposition (0: no tree decomposition, 1: BTD with tree decomposition, 2: RDS-BTD with tree decomposition, 3: RDS-BTD with path decomposition) (command line option -B)

#### static int btdSubTree

in RDS-BTD, cluster index for solving only this particular rooted cluster subtree (command line option -I)

#### static int btdRootCluster

chooses the root cluster index (command line option -R)

#### static int rootHeuristic

root cluster heuristic (0: maximum size, 1: maximum ratio of size by height-size, 2: minimum ratio of size by height-size, 3: minimum height) (command line option -root)

#### static bool reduceHeight

minimize cluster tree height when searching for the root cluster (command line option -minheight)

static bool maxsateval

static bool xmlflag

static bool xmlcop

static TLogProb markov\_log

static string evidence\_file

static FILE \*solution\_uai\_file

static string **solution\_uai\_filename** 

static string problemsaved\_filename

#### static bool isZ

computes logarithm of probability of evidence (a.k.a. log-partition function) in UAI format (command line option -logz)

static TLogProb logZ

static TLogProb logU

#### static TLogProb logepsilon

approximation factor for computing the log-partition function (command line option -epsilon)

#### static Double epsilon

floating-point epsilon

static bool uaieval

#### static string **stdin\_format**

file format used when reading a problem from a Unix pipe ("cfn", "wcsp", "uai", "LG", "cnf", "wcnf", "qpbo", "opb", "wbo", "lp") (command line option —stdin)

static double startCpuTime

static double startRealTime

static double startRealTimeAfterPreProcessing

#### static int splitClusterMaxSize

splits large clusters into a chain of smaller embedded clusters with a number of proper variables less than a given value (command line option -j)

#### static double **boostingBTD**

in BTD, merges recursively leaf clusters with their fathers if separator size smaller than *Toul-Bar2::elimDegree*, else in VNS, merges clusters if the ratio of number of separator variables by number of cluster variables is above a given threshold (command line option -E and possibly -e)

#### static int maxSeparatorSize

merges recursively clusters with their fathers if separator size greater than a given threshold (command line option -r)

#### static int minProperVarSize

merges recursively clusters with their fathers if the number of proper variables is less than a given threshold (command line option -X)

#### static bool heuristicFreedom

merges clusters automatically to give more freedom to variable ordering heuristics in BTD methods (command line option -F)

#### static int heuristicFreedomLimit

stops merging a cluster subtree during BTD search if we tried repeatedly to solve this cluster for the same separator assignment more than a given number of times (-1: no merging) (command line option -F)

static bool Berge\_Dec

static bool learning

static externalfunc timeOut

#### static std::atomic<bool> interrupted

#### static int seed

initial random seed value, or use current time if a negative value is given (command line option -seed)

#### static Double sigma

initial random noise standard deviation to be added to energy values when reading UAI format files (command line option -sigma)

#### static string incop\_cmd

in preprocessing, executes INCOP local search method to produce a better initial upper bound (default parameter string value "0 1 3 idwa 100000 cv v 0 200 1 0 0", see INCOP user manual http://imagine.enpc.fr/~neveub/incop/incop1.1/usermanual.ps) (command line option -i)

#### static string pils\_cmd

in preprocessing, executes PILS local search method to produce a better initial upper bound (default parameter string value "3 0 0.333 150 150 1500 0.1 0.5 0.1 0.1", see PILS article https://doi.org/10.1002/prot.26174) (command line option -pils)

#### static string lrBCD\_cmd

in preprocessing, executes LR-BCD to produce a better initial upper bound (default parameter string value "5 -2 3") (command line option -lrBCD)

#### static SearchMethod searchMethod

chooses between tree search and variable neighborhood search methods (0: tree search, 1: sequential unified VNS, 2: sequential unified decomposition guided VNS, 3: synchronous parallel UDGVNS, 4: asynchronous parallel UDGVNS, 5: tree decomposition heuristic) (command line option -vns)

#### static string clusterFile

cluster tree decomposition filename in COV or DEC format (with or without running intersection property)

#### static ofstream vns0utput

#### static VNSSolutionInitMethod vnsInitSol

initial solution for VNS-like methods (-1: random, -2: minimum domain values, -3: maximum domain values, -4: first solution found by DFS, >=0: or by LDS with at most n discrepancies (command line option -vnsini)

#### static int vnsLDSmin

minimum discrepancy value for VNS-like methods (command line option -ldsmin)

#### static int vnsLDSmax

maximum discrepancy value for VNS-like methods (command line option -ldsmax)

#### static VNSInc vnsLDSinc

discrepancy increment strategy for VNS-like methods (1: Increment by 1, 2: Multiply by 2, 3: Luby operator) (command line option -ldsinc)

#### static int vnsKmin

minimum neighborhood size for VNS-like methods (command line option -kmin)

#### static int vnsKmax

maximum neighborhood size for VNS-like methods (command line option -kmax)

#### static VNSInc vnsKinc

neighborhood size increment strategy for VNS-like methods (1: Increment by 1, 2: Multiply by 2, 3: Luby operator, 4: Increment by 1 until maximum cluster size then considers all variables) (command line option -kinc)

static int vnsLDScur

static int vnsKcur

#### static VNSVariableHeuristic vnsNeighborVarHeur

neighborhood heuristic method (0: random variables, 1: variables in conflict, 2: connected variables in conflict, 3: random cluster, 4: variables in conflict with maximum degree, 5: sorted cluster, 6: sorted cluster separator, 7: similar to 6, 8: randomized root cluster, 9: variables in partial conflict)

#### static bool vnsNeighborChange

static bool vnsNeighborSizeSync

static bool vnsParallelLimit

static bool vnsParallelSync

#### static string vnsOptimumS

stops VNS if a solution is found with a given cost (or better) in CFN format (command line option -best)

#### static Cost vnsOptimum

stops VNS if a solution is found with a given cost (or better) (command line option -best)

#### static bool parallel

parallel mode for tree search and VNS (see mpirun toulbar2)

#### static Long **hbfs**

performs hybrid best-first search with a given limit in the number of backtracks for depth-first search before visiting another open node (0: always DFS, 1: HBFS) (related to command line option -hbfs)

#### static Long hbfsGlobalLimit

restarts BTD-HBFS from the root cluster after a given number of backtracks (command line option -hbfs)

#### static Long hbfsAlpha

minimum recomputation node redundancy percentage threshold value (command line option -hbfsmin)

#### static Long hbfsBeta

maximum recomputation node redundancy percentage threshold value (command line option -hbfsmax)

#### static ptrdiff t hbfsCPLimit

maximum number of stored choice points before switching to normal DFS (warning! should always be cast to std::size\_t when used, so that -1 is equivalent to +infinity)

#### static ptrdiff\_t hbfsOpenNodeLimit

maximum number of stored open nodes before switching to normal DFS (command line option -open) (warning! should always be cast to std::size\_t when used, so that -1 is equivalent to +infinity)

#### static Long sortBFS

number of visited open nodes before sorting the remaining open nodes (command line option -sopen)

#### static Long eps

performs HBFS until a given number of open nodes are collected and exits (command line option -eps)

#### static string **epsFilename**

a given filename to print remaining valid (lower bound less than current upper bound) open nodes as partial assignments before exits (command line option -eps)

#### static bool verifyOpt

compiled in debug, checks if a given (optimal) solution is never pruned by propagation when the current upper bound is greater than the cost of this solution (see Solver::read\_solution, related to command line option -opt)

#### static Cost verifiedOptimum

compiled in debug, a given (optimal) solution cost (see Solver::read\_solution, related to command line option -opt)

#### static int bilevel

bilevel optimization using modified BTD with (at least) four clusters (P0 -> P1, P0 -> P2, P0 -> NegP2) corresponding to the restricted leader problem (P0 and P1), the follower problem (P2), and the negative follower problem (NegP2) (command line option -bilevel)

static vector<unsigned int> decimalPointBLP

static vector<Double> costMultiplierBLP

static vector<Cost> negCostBLP

static vector<Cost> initialLbBLP

static vector<Cost> initialUbBLP

CHAPTER

## **SEVEN**

## **MISCELLANEOUS FUNCTIONS**

#### void tb2init()

initialization of *ToulBar2* global variables (needed by numberjack/toulbar2) initialization of *ToulBar2* global variables (needed by numberjack/toulbar2)

#### void tb2checkOptions()

checks compatibility between selected options of *ToulBar2* (needed by numberjack/toulbar2) checks compatibility between selected options of *ToulBar2* (needed by numberjack/toulbar2)

## **INDEX**

MultiCFN::makeWeightedCSP (C++ function), 30  DECIMAL_POINT (C++ member), 4  H  HIDDEN_VAR_TAG (C++ member), 3  HIDDEN_VAR_TAG_HVE (C++ member), 3  HIDDEN_VAR_TAG_HVE (C++ member), 3  IMPLICIT_VAR_TAG_C++ member), 3  L  LARGE_NB_VARS (C++ member), 4  MAX_ARITY (C++ member), 4  MAX_ARITY (C++ member), 3  MAX_ARITY (C++ member), 3  MAX_ARITY (C++ member), 3  MAX_ARITY (C++ member), 3  MAX_ARITY (C++ member), 4  MAX_BRANCH_SIZE (C++ member), 3  MAX_DOMANL_SIZE (C++ member), 4  MAX_BRANCH_SIZE (C++ member), 4  MAX_BLIM_BIN (C++ member), 3  MAX_VAL (C++ member), 3  MAX_VAL (C++ member), 3  MAX_VAL (C++ member), 3  MAX_VAL (C++ member), 3  MILICFN::consputeSolution (C++ function), 28  MultiCFN::consputeSolution (C++ function), 31  MultiCFN::consputeSolution (C++ function), 31  MultiCFN::consputeSolution (C++ function), 31  MultiCFN::cost_function index (C++ function), 31  MultiCFN::cost_function index (C++ function), 30  MultiCFN::getNoiseActivation (C++ function), 30  MultiCFN::var_index (C++ function), 31  MRYPROJECTIONEXADOMSIZE (C++ member), 3  MARYPROJECTIONSIZE (C++ member), 3  NARYPROJECTIONSIZE (C++ member), 3  NARYPROJECTIONSIZE (C++ member), 3  NARYPROJECTIONSIZE (C++ member), 3  MARYPROJECTIONSIZE (C++ member), 3  MILICFN::cost_function index (C++ function), 30  MultiCFN::getNoiseActivation, 29  MultiCFN::mbValues(C++ function), 29  MultiCFN::makeWeightedCSP (C++ function), 29  MultiCFN::mbValues(C++ function), 29  MultiCFN::mbValues(C++ function), 29  MultiCFN::mbValues(C++ function), 29  MultiC	C	MultiCFN::getVariableIndex (C++ function), 29
DDECIMAL_POINT (C++ member), 4 HIDDEN_VAR_TAG (C++ member), 3 HIDDEN_VAR_TAG_HVE (C++ member), 3 HIDDEN_VAR_TAG_HVE_PRE (C++ member), 3 IMPLICIT_VAR_TAG (C++ member), 3 L LARGE_NB_VARS (C++ member), 4 MAX_RARITY (C++ member), 4 MAX_RARITY (C++ member), 3 MAX_DOMAIN_SIZE (C++ member), 3 MAX_EAC_ITER (C++ member), 3 MAX_LICEN_ESCENCE, 4 MAX_RAN_UC++ member), 3 MAX_LICEN_ESCENCE, 4 MAX_VAL (C++ member), 3 MAX_VAL	CHOICE_POINT_LIMIT ( $C++$ member), 3	MultiCFN::getWeight(C++ function), 29 MultiCFN::makeWeightedCSP(C++ function) 30
MultiCFN::nbCostFunctions (C++ function), 29 MultiCFN::nbVarLetons (C++ function), 29 MultiCFN::push_back (C++ function), 30 MultiCFN::push_back (C++ function), 31 MultiCFN::push_back (C++ function), 31 MultiCFN::setNoiseLevel (C++ function), 31 MultiCFN::setNoiseLevel (C++ function), 31 MultiCFN::setWeight (C++ function), 38 MultiCFN::varletons (C++ function), 30 MultiCFN::varletons (C++ function), 30 MultiCFN::varletons (C++ member), 3 MAX_DOMAIN_SIZE (C++ member), 3 MAX_PROJECTIONSMAXDOMSIZE (C++ member), 3 MAX_PROJECTIONSIZE (C++ member), 3 MAX_PROJECTIONSIZE (C++ member), 3 MAX_PROJECTIONSIZE (C++ member), 3 MAX_PROJECTIONSIZE (C++ member), 3 MultiCFN::convertToSolution (C++ function), 30 MultiCFN::ost_function_index (C++ function), 30 MultiCFN::pathsevel (-+ function), 30 MultiCFN::pathsevel (-+ function), 48  T tb2checkOptions (C++ function), 48	D	• • • • • • • • • • • • • • • • • • • •
HIDDEN_VAR_TAG (C++ member), 3 HIDDEN_VAR_TAG_HVE (C++ member), 3 HIDDEN_VAR_TAG_HVE_PRE (C++ member), 3 HIDDEN_VAR_TAG_HVE_PRE (C++ member), 3  IMPLICIT_VAR_TAG_C++ member), 3 L LARGE_NB_VARS (C++ member), 4 MAX_ARITY (C++ member), 4 MAX_BRANCH_SIZE (C++ member), 3 MAX_DOMAIN_SIZE (C++ member), 3 MAX_ECTER (C++ member), 3 MAX_ECTER (C++ member), 3 MAX_ELIM_BIN (C++ member), 3 MAX_ELIM_BIN (C++ member), 3 MAX_VAL (C++ member), 3 MULTICFN::obst_duction (C++ function), 28 MultiCFN::osst_MultiCFN:C++ function), 28 MultiCFN::osst_MultiCFN:C++ function), 30 MultiCFN::osst_function (C++ function), 31 MultiCFN::osst_function index (C++ function), 30 MultiCFN::getDecimalPoint (C++ function), 30 MultiCFN::getDecimalPoint (C++ function), 29 MultiCFN::getDecimalPoint (C++ function), 30 MultiCFN::getDecimalPoint (C++ function), 30 MultiCFN::getDecimalPoint (C++ function), 29 MultiCFN::getDecimalPoint (C++ function), 30 MultiCFN::getDecimalPoin	_	· · · · · · · · · · · · · · · · · · ·
MultiCFN::nbVariables (\$\hat{C}\$+ function), 29  MultiCFN::print (\$\hat{C}\$+ function), 30  MultiCFN::psthoiseActivation (\$\hat{C}\$+ function), 31  MultiCFN::setWoiseActivation (\$\hat{C}\$+ function), 31  MultiCFN::setWoiseLevel (\$\hat{C}\$+ function), 38  MultiCFN::setWoiseLevel (\$\hat{C}\$+ function), 39  MultiCFN::setWoiseLevel (\$\hat{C}\$+ function), 39  MultiC	DECIMAL_POINT (C++ member), 4	
HIDDEN_VAR_TAG (C++ member), 3 HIDDEN_VAR_TAG_HVE (C++ member), 3 HIDDEN_VAR_TAG_HVE (C++ member), 3 HIDDEN_VAR_TAG_HVE_PRE (C++ member), 3  I IMPLICIT_VAR_TAG (C++ member), 3  L LARGE_NB_VARS (C++ member), 4  MAX_BARANCH_SIZE (C++ member), 3  MAX_DOMAIN_SIZE (C++ member), 3  MAX_ELIM_BIN (C++ member), 3  MAX_ELIM_BIN (C++ member), 3  MAX_DAL (C++ member), 3  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN::convertToSolution (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::computeSolution (C++ function), 31  MultiCFN::computeSolution (C++ function), 31  MultiCFN::cost_function (C++ function), 32  MultiCFN::cost_function index (C++ member), 32  MultiCFN::cost_function index (C++ function), 30  MultiCFN::getLost (C++ function),	Н	
HIDDEN_VAR_TAG_HVE (C++ member), 3 HIDDEN_VAR_TAG_HVE_PRE (C++ member), 3    MultiCFN::push_back (C++ function), 28 MultiCFN::setNoiseActivation (C++ function), 31 MultiCFN::setNoiseActivation (C++ function), 32 MultiCFN::setNoiseActivation (C++ function), 31 MultiCFN::setNoiseActivation (C++ function), 32 MultiCFN::setNoiseActivation (C++ function), 31 MultiCFN::setNoiseActivation (C++ function), 32 MultiCFN::setNoiseActivation (C++ function), 33 MultiCFN::setNoiseActivation (C++ function), 34 MultiCFN::		
HIDDEN_VAR_TAG_HVE_PRE (C++ member), 3    MultiCFN::print (C++ function), 30     MultiCFN::print (C++ function), 30     MultiCFN::psetNoiseActivation (C++ function), 31     MultiCFN::setNoiseActivation (C++ function), 32     MultiCFN::setNoiseActivation (C++ function), 31     MultiCFN::setNoiseActivation (C++ function), 30     M		
MultiCFN::push_back (C++ function), 28  MultiCFN::push_back (C++ function), 28  MultiCFN::setNoiseActivation (C++ function), 31  MultiCFN::setNoiseLevel (C++ function), 31  MultiCFN::setWeight (C++ function), 28  MultiCFN::setWeight (C++ function), 28  MultiCFN::setWeight (C++ function), 28  MultiCFN::solution (C++ function), 31  MultiCFN::var_index (C++ function), 31  MultiCFN::var_index (C++ member), 32  MAX_BRANCH_SIZE (C++ member), 3  MAX_DOMAIN_SIZE (C++ member), 3  MAX_DOMAIN_SIZE (C++ member), 3  MAX_EAC_ITER (C++ member), 3  MAX_DEATUPLES (C++ member), 3  MAX_NB_TUPLES (C++ member), 3  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN::contention (C++ function), 28  MultiCFN::contention (C++ function), 31  MultiCFN::contention (C++ function), 31  MultiCFN::cost_function (C++ function), 30  MultiCFN::getCost (C++ function), 30  MultiCFN::getNetSeros(C++ function), 29  MultiCFN::getNetseros(C++ function), 30  MultiCFN::getNetseros(C++ function), 29  MultiCFN::getNetseros(C++ function), 30  MultiCFN::getNetseros(C++ function), 29  MultiCFN::getNetseros(C++ function), 29  MultiCFN::getNetseros(C++ function), 30  MultiCFN::getNetseros(C++ function), 48		
IMPLICIT_VAR_TAG (C++ member), 3  L  LARGE_NB_VARS (C++ member), 4  MAX_BCANCH_SIZE (C++ member), 3  MAX_DOMAIN_SIZE (C++ member), 4  MAX_BEAC_ITER (C++ member), 4  MAX_BLIM_BIN (C++ member), 3  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN:: one pute Solution (C++ function), 28  MultiCFN:: one pute Solution (C++ function), 31  MultiCFN:: one pute Solution (C++ function), 30  MultiCFN:: getNetworkName (C++ function), 30  MultiCFN:: getNetworkName (C++ function), 29  MultiCFN:: setNoiseLevel (C++ function), 31  MultiCFN:: setNoiseLevel (C++ function), 31  MultiCFN:: setNoiseAction (C++ function), 31  MultiCFN:	nizzzii_m_mz_nz_nz (e++ memoer), s	- · · · · · · · · · · · · · · · · · · ·
IMPLICIT_VAR_TAG (C++ member), 3  L  LARGE_NB_VARS (C++ member), 4  MAX_BRANCH_SIZE (C++ member), 4  MAX_BRANCH_SIZE (C++ member), 4  MAX_EAC_ITER (C++ member), 4  MAX_BLIM_BIN (C++ member), 4  MAX_BLIM_BIN (C++ member), 4  MAX_DETUPLES (C++ member), 4  MAX_DETUPLES (C++ member), 3  MAX_VAL (C++ member), 3  MILICFN::convertToSolution (C++ function), 28  MultiCFN::convertToSolution (C++ function), 31  MultiCFN::convertToSolution (C++ function), 31  MultiCFN::getPoeimalPoint (C++ function), 30  MultiCFN::getPoeimalPoint (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29		
L LARGE_NB_VARS (C++ member), 4 MultiCFN::setWeight (C++ function), 28 MultiCFN::solution (C++ type), 28 MultiCFN::var (C++ member), 32 MultiCFN::var_index (C++ member), 32 MAX_BRANCH_SIZE (C++ member), 3 MAX_DOMAIN_SIZE (C++ member), 3 MAX_EAC_ITER (C++ member), 4 MAX_ELIM_BIN (C++ member), 4 MAX_ELIM_BIN (C++ member), 4 MAX_ELIM_BIN (C++ member), 4 MAX_VAL (C++ member), 3 MAX_VAL (C++ member), 3 MAX_VAL (C++ member), 3 MIN_VAL (C++ member), 3 MultiCFN::computeSolutionValues (C++ function), 31 MultiCFN::convertToSolution (C++ function), 31 MultiCFN::cost_function_index (C++ member), 32 MultiCFN::getCost (C++ function), 30 MultiCFN::getCost (C++ function), 30 MultiCFN::getDecimalPoint (C++ function), 29 MultiCFN::getNetworkName (C++ function), 20 MultiCFN::getNetworkName (C++ function), 20 MultiCFN::getNetworkName (C++ function), 20 MultiCFN::getNetworkName (C++ function), 20 MultiCFN::get	TMPLICIT VAR TAG $(C++member)$ 3	
Large_NB_Vars (C++ member), 4  MultiCFN::solution (C++ type), 28  MultiCFN::var (C++ member), 32  MultiCFN::var_index (C++ member), 34  Max_Branch_size (C++ member), 3  Max_DOMAIN_Size (C++ member), 3  Max_EAC_ITER (C++ member), 4  Max_ELIM_BIN (C++ member), 4  Max_Branch_size (C++ member), 3  Max_NB_TUPLES (C++ member), 4  Max_ProjectionsMaxDomsize (C++ member), 3  Max_Val (C++ member), 3  MultiCFN::computeSolution (C++ function), 28  MultiCFN::computeSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::cost_function index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getCost (C++ function), 30  MultiCFN::getPecimalPoint (C++ function), 30  MultiCFN::getPecimalPoint (C++ function), 29  MultiCFN::getPecimalPoint (C++ function), 30	III BICIT_VIIIC_IIIG (C++ member), 3	
LARGE_NB_VARS (C++ member), 4  M  MultiCFN::tupleToIndex (C++ function), 31  MultiCFN::var_index (C++ member), 32  MAX_BRANCH_SIZE (C++ member), 3  MAX_DOMAIN_SIZE (C++ member), 3  MAX_ELIM_BIN (C++ member), 4  MAX_BELIM_BIN (C++ member), 4  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::computeSolution (C++ function), 31  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getCost (C++ function), 30  MultiCFN::getCost (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::delta(C++ function), 48  MultiCFN::delta(C++ function), 30  MultiCFN::delta(C++ function), 31  MultiCFN::delta(C++ function), 30  MultiCFN::delta(C++ fun	L	
MAX_ARITY (C++ member), 4  MAX_BRANCH_SIZE (C++ member), 3  MAX_DOMAIN_SIZE (C++ member), 3  MAX_ELITE_RICE (C++ member), 4  MAX_ELIT_BIN (C++ member), 4  MAX_BTANCH_SIZE (C++ member), 3  MAX_ELIT_BIN (C++ member), 4  MAX_ELIT_BIN (C++ member), 4  MAX_PROJECTION2MAXDOMSIZE (C++ member), 3  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN::computeSolution(2  MultiCFN::computeSolution(C++ function), 31  MultiCFN::cost_function(C++ member), 32  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29	LARGE NR VARS $(C++member)$ 4	
MAX_ARITY (C++ member), 4  MAX_BRANCH_SIZE (C++ member), 3  MAX_DOMAIN_SIZE (C++ member), 3  MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 4  MAX_BRANCH_SIZE (C++ member), 3  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::cost_function (C++ function), 31  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getCost (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29	Emide_ne_vince (e     member);	
MAX_BRANCH_SIZE (C++ member), 3  MAX_DOMAIN_SIZE (C++ member), 4  MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::computeSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29	M	
MAX_BRANCH_SIZE (C++ member), 3  MAX_DOMAIN_SIZE (C++ member), 4  MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::computeSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29	$MAX\_ARITY (C++ member), 4$	N.I.
MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN: computeSolutionValues (C++ function), 31  MultiCFN: cost_function_index (C++ member), 32  MultiCFN: getCost (C++ function), 30  MultiCFN: getDecimalPoint (C++ function), 30  MultiCFN: getNetworkName (C++ function), 30  MultiCFN: getNetworkName (C++ function), 29		N
MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::cost_function (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetsetScore (C++ function), 29  MultiCFN::getNetsetScore (C++ function), 29  MultiCFN::getNetsetScore (C++ function), 20		VIDEO CLAVOR (C
MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getSore (C++ function), 20  MultiCFN::getSore (C++ function), 20  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29	$MAX_DOMAIN_SIZE(C++ member), 3$	NARY2CLAUSE (C++ member), 4
MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN:: computeSolutionValues (C++ function), 31  MultiCFN:: convertToSolution (C++ function), 31  MultiCFN:: cost_function (C++ member), 32  MultiCFN:: cost_function_index (C++ member), 32  MultiCFN:: getCost (C++ function), 30  MultiCFN:: getDecimalPoint (C++ function), 30  MultiCFN:: getNetworkName (C++ function), 29  MultiCFN:: getNetworkName (C++ function), 29  MultiCFN:: getSone (C++ function), 29  MultiCFN:: getSone (C++ function), 29  MultiCFN:: getSone (C++ function), 29		
MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getScope (C++ function), 29  MultiCFN::getScope (C++ function), 20  MultiCFN::getScope (C++ function), 20	MAX_EAC_ITER (C++ member), 4 MAX_ELIM_BIN (C++ member), 3	NARYDECONNECTSIZE (C++ member), 3 NARYPROJECTION2MAXDOMSIZE (C++ member), 3
MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getScope (C++ function), 20  MultiCFN::getScope (C++ function), 20	MAX_EAC_ITER (C++ member), 4 MAX_ELIM_BIN (C++ member), 3 MAX_NB_TUPLES (C++ member), 4	NARYDECONNECTSIZE $(C++member)$ , 3 NARYPROJECTION2MAXDOMSIZE $(C++member)$ , 3 NARYPROJECTION3MAXDOMSIZE $(C++member)$ , 3
MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getSore (C++ function), 29  MultiCFN::getSore (C++ function), 29  MultiCFN::getSore (C++ function), 29	MAX_EAC_ITER (C++ member), 4 MAX_ELIM_BIN (C++ member), 3 MAX_NB_TUPLES (C++ member), 4 MAX_VAL (C++ member), 3	NARYDECONNECTSIZE (C++ member), 3 NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTIONPRODDOMSIZE (C++ member), 3
MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getScope (C++ function), 29  MultiCFN::getScope (C++ function), 20  MultiCFN::getScope (C++ function), 20	MAX_EAC_ITER (C++ member), 4 MAX_ELIM_BIN (C++ member), 3 MAX_NB_TUPLES (C++ member), 4 MAX_VAL (C++ member), 3 MIN_VAL (C++ member), 3	NARYDECONNECTSIZE (C++ member), 3 NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTIONPRODDOMSIZE (C++ member), 3
MultiCFN::convertToSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getScope (C++ function), 29  MultiCFN::getScope (C++ function), 29	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28	NARYDECONNECTSIZE (C++ member), 3 NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTIONPRODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3
MultiCFN::convertToSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32  MultiCFN::cost_function_index (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getScope (C++ function), 29  MultiCFN::getScope (C++ function), 29	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28	NARYDECONNECTSIZE (C++ member), 3 NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTIONPRODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3
MultiCFN::cost_function (C++ member), 32 MultiCFN::getCost (C++ function), 30 MultiCFN::getDecimalPoint (C++ function), 30 MultiCFN::getNetworkName (C++ function), 29 MultiCFN::getScope (C++ function), 29 MultiCFN::getScope (C++ function), 29 MultiCFN::getScope (C++ function), 29 MultiCFN::getScope (C++ function), 29	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function),	NARYDECONNECTSIZE (C++ member), 3 NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTIONPRODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3
MultiCFN::cost_function_index(C++ member), 32 MultiCFN::getCost(C++ function), 30 MultiCFN::getDecimalPoint(C++ function), 30 MultiCFN::getNetworkName(C++ function), 29 MultiCFN::getScore(C++ function), 29 MultiCFN::getScore(C++ function), 29	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31	NARYDECONNECTSIZE (C++ member), 3 NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTIONPRODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3  O  OPEN_NODE_LIMIT (C++ member), 3
MultiCFN::getDecimalPoint (C++ function), 30 MultiCFN::getNetworkName (C++ function), 29 MultiCFN::getScope (C++ function), 29 MultiCFN::getScope (C++ function), 29	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 31	NARYDECONNECTSIZE (C++ member), 3 NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTIONPRODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3  O OPEN_NODE_LIMIT (C++ member), 3  S
MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29  tb2init (C++ function), 48	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 31  MultiCFN::cost_function (C++ member), 32	NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTIONPRODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3  O  OPEN_NODE_LIMIT (C++ member), 3  S STORE_SIZE (C++ member), 3
MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getNetworkName (C++ function), 29  tb2init (C++ function), 48	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 32  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30	NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTIONPRODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3  O  OPEN_NODE_LIMIT (C++ member), 3  S STORE_SIZE (C++ member), 3
	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 32  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30	NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTION9RODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3  O  OPEN_NODE_LIMIT (C++ member), 3  S  STORE_SIZE (C++ member), 3  T
Mallai CENT and Calantina (Calantina (Calant	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 32  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29	NARYDECONNECTSIZE (C++ member), 3  NARYPROJECTION2MAXDOMSIZE (C++ member), 3  NARYPROJECTION3MAXDOMSIZE (C++ member), 3  NARYPROJECTIONPRODDOMSIZE (C++ member), 3  NARYPROJECTIONSIZE (C++ member), 3  O  OPEN_NODE_LIMIT (C++ member), 3  S  STORE_SIZE (C++ member), 3  T  tb2checkOptions (C++ function), 48
	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 32  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getScope (C++ function), 29	NARYDECONNECTSIZE (C++ member), 3  NARYPROJECTION2MAXDOMSIZE (C++ member), 3  NARYPROJECTION3MAXDOMSIZE (C++ member), 3  NARYPROJECTIONPRODDOMSIZE (C++ member), 3  NARYPROJECTIONSIZE (C++ member), 3  O  OPEN_NODE_LIMIT (C++ member), 3  S  STORE_SIZE (C++ member), 3  T  tb2checkOptions (C++ function), 48
	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 32  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getScope (C++ function), 29  MultiCFN::getScope (C++ function), 31	NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTION9RODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3  O OPEN_NODE_LIMIT (C++ member), 3  S STORE_SIZE (C++ member), 3  T tb2checkOptions (C++ function), 48 tb2init (C++ function), 48 ToulBar2 (C++ class), 33 ToulBar2::addAMOConstraints (C++ member), 41
Mail $\pm 3$ ( ENL $\pm 0.00\pm 1.37900$ ( $\ell$ ) 1 $\pm turn ottom)$ (21)	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::convertToSolution (C++ function), 32  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getCost (C++ function), 30  MultiCFN::getNetworkName (C++ function), 29  MultiCFN::getScope (C++ function), 29  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolutionValues (C++ function), 31  MultiCFN::getSolutionValues (C++ function), 31	NARYDECONNECTSIZE (C++ member), 3  NARYPROJECTION2MAXDOMSIZE (C++ member), 3  NARYPROJECTION3MAXDOMSIZE (C++ member), 3  NARYPROJECTIONPRODDOMSIZE (C++ member), 3  NARYPROJECTIONSIZE (C++ member), 3  O  OPEN_NODE_LIMIT (C++ member), 3  S  STORE_SIZE (C++ member), 3  T  tb2checkOptions (C++ function), 48  tb2init (C++ function), 48  ToulBar2 (C++ class), 33  ToulBar2::addAMOConstraints (C++ member), 41  ToulBar2::addAMOConstraints_(C++ member), 41
MultiCFN::getType (C++ function), 30  MultiCFN::getType (C++ function), 30  ToulBar2::allelefreqdistrib (C++ member), 39	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::cost_function (C++ function), 32  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getSolution (C++ function), 29  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolutionValues (C++ function), 31  MultiCFN::getType (C++ function), 30	NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTION9RODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3  O OPEN_NODE_LIMIT (C++ member), 3  S STORE_SIZE (C++ member), 3  T tb2checkOptions (C++ function), 48 tb2init (C++ function), 48 ToulBar2 (C++ class), 33 ToulBar2::addAMOConstraints (C++ member), 41 ToulBar2::allelefreqdistrib (C++ member), 39
MultiCFN::getType(C++ function), 30 ToulBar2::allelefreqdistrib(C++ member) 39	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::cost_function (C++ function), 32  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getSolution (C++ function), 29  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolutionValues (C++ function), 31  MultiCFN::getType (C++ function), 30	NARYDECONNECTSIZE (C++ member), 3  NARYPROJECTION2MAXDOMSIZE (C++ member), 3  NARYPROJECTION3MAXDOMSIZE (C++ member), 3  NARYPROJECTIONPRODDOMSIZE (C++ member), 3  NARYPROJECTIONSIZE (C++ member), 3  O  OPEN_NODE_LIMIT (C++ member), 3  S  STORE_SIZE (C++ member), 3  T  tb2checkOptions (C++ function), 48  tb2init (C++ function), 48  ToulBar2 (C++ class), 33  ToulBar2::addAMOConstraints (C++ member), 41  ToulBar2::addAMOConstraints_(C++ member), 41
Mail $\pm 3$ ( ENL $\pm 0.00\pm 1.37900$ ( $\ell$ ) 1 $\pm turn ottom)$ (21)	MAX_EAC_ITER (C++ member), 4  MAX_ELIM_BIN (C++ member), 3  MAX_NB_TUPLES (C++ member), 4  MAX_VAL (C++ member), 3  MIN_VAL (C++ member), 3  MultiCFN (C++ class), 28  MultiCFN::~MultiCFN (C++ function), 28  MultiCFN::computeSolutionValues (C++ function), 31  MultiCFN::cost_function (C++ function), 32  MultiCFN::cost_function (C++ member), 32  MultiCFN::getCost (C++ function), 30  MultiCFN::getDecimalPoint (C++ function), 30  MultiCFN::getSolution (C++ function), 29  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolution (C++ function), 31  MultiCFN::getSolutionValues (C++ function), 31  MultiCFN::getType (C++ function), 30	NARYPROJECTION2MAXDOMSIZE (C++ member), 3 NARYPROJECTION3MAXDOMSIZE (C++ member), 3 NARYPROJECTION9RODDOMSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3 NARYPROJECTIONSIZE (C++ member), 3  O OPEN_NODE_LIMIT (C++ member), 3  S STORE_SIZE (C++ member), 3  T tb2checkOptions (C++ function), 48 tb2init (C++ function), 48 ToulBar2 (C++ class), 33 ToulBar2::addAMOConstraints (C++ member), 41 ToulBar2::allelefreqdistrib (C++ member), 39

```
ToulBar2::approximateCountingBTD (C++ mem-
                                                ToulBar2::eps (C++ member), 46
                                                ToulBar2::epsFilename (C++ member), 46
        ber), 35
                                                ToulBar2::epsilon (C++ member), 42
ToulBar2::backtrackLimit (C++ member), 37
ToulBar2::bayesian (C++ member), 38
                                                ToulBar2::errorg (C++ member), 38
ToulBar2::bep (C++member), 40
                                                ToulBar2::evidence_file (C++ member), 42
ToulBar2::Berge_Dec (C++ member), 43
                                                ToulBar2::externalUB(C++ member), 34
ToulBar2::bilevel (C++ member), 46
                                                ToulBar2::foundersprob_class(C++ member), 38
ToulBar2::binaryBranching (C++ member), 35
                                                ToulBar2::FullEAC (C++ member), 33
ToulBar2::bisupport (C++ member), 35
                                                ToulBar2::generation (C++member), 39
ToulBar2::boostingBTD (C++ member), 43
                                                ToulBar2::getCostMultiplier (C++ function), 33
ToulBar2::btdMode (C++ member), 41
                                                ToulBar2::getCostMultiplierInt (C++ function),
ToulBar2::btdRootCluster (C++ member), 42
                                                        33
ToulBar2::btdSubTree (C++ member), 42
                                                ToulBar2::gz (C++ member), 38
ToulBar2::bz2 (C++ member), 38
                                                ToulBar2::haplotype (C++ member), 38
ToulBar2::cardinality (C++ member), 41
                                                ToulBar2::hbfs (C++ member), 45
ToulBar2::cfn (C++ member), 38
                                                ToulBar2::hbfsAlpha(C++ member), 45
ToulBar2::clusterFile (C++ member), 44
                                                ToulBar2::hbfsBeta(C++ member), 46
                                                ToulBar2::hbfsCPLimit (C++ member), 46
ToulBar2::consecutiveAllele (C++ member), 39
ToulBar2::constrOrdering (C++ member), 37
                                                ToulBar2::hbfsGlobalLimit (C++ member), 45
ToulBar2::costfuncSeparate (C++ member), 36
                                                ToulBar2::hbfsOpenNodeLimit (C++ member), 46
ToulBar2::costMultiplier (C++ member), 40
                                                ToulBar2::heuristicFreedom (C++ member), 43
ToulBar2::costMultiplier_(C++ member), 40
                                                ToulBar2::heuristicFreedomLimit(C++ member),
ToulBar2::costMultiplierBLP(C++ member), 46
                                                        43
ToulBar2::costThreshold(C++ member), 39
                                                ToulBar2::hve (C++member), 36
ToulBar2::costThresholdPre(C++ member), 39
                                                ToulBar2::incop_cmd (C++ member), 44
ToulBar2::costThresholdPreS(C++ member), 39
                                                ToulBar2::initialLbBLP (C++ member), 46
ToulBar2::costThresholdS(C++ member), 39
                                                ToulBar2::initialUbBLP(C++ member), 46
ToulBar2::debug (C++ member), 34
                                                ToulBar2::interrupted (C++ member), 43
ToulBar2::decimalPoint (C++ member), 40
                                                ToulBar2::isZ (C++ member), 42
ToulBar2::decimalPointBLP (C++ member), 46
                                                ToulBar2::knapsackDP (C++ member), 41
ToulBar2::DEE (C++ member), 37
                                                ToulBar2::lastConflict (C++ member), 36
ToulBar2::DEE_(C++member), 37
                                                ToulBar2::LcLevel (C++ member), 40
ToulBar2::deltaUb (C++ member), 40
                                                ToulBar2::LcLevel_prev (C++ member), 40
ToulBar2::deltaUbAbsolute(C++ member), 40
                                                ToulBar2::lds (C++member), 37
ToulBar2::deltaUbRelativeGap (C++ member), 40
                                                ToulBar2::learning (C++ member), 43
ToulBar2::deltaUbS(C++ member), 40
                                                ToulBar2::limited (C++ member), 37
ToulBar2::dichotomicBranching (C++member), 35
                                                ToulBar2::logepsilon (C++ member), 42
ToulBar2::dichotomicBranchingSize (C++ mem-
                                                ToulBar2::logU (C++ member), 42
        ber), 35
                                                ToulBar2::logZ(C++ member), 42
ToulBar2::divBound(C++ member), 41
                                                ToulBar2::lp (C++ member), 41
ToulBar2::divMethod(C++ member), 41
                                                ToulBar2::lrBCD_cmd (C++ member), 44
ToulBar2::divNbSol (C++ member), 41
                                                ToulBar2::map_file (C++ member), 38
ToulBar2::divRelax (C++ member), 41
                                                ToulBar2::markov_log(C++ member), 42
ToulBar2::divWidth(C++ member), 41
                                                ToulBar2::maxEACIter (C++ member), 40
ToulBar2::dumpOriginalAfterPreprocessing
                                                ToulBar2::maxsateval (C++ member), 42
        (C++ member), 35
                                                ToulBar2::maxSeparatorSize (C++ member), 43
ToulBar2::dumpWCSP (C++ member), 35
                                                ToulBar2::minProperVarSize (C++ member), 43
ToulBar2::elimDegree (C++ member), 35
                                                ToulBar2::minsumDiffusion (C++ member), 36
ToulBar2::elimDegree_(C++ member), 36
                                                ToulBar2::MSTDAC (C++ member), 37
                                                ToulBar2::nbDecisionVars (C++ member), 37
ToulBar2::elimDegree_preprocessing (C++ mem-
        ber), 36
                                                ToulBar2::nbTimesIsVAC (C++ member), 33
ToulBar2::elimDegree_preprocessing_
                                                ToulBar2::nbTimesIsVACitThresholdMoreThanOne
                                         (C++
       member), 36
                                                        (C++ member), 33
ToulBar2::elimSpaceMaxMB (C++ member), 36
                                                ToulBar2::negCostBLP (C++ member), 46
```

```
ToulBar2::newsolution (C++member), 38
                                                ToulBar2::solutionBasedPhaseSaving (C++ mem-
ToulBar2::NormFactor (C++ member), 38
                                                        ber), 35
                                                ToulBar2::solutionFile (C++ member), 34
ToulBar2::opb (C++ member), 40
ToulBar2::parallel(C++ member), 45
                                                ToulBar2::solutionFileRewindPos (C++ member),
ToulBar2::pedigree (C++ member), 38
ToulBar2::pedigreeCorrectionMode (C++ mem-
                                                ToulBar2::sortBFS (C++ member), 46
       ber), 39
                                                ToulBar2::sortDomains (C++ member), 35
ToulBar2::pedigreePenalty (C++ member), 39
                                                ToulBar2::sortedDomains (C++ member), 35
ToulBar2::pils_cmd (C++ member), 44
                                                ToulBar2::splitClusterMaxSize(C++ member), 43
                                                ToulBar2::startCpuTime (C++ member), 43
ToulBar2::preprocessFunctional (C++ member),
                                                ToulBar2::startRealTime (C++ member), 43
ToulBar2::preprocessNary (C++ member), 36
                                                ToulBar2::startRealTimeAfterPreProcessing
ToulBar2::preprocessTernaryRPC (C++ member),
                                                        (C++ member), 43
                                                ToulBar2::Static_variable_ordering (C++ mem-
ToulBar2::problemsaved_filename (C++ member),
                                                        ber), 36
        42
                                                ToulBar2::stdin_format (C++ member), 43
ToulBar2::pwc (C++ member), 36
                                                ToulBar2::timeOut (C++ member), 43
ToulBar2::pwcMinimalDualGraph(C++ member), 36
                                                ToulBar2::trwsAccuracy (C++ member), 39
ToulBar2::qpbo (C++ member), 40
                                                ToulBar2::trwsNIter (C++ member), 39
ToulBar2::gpboQuadraticCoefMultiplier
                                                ToulBar2::trwsNIterComputeUb (C++ member), 39
       member), 40
                                                ToulBar2::trwsNIterNoChange (C++ member), 39
ToulBar2::QueueComplexity (C++ member), 36
                                                ToulBar2::trwsOrder (C++ member), 39
ToulBar2::RASPS (C++ member), 33
                                                ToulBar2::uai (C++ member), 38
ToulBar2::RASPSangle (C++ member), 34
                                                ToulBar2::uaieval (C++ member), 43
ToulBar2::RASPSitThresholds (C++ member), 34
                                                ToulBar2::useRASPS (C++ member), 33
ToulBar2::RASPSlastitThreshold (C++ member),
                                                ToulBar2::vac (C++member), 39
                                                ToulBar2::vac_prev (C++ member), 39
ToulBar2::RASPSnbBacktracks (C++ member), 34
                                                ToulBar2::VAClin (C++ member), 41
ToulBar2::RASPSnbStrictACVariables (C++ mem-
                                                ToulBar2:: VACthreshold (C++ member), 33
                                                ToulBar2::vacValueHeuristic (C++ member), 40
        ber), 34
ToulBar2::RASPSreset (C++ member), 34
                                                ToulBar2::varOrder (C++ member), 41
ToulBar2::RASPSsaveitThresholds (C++ member),
                                                ToulBar2::verbose (C++ member), 33
                                                ToulBar2::verifiedOptimum(C++ member), 46
ToulBar2::reduceHeight (C++ member), 42
                                                ToulBar2::verifyOpt (C++ member), 46
                                                ToulBar2::version (C++ member), 33
ToulBar2::removevalue (C++ member), 38
ToulBar2::resolution (C++ member), 38
                                                ToulBar2::vnsInitSol (C++ member), 44
ToulBar2::resolution_Update(C++ member), 38
                                                ToulBar2::vnsKcur (C++ member), 45
ToulBar2::restart (C++ member), 37
                                                ToulBar2::vnsKinc (C++ member), 45
ToulBar2::rootHeuristic(C++ member), 42
                                                ToulBar2::vnsKmax(C++member), 45
ToulBar2::searchMethod(C++ member), 44
                                                ToulBar2::vnsKmin(C++ member), 44
ToulBar2::seed (C++ member), 44
                                                ToulBar2::vnsLDScur (C++ member), 45
ToulBar2::setCostMultiplier (C++ function), 33
                                                ToulBar2::vnsLDSinc (C++ member), 44
ToulBar2::setmax (C++ member), 38
                                                ToulBar2::vnsLDSmax (C++ member), 44
ToulBar2::setmin (C++ member), 37
                                                ToulBar2::vnsLDSmin(C++ member), 44
ToulBar2::setminobj (C++ member), 38
                                                ToulBar2::vnsNeighborChange (C++ member), 45
ToulBar2::setvalue (C++ member), 37
                                                ToulBar2::vnsNeighborSizeSync(C++ member), 45
                                                ToulBar2::vnsNeighborVarHeur (C++ member), 45
ToulBar2::showHidden (C++ member), 34
ToulBar2::showSolutions (C++ member), 34
                                                ToulBar2::vnsOptimum (C++ member), 45
ToulBar2::sigma (C++ member), 44
                                                ToulBar2::vnsOptimumS (C++ member), 45
ToulBar2::singletonConsistency (C++ member),
                                                ToulBar2::vnsOutput (C++ member), 44
                                                ToulBar2::vnsParallelLimit (C++ member), 45
ToulBar2::solution_uai_file (C++ member), 42
                                                ToulBar2::vnsParallelSync (C++ member), 45
ToulBar2::solution_uai_filename (C++ member),
                                                ToulBar2::wcnf (C++ member), 40
                                                ToulBar2::weightedDegree (C++ member), 37
        42
```

<pre>ToulBar2::weightedTightness (C++ member), 37</pre>	WeightedCSP::getLb $(C++function)$ , 5
ToulBar2::writeSolution $(C++ member)$ , 34	<pre>WeightedCSP::getListSuccessors (C++ function),</pre>
ToulBar2::xmlcop $(C++ member)$ , 42	19
ToulBar2::xmlflag $(C++ member)$ , 42	${\tt WeightedCSP::getMaxCurrentDomainSize} \qquad (C++$
ToulBar2::xz $(C++ member)$ , 38	function), 10
14/	WeightedCSP::getMaxDomainSize( $C++function$ ), 10
W	<pre>WeightedCSP::getMaxUnaryCost(C++ function), 9</pre>
WeightedCSP $(C++ class)$ , 5	$WeightedCSP::getMaxUnaryCostValue\ (C++\ func-$
<pre>WeightedCSP::~WeightedCSP(C++ function), 5</pre>	tion), 9
<pre>WeightedCSP::addValueName (C++ function), 11</pre>	WeightedCSP::getName $(C++function)$ , 5, 7
WeightedCSP::assign ( $C++$ function), 8	WeightedCSP::getNbDEE ( $C++$ function), 11
<pre>WeightedCSP::assigned (C++ function), 8</pre>	<pre>WeightedCSP::getNegativeLb (C++ function), 6</pre>
<pre>WeightedCSP::assignLS (C++ function), 8</pre>	WeightedCSP::getSolution ( $C++$ function), 20
WeightedCSP::buildTreeDecomposition $(C++$	WeightedCSP::getSolutionCost $(C++function)$ , 20
function), 21	WeightedCSP::getSolutions $(C++function)$ , 20
WeightedCSP::canbe $(C++function)$ , 8	${\tt WeightedCSP::getSolutionValue} \ (C++function), 20$
WeightedCSP::cannotbe $(C++function)$ , 8	WeightedCSP::getSolver $(C++function)$ , 5
<pre>WeightedCSP::cartProd (C++ function), 10</pre>	WeightedCSP::getSup $(C++function)$ , 7
<pre>WeightedCSP::Cost2ADCost(C++ function), 21</pre>	WeightedCSP::getSupport $(C++function)$ , 9
<pre>WeightedCSP::Cost2LogProb(C++ function), 21</pre>	WeightedCSP::getTreeDec $(C++function)$ , 21
<pre>WeightedCSP::Cost2Prob(C++ function), 21</pre>	WeightedCSP::getTrueDegree $(C++function)$ , 9
<pre>WeightedCSP::Cost2RDCost(C++ function), 21</pre>	WeightedCSP::getUb $(C++function)$ , 6
<pre>WeightedCSP::deactivatePropagate (C++ func-</pre>	WeightedCSP::getUnaryCost $(C++function)$ , 8
tion), 10	WeightedCSP::getValue $(C++function)$ , 7
<pre>WeightedCSP::decimalToCost (C++ function), 21</pre>	WeightedCSP::getValueName ( $C++$ function), 11
<pre>WeightedCSP::deconnect (C++ function), 8</pre>	<pre>WeightedCSP::getVarIndex (C++ function), 7</pre>
<pre>WeightedCSP::decrease (C++ function), 8</pre>	${\tt WeightedCSP::getWeightedDegree} \ (\textit{C++function}), 9$
<pre>WeightedCSP::decreaseLb (C++ function), 6</pre>	WeightedCSP::increase ( $C++$ function), $8$
<pre>WeightedCSP::DoubletoCost (C++ function), 21</pre>	WeightedCSP::increaseLb $(C++function)$ , 6
WeightedCSP::dump $(C++function)$ , 21	WeightedCSP::iniSingleton $(C++function)$ , 22
<pre>WeightedCSP::dump_CFN (C++ function), 21</pre>	<pre>WeightedCSP::initDivVariables(C++function), 22</pre>
WeightedCSP::enforceUb(C++ function), 6	${\tt WeightedCSP::initSolutionCost} \ (\textit{C++function}), 20$
<pre>WeightedCSP::enumerated (C++ function), 7</pre>	WeightedCSP::isactivatePropagate ( $C++$ func-
<pre>WeightedCSP::finiteUb (C++ function), 6</pre>	tion), 10
<pre>WeightedCSP::getBergeDecElimOrder (C++ func-</pre>	WeightedCSP::isfinite $(C++function)$ , 7
tion), 19	WeightedCSP::isGlobal $(C++function)$ , 19
<pre>WeightedCSP::getBestValue(C++ function), 9</pre>	WeightedCSP::isKnapsack ( $C++$ function), 19
<pre>WeightedCSP::getDACOrder (C++ function), 8</pre>	<pre>WeightedCSP::LogProb2Cost (C++ function), 21</pre>
<pre>WeightedCSP::getDDualBound (C++ function), 6</pre>	WeightedCSP::LogSumExp $(C++function)$ , 21
<pre>WeightedCSP::getDegree (C++ function), 9</pre>	WeightedCSP::makeEnumeratedVariable $(C++$
<pre>WeightedCSP::getDivVariables (C++ function), 22</pre>	function), 11
<pre>WeightedCSP::getDLb (C++ function), 6</pre>	<pre>WeightedCSP::makeIntervalVariable (C++ func-</pre>
<pre>WeightedCSP::getDomainInitSize(C++function),7</pre>	<i>tion</i> ), 11
<pre>WeightedCSP::getDomainSize(C++ function), 7</pre>	<pre>WeightedCSP::makeWeightedCSP(C++ function), 22</pre>
<pre>WeightedCSP::getDomainSizeSum(C++function), 10</pre>	WeightedCSP::medianArity ( $C++$ function), 10
<pre>WeightedCSP::getDPrimalBound (C++ function), 6</pre>	WeightedCSP::medianDegree ( $C++$ function), 10
<pre>WeightedCSP::getDUb (C++ function), 6</pre>	${\tt WeightedCSP::medianDomainSize} \ (C++function), 10$
<pre>WeightedCSP::getEnumDomain (C++ function), 7</pre>	WeightedCSP::nextValue $(C++function)$ , 8
<pre>WeightedCSP::getEnumDomainAndCost (C++ func-</pre>	WeightedCSP::numberOfConnectedBinaryConstraints
tion), 7	( <i>C</i> ++ <i>function</i> ), 10
<pre>WeightedCSP::getIndex (C++ function), 5</pre>	<pre>WeightedCSP::numberOfConnectedConstraints</pre>
WeightedCSP::getInf $(C++function)$ , 7	(C++function), 10
WeightedCSP::getIsPartOfOptimalSolution	${\tt WeightedCSP::} number Of Connected Knapsack Constraints$
(C++ function), 9	(C++function), 10

WeightedCSP::numberOfConstraints ( $C++$ func-	WeightedCSP::postWSameGcc ( $C++$ function), 17
tion), 10	WeightedCSP::postWSum(C++ function), 18
WeightedCSP::numberOfUnassignedVariables	<pre>WeightedCSP::postWVarAmong (C++ function), 14</pre>
(C++ function), 10	<pre>WeightedCSP::postWVarSum(C++ function), 18</pre>
<pre>WeightedCSP::numberOfVariables (C++ function),</pre>	<pre>WeightedCSP::preprocessing (C++ function), 9</pre>
10	<pre>WeightedCSP::print (C++ function), 21</pre>
WeightedCSP::postAllDifferentConstraint	<pre>WeightedCSP::printSolution(C++ function), 20, 21</pre>
(C++ function), 12	<pre>WeightedCSP::printVACStat (C++ function), 22</pre>
<pre>WeightedCSP::postBinaryConstraint (C++ func-</pre>	WeightedCSP::Prob2Cost (C++ function), 21
tion), 11	WeightedCSP::propagate (C++ function), 10
<pre>WeightedCSP::postCliqueConstraint (C++ func-</pre>	<pre>WeightedCSP::reactivatePropagate (C++ func-</pre>
tion), 12	tion), 10
<pre>WeightedCSP::postDisjunction(C++ function), 12</pre>	<pre>WeightedCSP::read_legacy (C++ function), 19</pre>
<pre>WeightedCSP::postGlobalConstraint (C++ func-</pre>	<pre>WeightedCSP::read_lp (C++ function), 20</pre>
tion), 14	<pre>WeightedCSP::read_opb (C++ function), 20</pre>
<pre>WeightedCSP::postGlobalFunction (C++ function),</pre>	WeightedCSP::read_qpbo (C++ function), 20
14	WeightedCSP::read_random(C++ function), 20
WeightedCSP::postIncrementalBinaryConstraint	WeightedCSP::read_uai2008 (C++ function), 19
(C++ function), 11	WeightedCSP::read_wcnf (C++ function), 20
WeightedCSP::postIncrementalTernaryConstraint	
(C++ function), 11	WeightedCSP::remove (C++ function), 8
WeightedCSP::postIncrementalUnaryConstraint	WeightedCSP::removeSingleton (C++ function), 22
(C++ function), 11	WeightedCSP::resetTightness (C++ function), 9
<pre>WeightedCSP::postKnapsackConstraint (C++)</pre>	WeightedCSP::resetTightnessAndWeightedDegree
function), 12, 13	(C++ function), 9
<pre>WeightedCSP::postMaxWeight (C++ function), 18</pre>	WeightedCSP::resetWeightedDegree (C++ func-
WeightedCSP::postMST (C++ function), 17	tion), 9
WeightedCSP::postNaryConstraintBegin $(C++)$	WeightedCSP::restoreSolution(C++ function), 21
function), 12	WeightedCSP::setBestValue (C++ function), 9
<pre>WeightedCSP::postNaryConstraintEnd (C++ func-</pre>	WeightedCSP::setDACOrder (C++ function), 19
tion), 12	WeightedCSP::setInfiniteCost (C++ function), 6
<pre>WeightedCSP::postNaryConstraintTuple (C++)</pre>	WeightedCSP::setIsPartOfOptimalSolution
function), 12	(C++function), 9
WeightedCSP::postNullaryConstraint (C++ func-	WeightedCSP::setLb (C++ function), 21
tion), 11	WeightedCSP::setName (C++ function), 5
WeightedCSP::postSpecialDisjunction $(C++)$	WeightedCSP::setSolution (C++ function), 20
function), 12	WeightedCSP::setUb (C++ function), 21
WeightedCSP::postSupxyc(C++ function), 12	WeightedCSP::sortConstraints (C++ function), 9
WeightedCSP::postTernaryConstraint (C++ func-	WeightedCSP::toIndex (C++ function), 8
tion), 11	WeightedCSP::toValue (C++ function), 8
WeightedCSP::postUnary(C++ function), 11, 12	WeightedCSP::unassigned(C++function), 8
WeightedCSP::postUnaryConstraint (C++ func-	WeightedCSP::updateDUb (C++ function), 6
	WeightedCSP::updateSingleton (C++ function), 22
<pre>tion), 11, 12 WeightedCSP::postWAllDiff(C++ function), 15, 16</pre>	WeightedCSP::updateUb (C++ function), 6
WeightedCSP::postWAmong (C++ function), 14	
WeightedCSP::postWDivConstraint(C++ function),	WeightedCSP::verify $(C++function)$ , 10 WeightedCSP::whenContradiction $(C++function)$ , 9
19	- · · · · · · · · · · · · · · · · · · ·
	WeightedCSPSolver (C++ class), 23 WeightedCSPSolver::~WeightedCSPSolver (C++
WeightedCSP::postWeightedCSPConstraint ( $C++$	
function), 13	function), 23
WeightedCSP::postWGcc(C++function), 16	WeightedCSPSolver::assign(C++ function), 23 WeightedCSPSolver::beginSolve(C++ function), 24
<pre>WeightedCSP::postWGrammarCNF (C++ function), 17 WeightedCSP::postWOverlap (C++ function), 18</pre>	WeightedCSPSolver::beginSolve(C++function), 24
	WeightedCSPSolver::decrease (C++ function), 23
WeightedCSP::postWRegular(C++ function), 15	WeightedCSPSolver::dump_wcsp(C++ function), 26 WeightedCSPSolver::ondSolve(C++ function), 24
WeightedCSP::postWSame ( $C++$ function), 16, 17	WeightedCSPSolver::endSolve(C++ function), 24

```
WeightedCSPSolver::getDDualBound (C++ func-
        tion), 27
WeightedCSPSolver::getNbBacktracks (C++ func-
        tion), 23
WeightedCSPSolver::getNbNodes(C++function), 23
WeightedCSPSolver::getSolution (C++ function),
WeightedCSPSolver::getSolutionCost (C++ func-
        tion), 26
WeightedCSPSolver::getSolutions (C++ function),
WeightedCSPSolver::getSolutionValue
                                          (C++
        function), 26
WeightedCSPSolver::getUnassignedVars
                                          (C++
        function), 27
WeightedCSPSolver::getWCSP(C++ function), 23
WeightedCSPSolver::hybridSolve (C++ function),
WeightedCSPSolver::increase (C++ function), 23
WeightedCSPSolver::lrBCD (C++ function), 25
WeightedCSPSolver::makeWeightedCSPSolver
        (C++ function), 27
WeightedCSPSolver::narycsp(C++ function), 24
WeightedCSPSolver::numberOfUnassignedVariables
        (C++ function), 27
WeightedCSPSolver::parse_solution (C++ func-
        tion), 26
WeightedCSPSolver::pils (C++ function), 25
WeightedCSPSolver::preprocessing (C++ func-
WeightedCSPSolver::read_random (C++ function),
WeightedCSPSolver::read_solution (C++ func-
        tion), 26
WeightedCSPSolver::read_wcsp(C++ function), 23
WeightedCSPSolver::recursiveSolve (C++ func-
        tion), 24
WeightedCSPSolver::recursiveSolveLDS
                                          (C++
        function), 24
WeightedCSPSolver::remove (C++ function), 23
WeightedCSPSolver::solve (C++ function), 24
WeightedCSPSolver::solve_symmax2sat
                                          (C++
        function), 25
WRONG_VAL (C++member), 3
```