Technical specifications of the RandomCsp library — version 1.8.0

J.I. van Hemert
jvhemert@cs.leidenuniv.nl

February 16, 2004

Contents

1 Type definitions for CspC

General type definitons

Names

These types can and should be used as a type for variables in your own code.

1.1 typedef unsigned int **DomainSizeT**

Size of the domain of a variable

Size of the domain of a variable

1.2 typedef unsigned int **NumberOfVariablesT**

Number of variables in the CSP

Number of variables in the CSP

1.3

typedef unsigned int **VariableT**

Value of a variable from the domain of the variable

Value of a variable from the domain of the variable

1.4

typedef unsigned int **ValueT**

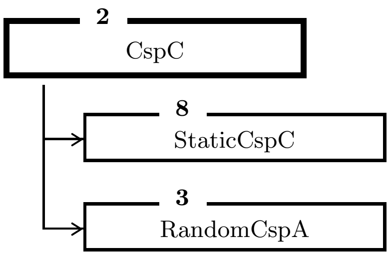
Value of a variable from the domain of the variable

Value of a variable from the domain of the variable

2 class **CspC**

Constraint Satisfaction Problem

Inheritance



Public Members

2.1	struct	**ConstraintS**	*The storage for a constraint between two variables*	7
2.2	typedef ConstraintS *	**ConstraintT**	*The type for a constraint between two variables.*	8
2.3	static const int	**MajorVersion**		8
2.4	static const int	**MinorVersion**		8
2.5	static const int	**UpdateVersion**		9
2.6	static const int	**DebuggingCode**		9
2.7	void	**ResetConflictChecksCounter** ()		9
2.8	unsigned int	**GetNumberOfConflictChecks** ()		9
2.9	unsigned long	**GetNumberOfConflictPairs** ()		9
2.10		**CspC** (NumberOfVariablesT const iNumberOfVariables)	*Constructor*	9
2.11		**CspC** (const CspC & iCsp)	*Copy constructor*	10
2.12	CspC const&	**operator=** (const CspC & iCsp)	*Operator =*	10
2.13	virtual	**~CspC** ()	*Destructor*	10
2.14	NumberOfVariablesT const			

Protected Members

Private Members

A base class for implementing Binary ConstrainS Satisfaction Problems.

2.1

struct **ConstraintS**

The storage for a constraint between two variables

Members

The storage for a constraint between two variables

2.1.1

ConflictT** **Conflicts**

2.1.2

ConstraintS* **Previous**

2.1.3

ConstraintS* **Next**

2.2

typedef ConstraintS * **ConstraintT**

The type for a constraint between two variables.

The type for a constraint between two variables.

2.3

static const int **MajorVersion**

2.4

static const int **MinorVersion**

2.5

static const int **UpdateVersion**

2.6

static const int **DebuggingCode**

2.7

void **ResetConflictChecksCounter** ()

2.8

unsigned int **GetNumberOfConflictChecks** ()

2.9

unsigned long **GetNumberOfConflictPairs** ()

2.10

CspC (NumberOfVariablesT const iNumberOfVariables)

Constructor

Constructor, allocates memory for all the objects except for the conflicts. The conflicts will be allocated as needed when constraints are added to the CSP.

Return Value: `nothing.`
Parameters: `iNumberOfVariables` number of variables used in the CSP.
Author: J.I. van Hemert
Version: 1.0

2.11

CspC (const CspC & iCsp)

Copy constructor

Copy constructor for CspC. Takes an existing CSP as input and copies it to the new CSP, including all constraints and conflicts.

Return Value: `nothing.`
Parameters: `iCsp` existing object of CspC class.
Author: J.I. van Hemert
Version: 1.0

2.12

CspC const& **operator=** (const CspC & iCsp)

Operator =

Destroys the left CSP and overwrites it with the right CSP. The left CSP will receive the constraints and conflicts of the right CSP.

Return Value: `new` CSP, copied from iCsp.
Parameters: `iCsp` existing CSP.
Author: J.I. van Hemert
Version: 1.0

2.13

virtual **~CspC** ()

Destructor

Destructor deallocates all memory used by the object.

Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.0

2.14

NumberOfVariablesT const **GetNumberOfVariables** () const

Getting the number of variables

Get the number of variables present in the CSP.

Return Value: `number` of variables in CSP.
Author: J.I. van Hemert
Version: 1.0

2.15

DomainSizeT const **GetDomainSize** (VariableT const iVariable) const

Getting the domainsize

Get the size of the domain of a variable.

Return Value: `Size` of the domain of the given variable.
Parameters: `iVariable` A number corresponding to a variable.
Author: J.I. van Hemert
Version: 1.0

2.16

void const **SetDomainSize** (VariableT const iVariable, DomainSizeT const iDomainSize)

Setting the domainsize

Set the size of the domain of a variable.

Return Value: `nothing.`
Parameters: `iVariable` A number corresponding to a variabe.
`iDomainSize` size of the domain of iVariable.
Author: J.I. van Hemert
Version: 1.0

2.17

bool const **IsValid** (ValueT const iSolution []) const

Checking for valid solutions

A way of checking if solutions for the CSP are valid. The input array with the size of the number of variables in the problem is checked as a solution for the CSP. If there are no conflicts true is returned.

Return Value: `true` if the offered solution does not offend a conflict,false otherwise.
Parameters: `iSolution` an array with values, where the first array indexequals the first variable.
Author: J.I. van Hemert
Version: 1.0

2.18

int const **GetNumberOfConflicts** (ValueT const iSolution []) const

Getting the number of conflicts

A way of finding out how many conflicts a solution violates within the CSP. Given a array of size equal to the number of variables, check how many conflicts this solutions violates in the CSP.

Return Value: `number` of violated conflicts.
Parameters: `iSolution` an array with values, where the first array indexequals the first variable.
Author: J.I. van Hemert
Version: 1.0

2.19

unsigned long const **GetNumberOfConstraints** ( ) const

Getting the number of constraints

Get the number of constraints currently in the CSP.

Return Value: `number` of constraints in CSP.
Author: J.I. van Hemert

Version: 1.0

2.20

void const **AddConstraint** (VariableT const iVariable1, VariableT const iVariable2)

Adding constraints

Add a constraint between two variables.

Return Value: `nothing.`
Parameters: `iVariable1` one of the two variables.
`iVariable2` another variable.
Author: J.I. van Hemert
Version: 1.0

2.21

void const **RemoveConstraint** (VariableT const iVariable1, VariableT const iVariable2)

Removing constraints

Removes a constraint between two variables. Frees up memory from conflicts of the constraint.

Return Value: `nothing.`
Parameters: `iVariable1` one of the two variables.
`iVariable2` another variable.
Author: J.I. van Hemert
Version: 1.0

2.22

bool const **IsConstraint** (VariableT const iVariable1, VariableT const iVariable2) const

Testing constraints

Check for the existence of a constraint between two variables.

Return Value:	`true`	when the constraint exists, else it returns false.
Parameters:	`iVariable1`	one of the two variables.
	`iVariable2`	another variable.
Author:	J.I. van Hemert	
Version:	1.0	

2.23

void const **AddConflict** (VariableT const iVariable1, VariableT const iVariable2, ValueT const iValue1, ValueT const iValue2)

Adding conflicts

Add a conflict between the values of two variables.

Return Value:	`nothing.`	
Parameters:	`iVariable1`	one of the two variables.
	`iVariable2`	another variable.
	`iValue1`	value of first variable.
	`iValue2`	value of the second variable.
Author:	J.I. van Hemert	
Version:	1.0	

2.24

void const **RemoveConflict** (VariableT const iVariable1, VariableT const iVariable2, ValueT const iValue1, ValueT const iValue2)

Removing conflicts

Remove a conflict between the values of two variables.

Return Value:	`nothing.`	
Parameters:	`iVariable1`	one of the two variables.
	`iVariable2`	another variable.
	`iValue1`	value of first variable.
	`iValue2`	value of the second variable.
Author:	J.I. van Hemert	

Version: 1.0

2.25

bool const **IsConflict** (VariableT const iVariable1, VariableT const iVariable2, ValueT const iValue1, ValueT const iValue2) const

Testing conflicts

Check for an existing conflict between the values of two variables.

Return Value: `true` for existing constraint with conflict between values,false for non-existing conflict or non-existing constraint.

Parameters:
`iVariable1` one of the two variables.
`iVariable2` another variable.
`iValue1` value of first variable.
`iValue2` value of the second variable.

Author: J.I. van Hemert

Version: 1.0

2.26

bool const **SaveToFileInMatrix** (char const* const iFilename) const

Saving a CSP in matrix format

Save the CSP to a file. The fileformat is simple, first the number of variables is saved, then on the next line all the domainsizes of the variables are saved with spaces between them. A blank line follows, after which the conflicts are saved as follows. For every _possible_ constraint we generate a matrix. First for 1,2 then for 1,3 etc. until n-1,n. The matrix represents the constraint (all zero's if no constraint exists) with k rows for domainsize k of i in i,j and l columns for domainsize l of j in i,j.

Return Value: `true` for success, false for a failure.

Parameters: `iFilename` the filename of the file to save to.

Author: J.I. van Hemert

Version: 1.0

2.27

bool const **SaveToFileInList** (char const* const iFilename) const

Saving a CSP in list format

Save the CSP to a file. On the second line a keyword denotes whether this file contains the conflict pairs or the non-conflict pairs. Then number of variables is saved, on the next line all the domainsizes of the variables are saved with spaces between them. A blank line follows, after which the conflicts are saved as follows. For every non-empty constraint we generate a line that contains first two number denoting the two variables of the constraint followed by a (non-empty) list of domain value pairs. Note that these pairs can either denote conflicts or non-conflicts depending on the keyword mentioned above. Reason for this is to account for efficient saving (if a CSP has a conflict ratio of 99% it is more efficient in disk space to save the non-conflicts).

Return Value: `true` for success, false for a failure.
Parameters: `iFilename` the filename of the file to save to.
Author: J.I. van Hemert @version 1.0

2.28

bool const **LoadFromFile** (char const* iFilename)

Loading a CSP

Load the CSP from a file.

Return Value: `true` for success, false for a failure.
Parameters: `iFilename` the filename of the file to load from.
Author: J.I. van Hemert
Version: 1.0

2.29

bool const **LoadFromStdin** ( )

2.30

bool const **IsDirty** () const

Testing dirty

Check if a CSP file has been touched.

Return Value:	`false` if it is still empty, true if it has been touched.
Author:	J.I. van Hemert
Version:	1.0

2.31

void const **Reset** ()

Resetting the CSP

Removes all constraints, conflicts and domainsizes from the CSP, freeing memory for the conflicts, but without freeing memory for the constraints. The CSP can be reused for a new CSP, because the number of variables is not cleared.

Return Value:	`nothing.`
Author:	J.I. van Hemert
Version:	1.0

2.32

Virtual member-function

This *pure virtual function* has to be implemented by the class that is inherits from this class. It should produce a Constraint Satisfaction Problem according to some specific algorithm.

Return Value:	`nothing.`

2.33

bool const **IsSolvable** () const

This function determines whether this CSP instance is solvable or not.

This function determines whether this CSP instance is solvable or not. It uses standard backtracking to find out, which is *very* slow, especially when you the instance lies near or in the phase transition.

Return Value: `boolean,` false if no solutions or true if at least one solution.
Version: : 1.3

2.34

ErrorC* **Error**

Error handling object

Error handling object

2.35

ConstraintT const **GetConstraint** (VariableT const iVariable1, VariableT const iVariable2) const

Getting constraints

Gets a constraint between two variables. Still not sure if we want this to be public or protected.

Return Value: `a` constraint.
Parameters: `iVariable1` one of the two variables.
`iVariable2` another variable.
See Also: the structure of a constraint in ConstraintS.
Author: J.I. van Hemert
Version: 1.0

2.36

mutable unsigned int **conflict_checks**

To measure the amount of conflict checks performed

To measure the amount of conflict checks performed

2.37

bool const **IsConsistent** (unsigned int current, ValueT const iSolution []) const

Used by Backtrack

Used by Backtrack

2.38

bool const **Backtrack** (unsigned int current, ValueT iSolution []) const

2.39

NumberOfVariablesT **NumberOfVariables**

Number of variable in the csp

Number of variable in the csp

2.40

struct **VariableS**

Type for a variable

Members

Type for a variable

2.40.1

DomainSizeT **DomainSize**

2.41

ConstraintT** **Constraints**

A double indexed array for holding constraints

A double indexed array for holding constraints

2.42

ConstraintT **ConstraintsList**

And a doubly linked list for easy lookup of all constraints

And a doubly linked list for easy lookup of all constraints

2.43

unsigned long **NumberOfConstraints**

Counts the number of constraints currently in the CSP

Counts the number of constraints currently in the CSP

2.44

unsigned long **NumberOfConflictPairs**

Counts the number of pairs of conflicts currently in the CSP

Counts the number of pairs of conflicts currently in the CSP

2.45

unsigned long **NumberOfPossibleConflictPairs**

Counts the number of _possible_ pairs of conflicts currently in the CSP

Counts the number of _possible_ pairs of conflicts currently in the CSP

2.46

VariableS* **Variables**

A single indexed array for holding the variables

A single indexed array for holding the variables

2.47

bool **Dirty**

True if csp has been initialised, false if reset

True if csp has been initialised, false if reset

2.48

bool const **IsValidParameter** (VariableT const iVariable) const

Testing variables

Check for the existence of a variable.

Return Value: `true` for existence, false for non-existence.
Parameters: `iVariable` the variable to check.
Author: J.I. van Hemert
Version: 1.0

2.49

bool const **IsValidParameter** (VariableT const iVariable, ValueT const iValue) const

Testing values

Check for the existence of a variable and if its value lies in its domain.

Return Value: `true` for existence and valid value, false for non-existenceor non-valid value.
Parameters: `iVariable` the variable to check.
`iValue` value of the variable.
Author: J.I. van Hemert
Version: 1.0

2.50

void const **Create** (NumberOfVariablesT const iNumberOfVariables)

Creating a CSP problem

Creates memory needed for holding the CSPs problem. initialises the CSP by allocating enough memory to hold all the constraints. No memory is allocated for the conflicts yet.

Return Value: `nothing.`
Parameters: `iNumberOfVariables` number of variables used in the CSP.
Author: J.I. van Hemert
Version: 1.0

2.51

void const **Destroy** ()

Destroying the CSP

Deallocates memory from the CSP problem. Used in constructors and destructors.

Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.0

2.52

void const **Copy** (CspC const & iCsp)

Copying a CSP

Copies an existing CSP to this one by calling Create and copying all the constraints and conflicts.

Return Value: `nothing.`
Parameters: `iCsp` existing CSP.
Author: J.I. van Hemert @version 1.0

2.53

void const **Swap** (VariableT & iVariable1, VariableT & iVariable2) const

Swap function

Swap the two parameters.

Return Value: `nothing.`
Parameters: `iVariable1` First parameter.
`iVariable2` Second parameter.
Author: J.I. van Hemert
Version: 1.0

2.54

bool const **LoadFromFileInMatrix** (istream & InputFile)

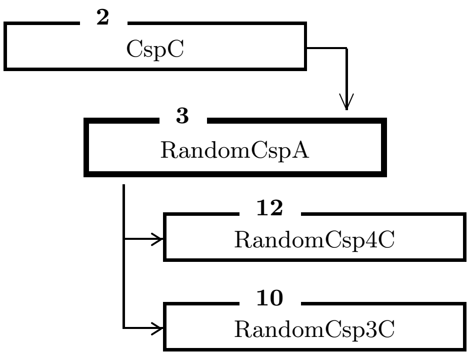
2.55

bool const **LoadFromFileInList** (istream & InputFile)

3 class **RandomCspA** : public CspC

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is an abstract class that can be extended to a class capable of producing Random Binary Constraint Satisfaction Problems. The class should implement the ProduceProblem function.

Author: J.I. van Hemert @version 1.1

3.1 **RandomCspA** (NumberOfVariablesT const iNumberOfVariables)

Constructor

Constructor copies all the parameters for further usage.

Return Value:	`nothing.`	
Parameters:	`iNumberOfVariables`	Number of variables in CSP.
	`iDomainSize`	Size of domain of each variable *or* maximum size of domain.
	`iRandomDomainSize`	If true than a the domainsize will be varied over the variables.
	`iTightness`	Determines percentage of conflicts that are generated.
	`iConnectivity`	Percentage of constraints that are generated.
Author:	J.I. van Hemert	
Version:	1.2	

3.2

~RandomCspA ()

Destructor

For C++'s sake we have to implement the virtual destructor.

Return Value:	`nothing.`
Author:	J.I. van Hemert
Version:	1.2

3.3

void const **InitRandomGenerator** (int const iRandomSeed)

Initialising the randomgenerator

Initialise the randomgenerator using a seed.

Parameters:	`iRandomSeed`	Seed used for initialising the random generator.
	`return`	nothing.
Author:	J.I. van Hemert	
Version:	1.2	

3.4

virtual void const **ProduceProblem** ()

Producing a Random CSP

Produce a constraint satisfaction method using the method of Patrick Prosser, which he mentiones in his article 'Binary Constraint Satisfaction Problems: Some are Harder than Others' in '11th European Conference on Artificial Inteligence (ECAI94)' edited by A. Cohn and published in 1994 by * John Wiley & Sons, Ltd.

Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.2

4 class **ErrorC**

Public Members

4.1 enum **ErrorTypeT**

Type of error that needs to be reported, not that warnings will not halt the program, but all others will.

Type of error that needs to be reported, not that warnings will not halt the program, but all others will. The exitcode of the program will correspond to the sequence here.

4.2 **ErrorC** (char const* const iOwner)

4.3 **˜ErrorC** ()

4.4 void const **Give** (char const* const iError, ErrorTypeT const iErrorType) const

4.5

char* **Owner**

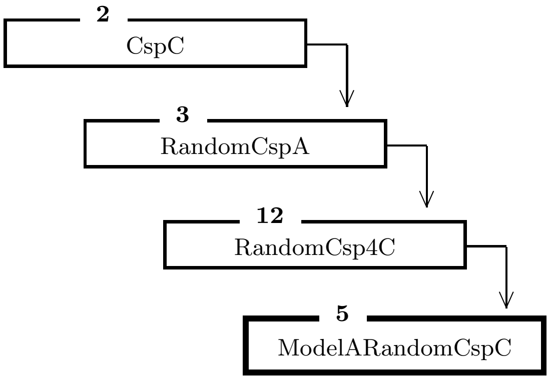
Name string of my owner

Name string of my owner

5 class **ModelARandomCspC** : public RandomCsp4C

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is capable of producing Random Binary Constraint Satisfaction Problems. The usage is quite simple. After the creation of a object of this class with all the correct parameters, the InitRandomGenerator function should be used to set the seeds for the randomgenerator. This has to be done at least once. Then a problem can be generated with the ProduceProblem function and a filename can be produced with the ProduceFileName function

Author: J.I. van Hemert
Version: 1.0

5.1 void const **ProduceProblem** ()

Producing a Random CSP

Produce a constraint satisfaction problem using the method of Model A.

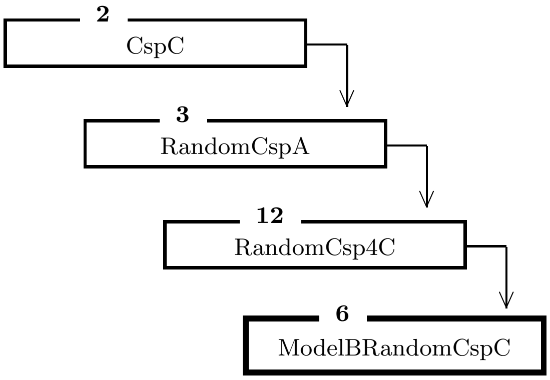
Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.0

6

class **ModelBRandomCspC** : public RandomCsp4C

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is capable of producing Random Binary Constraint Satisfaction Problems. The usage is quite simple. After the creation of a object of this class with all the correct parameters, the InitRandomGenerator function should be used to set the seeds for the randomgenerator. This has to be done at least once. Then a problem can be generated with the ProduceProblem function and a filename can be produced with the ProduceFileName function

Author: J.I. van Hemert
Version: 1.1

6.1

void const **ProduceProblem** ()

Producing a Random CSP

Produce a constraint satisfaction method using the method of Model B.

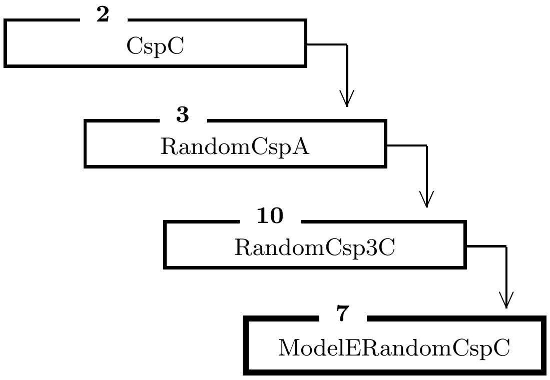
Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.1

7

class **ModelERandomCspC** : public RandomCsp3C

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is capable of producing Random Binary Constraint Satisfaction Problems. The usage is quite simple. After the creation of a object of this class with all the correct parameters, the InitRandomGenerator function should be used to set the seeds for the randomgenerator. This has to be done at least once. Then a problem can be generated with the ProduceProblem function and a filename can be produced with the ProduceFileName function The output is a binary csp instance created with model E.

Author: J.I. van Hemert
Version: 1.4

7.1

void const **ProduceProblem** ()

Producing a Random CSP

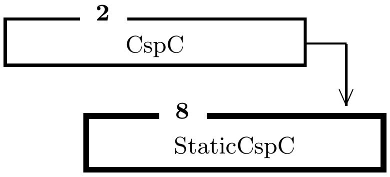
Produce a constraint satisfaction method using ModelE.

Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.2

8 class **StaticCspC** : public CspC

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is only useful to load instances from disk.

Author: J.I. van Hemert @version 1.2

8.1 **StaticCspC** (char* iFileName)

Constructor

Constructor copies all the parameters for further usage.

Return Value: `nothing.`
Parameters: `iFileName` Filename of instance on disk
Author: J.I. van Hemert
Version: 1.2

8.2

~StaticCspC ()

Destructor

For C++'s sake we have to implement the virtual destructor.

Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.2

9 Type definitions for RandomCsp3C

General type definitons

Names

These types can and should be used as a type for variables in your own code.

9.1 typedef double **TightnessT**

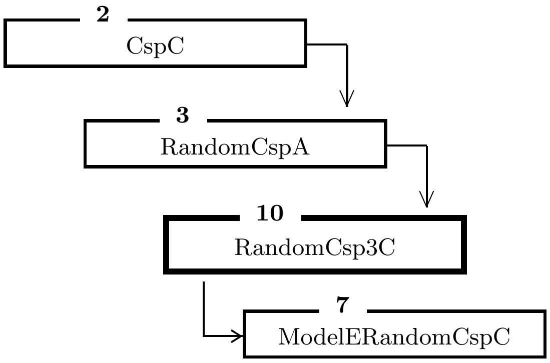
Determines the number of conflicts the csp might contain

Determines the number of conflicts the csp might contain

10 class **RandomCsp3C** : public RandomCspA

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is a empty class that can be extended to a class capable of producing Random Binary Constraint Satisfaction Problems. The class should implement the ProduceProblem function. This class is for models that use three parameters to describe instances: number of variables, domain size and tightness. Also provided is a parameter to create random domain sizes over the variables.

Author: J.I. van Hemert @version 1.2

10.1

RandomCsp3C (NumberOfVariablesT const iNumberOfVariables, DomainSizeT const iDomainSize, bool const iRandomDomainSize, TightnessT const iTightness)

Constructor

Constructor copies all the parameters for further usage.

Return Value:	`nothing.`	
Parameters:	`iNumberOfVariables`	Number of variables in CSP.
	`iDomainSize`	Size of domain of each variable *or* maximum size of domain.
	`iRandomDomainSize`	If true than a the domainsize will be varied over the variables.
	`iTightness`	Determines percentage of conflicts that are generated.
	`iConnectivity`	Percentage of constraints that are generated.
Author:	J.I. van Hemert	
Version:	1.2	

10.2

~RandomCsp3C ()

Destructor

For C++'s sake we have to implement the virtual destructor.

Return Value:	`nothing.`
Author:	J.I. van Hemert
Version:	1.2

10.3

void **ProduceFileName** (char* FileName)

Producing a filename

Create a filename from all the parameters of the CSP.

Return Value: `nothing` (FileName is the returnvalue).
Parameters: `FileName` reference to a string (char *) that should haveenough memory for holding the filename produced by this function,maximum required is 100 bytes.
Author: J.I. van Hemert
Version: 1.2

10.4

virtual void const **ProduceProblem** ()

Producing a Random CSP

Produce a constraint satisfaction method using the method of Patrick Prosser, which he mentiones in his article 'Binary Constraint Satisfaction Problems: Some are Harder than Others' in '11th European Conference on Artificial Inteligence (ECAI94)' edited by A. Cohn and published in 1994 by * John Wiley & Sons, Ltd.

Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.2

11 Type definitions for RandomCsp4C

General type definitons

Names

These types can and should be used as a type for variables in your own code.

11.1 typedef double **TightnessT**

Determines the number of conflicts a constraint contains

Determines the number of conflicts a constraint contains

11.2 typedef double **ConnectivityT**

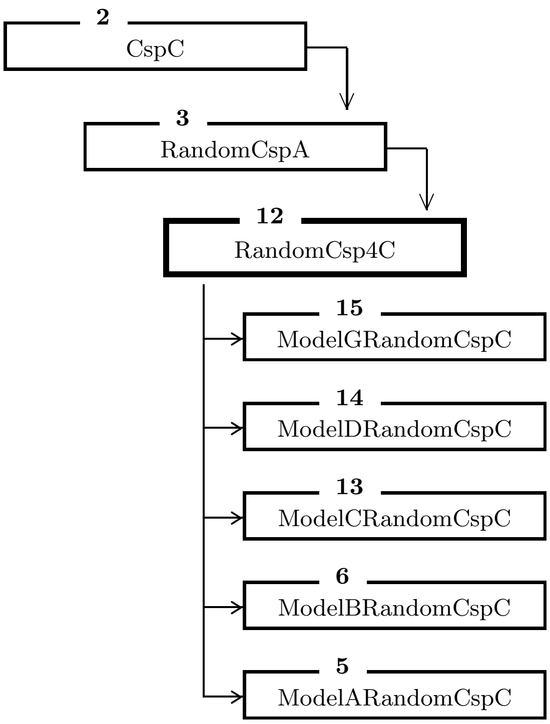
Determines the number of nodes in the network

Determines the number of nodes in the network

12 class **RandomCsp4C** : public RandomCspA

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is a empty class that can be extended to a class capable of producing Random Binary Constraint Satisfaction Problems. The class should implement the ProduceProblem function. This base class provides an interface to models that use four parameters to describe instances: number of variables, domain size, tightness of constraints and connectivity (density of constraints). Also provided is a parameter to ask for random domain sizes over the variables.

Author: J.I. van Hemert @version 1.2

12.1

RandomCsp4C (NumberOfVariablesT const iNumberOfVariables, DomainSizeT const iDomainSize, bool const iRandomDomainSize, TightnessT const iTightness, ConnectivityT const iConnectivity)

Constructor

Constructor copies all the parameters for further usage.

Return Value:	`nothing.`	
Parameters:	`iNumberOfVariables`	Number of variables in CSP.
	`iDomainSize`	Size of domain of each variable *or* maximum size of domain.
	`iRandomDomainSize`	If true than a the domainsize will be varied over the variables.
	`iTightness`	Determines percentage of conflicts that are generated.
	`iConnectivity`	Percentage of constraints that are generated.
Author:	J.I. van Hemert	
Version:	1.2	

12.2

~RandomCsp4C ()

Destructor

For C++'s sake we have to implement the virtual destructor.

Return Value:	`nothing.`
Author:	J.I. van Hemert
Version:	1.2

12.3

void **ProduceFileName** (char* FileName)

Producing a filename

Create a filename from all the parameters of the CSP.

Return Value: `nothing` (FileName is the returnvalue).
Parameters: `FileName` reference to a string (char *) that should haveenough memory for holding the filename produced by this function,maximum required is 100 bytes.
Author: J.I. van Hemert
Version: 1.2

12.4

virtual void const **ProduceProblem** ()

Producing a Random CSP

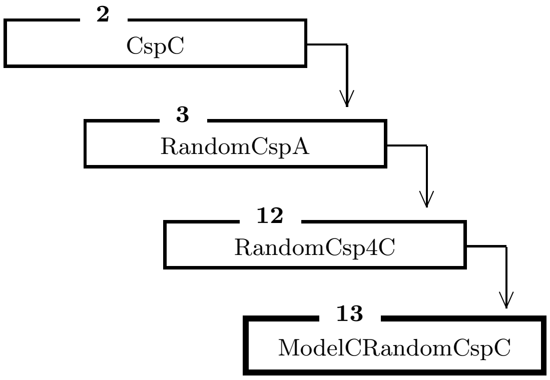
Produce a constraint satisfaction method using the method of Patrick Prosser, which he mentiones in his article 'Binary Constraint Satisfaction Problems: Some are Harder than Others' in '11th European Conference on Artificial Inteligence (ECAI94)' edited by A. Cohn and published in 1994 by * John Wiley & Sons, Ltd.

Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.2

13 class **ModelCRandomCspC** : public RandomCsp4C

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is capable of producing Random Binary Constraint Satisfaction Problems. The usage is quite simple. After the creation of a object of this class with all the correct parameters, the InitRandomGenerator function should be used to set the seeds for the randomgenerator. This has to be done at least once. Then a problem can be generated with the ProduceProblem function and a filename can be produced with the ProduceFileName function

Author: J.I. van Hemert
Version: 1.5

13.1 void const **ProduceProblem** ()

Producing a Random CSP

Produce a constraint satisfaction method using the method of Model C.

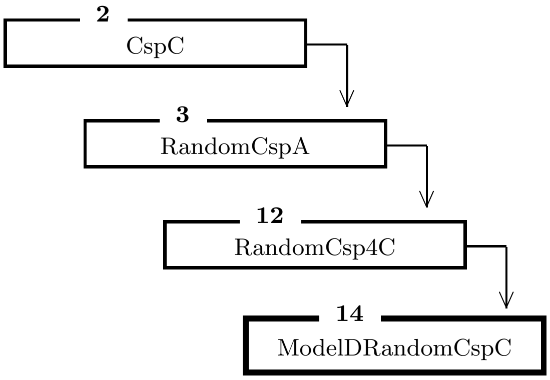
Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.5

14

class **ModelDRandomCspC** : public RandomCsp4C

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is capable of producing Random Binary Constraint Satisfaction Problems. The usage is quite simple. After the creation of a object of this class with all the correct parameters, the InitRandomGenerator function should be used to set the seeds for the randomgenerator. This has to be done at least once. Then a problem can be generated with the ProduceProblem function and a filename can be produced with the ProduceFileName function

Author: J.I. van Hemert
Version: 1.5

14.1

void const **ProduceProblem** ()

Producing a Random CSP

Produce a constraint satisfaction method using the method of Model D.

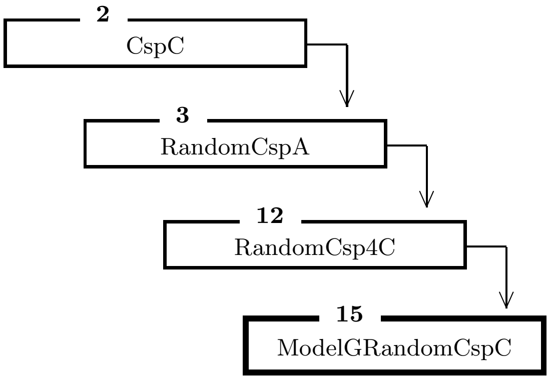
Return Value: `nothing.`
Author: J.I. van Hemert
Version: 1.5

15

class **ModelGRandomCspC** : public RandomCsp4C

Random Constraint Satisfaction Problems

Inheritance



Public Members

This class is capable of producing Random Binary Constraint Satisfaction Problems. The usage is quite simple. After the creation of a object of this class with all the correct parameters, the InitRandomGenerator function should be used to set the seeds for the randomgenerator. This has to be done at least once. Then a problem can be generated with the ProduceProblem function and a filename can be produced with the ProduceFileName function

Produce a constraint satisfaction method using such that it is GA-Hard. That is, the problem is very deceptive and will cause a standard EA to get stuck in a trap, Idea originates from research done in Antwerp by J.I. van Hemert (me), L. Schoofs and B. Naudts.

By using a geometric distribution we bias which value pairs get the most conflicts added. The lower the value of a variable the more chance it has to be selected for a conflict pair. One exception is the pair (0,0), this pair is *never* formed, thereby making sure a solution always exist, namely 0,0,0,...,0.

Author: J.I. van Hemert
Version: 1.4

15.1

void const **ProduceProblem** ()

Producing a Random CSP

Produce a constraint satisfaction problem using method G (GA-hard)

Return Value:	`nothing.`
Author:	J.I. van Hemert
Version:	1.5

Class Graph