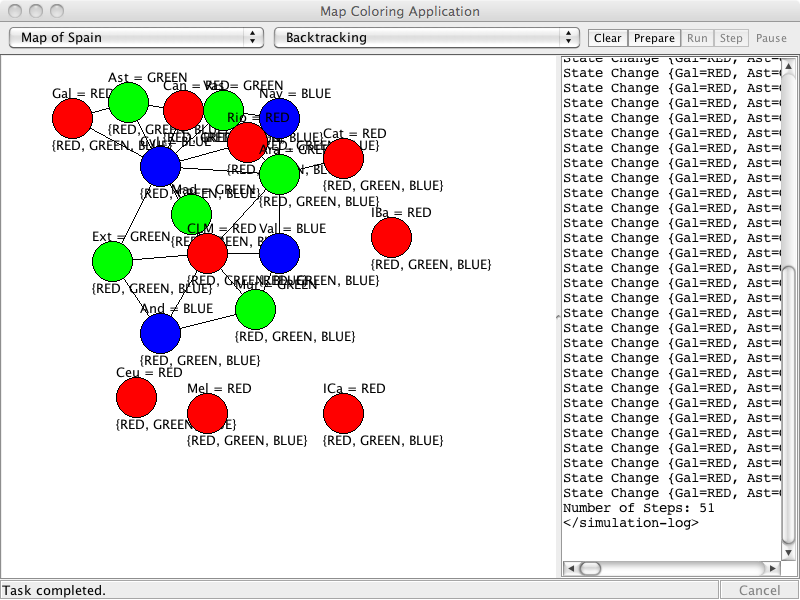


**SISTEMAS INTELIGENTES.**

**PRÁCTICA DE CSPs 1**

1. Esta práctica se divide en dos tareas.

1.a. La primera tarea es colorear el mapa de las comunidades autónomas y las dos ciudades autónomas de España (en total 19 entidades), tal y como muestra la figura 1.

Figura 1. CSPs Mapa de España

Para realizar la práctica nos basaremos en las clases ya definidas en la librería y definimos dos clases nuevas:

1. **intsys.csp.MapCSPSpain.** Esta clase definirá el problema de satisfacción de restricciones (CSP).

Será similar a la clase aima.core.applications.search.csp.MapCSP

1. **intsys.csp.MapColoringAppSpain.** Esta clase implementará una aplicación para ejecutar los algoritmos de resolución de problemas de satisfacción de restricciones sobre el problema especificado.

Será similar a la clase aima.gui.applications.search.csp.MapColoringApp

Una vez finalizado rellene las siguientes tablas comparativa, una para el mapa de España y otra para el mapa de Australia:

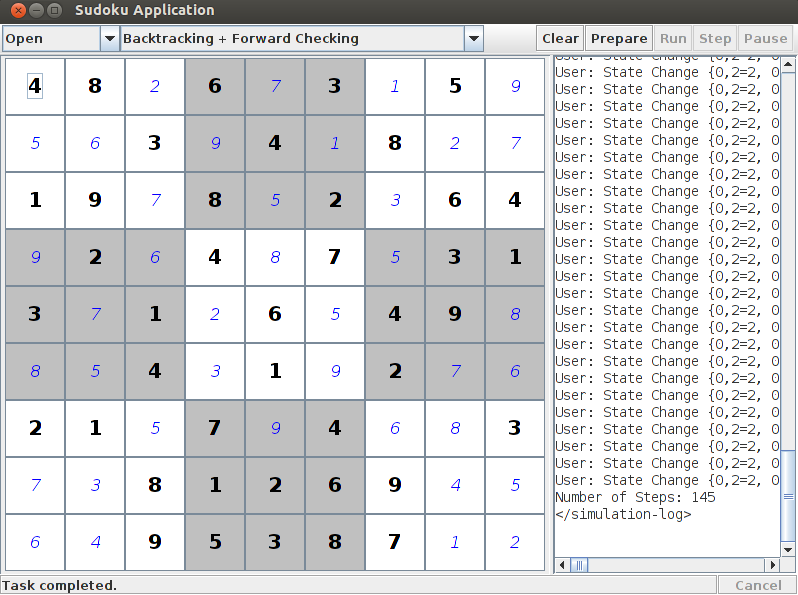
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Australia** | Back[[1]](#footnote-1) | Back + MRV &DEG | Back + FC[[2]](#footnote-2) | Back + FC + MRV | Back + FC + LCV | Back + AC3 | Back + AC3 + MRV & DEG +LCV | Min Conflicts |
| Número Pasos para obtener solución | 27 | 15 | 20 | 11 | 20 | 16 | 10 | 7 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **España** | Back[[3]](#footnote-3) | Back + MRV &DEG | Back + FC[[4]](#footnote-4) | Back + FC + MRV | Back + FC + LCV | Back + AC3 | Back + AC3 + MRV & DEG +LCV | Min Conflicts |
| Número Pasos para obtener solución | 30 | 134 | 30 | 32 | 29 | 28 | 29 | 51 |

**Conclusiones del experimento:**

En este caso el algoritmo que implementa BACK + AC3 proporciona ante un orden inusual mejores resultados que los demás algoritmos.

1.b. La segunda tarea es implementar en la clase AllDiffConstraint, incluida en los proyectos de clase, el método **isSatisfiedWith**, que permita utilizar restricciones n-arias en el sudoku y definir el problema.

**

Una vez finalizado rellene la tabla comparativa:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Back[[5]](#footnote-5) | Back + MRV &DEG | Back + FC[[6]](#footnote-6) | Back + FC + MRV | Back + FC + LCV | Back + AC3 | Back + AC3 + MRV & DEG +LCV | Min Conflicts |
| Número Pasos para obtener Solución |  |  |  |  |  |  |  |  |

**Conclusiones del experimento:**

P.ej. ¿Sería necesario evaluar más información sobre los algoritmos?

MAPCSPSPAIN

package aima.core.search.csp;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Artificial Intelligence A Modern Approach (3rd Ed.): Figure 6.1, Page 204.<br>

\* <br>

\* The principal states and territories of Australia. Coloring this map can be

\* viewed as a constraint satisfaction problem (CSP). The goal is to assign

\* colors to each region so that no neighboring regions have the same color.

\*

\* @author Ruediger Lunde

\* @author Mike Stampone

\*/

public class MapCSP extends CSP {

public static final Variable ANDALUCIA = new Variable("ANDALUCIA");

public static final Variable MURCIA = new Variable("MURCIA");

public static final Variable CEUTA = new Variable("CEUTA");

public static final Variable MELILLA = new Variable("MELILLA");

public static final Variable ISLASCANARIAS = new Variable("ISLASCANARIAS");

public static final Variable VALENCIA = new Variable("VALENCIA");

public static final Variable EXTREMADURA = new Variable("EXTREMADURA");

public static final Variable ISLASBALEARES = new Variable("ISLASBALEARES");

public static final Variable CASTILLALAMANCHA = new Variable("CASTILLALAMANCHA");

public static final Variable MADRID = new Variable("MADRID");

public static final Variable CASTILLAYLEON = new Variable("CASTILLAYLEON");

public static final Variable ARAGON = new Variable("ARAGON");

public static final Variable CATALUNYA = new Variable("CATALUNYA");

public static final Variable GALICIA = new Variable("GALICIA");

public static final Variable LARIOJA = new Variable("LARIOJA");

public static final Variable PAISVASCO = new Variable("PAISVASCO");

public static final Variable NAVARRA = new Variable("NAVARRA");

public static final Variable CANTABRIA = new Variable("CANTABRIA");

public static final Variable ASTURIAS = new Variable("ASTURIAS");

public static final String RED = "RED";

public static final String GREEN = "GREEN";

public static final String BLUE = "BLUE";

/\*\*

\* Returns the principle states and territories of Australia as a list of

\* variables.

\*

\* @return the principle states and territories of Australia as a list of

\* variables.

\*/

private static List<Variable> collectVariables() {

List<Variable> variables = new ArrayList<Variable>();

variables.add(ANDALUCIA);

variables.add(EXTREMADURA);

variables.add(MURCIA);

variables.add(CEUTA);

variables.add(MELILLA);

variables.add(VALENCIA);

variables.add(ISLASCANARIAS);

variables.add(ISLASBALEARES);

variables.add(CASTILLALAMANCHA);

variables.add(MADRID);

variables.add(CASTILLAYLEON);

variables.add(ARAGON);

variables.add(CATALUNYA);

variables.add(GALICIA);

variables.add(LARIOJA);

variables.add(PAISVASCO);

variables.add(CANTABRIA);

variables.add(ASTURIAS);

variables.add(NAVARRA);

return variables;

}

/\*\*

\* Constructs a map CSP for the principal states and territories of

\* Australia, with the colors Red, Green, and Blue.

\*/

public MapCSP() {

super(collectVariables());

Domain colors = new Domain(new Object[] { RED, GREEN, BLUE });

for (Variable var : getVariables())

setDomain(var, colors);

addConstraint(new NotEqualConstraint(ANDALUCIA, EXTREMADURA));

addConstraint(new NotEqualConstraint(ANDALUCIA, CASTILLALAMANCHA));

addConstraint(new NotEqualConstraint(ANDALUCIA, MURCIA));

addConstraint(new NotEqualConstraint(MURCIA, CASTILLALAMANCHA));

addConstraint(new NotEqualConstraint(MURCIA, VALENCIA));

addConstraint(new NotEqualConstraint(EXTREMADURA, CASTILLALAMANCHA));

addConstraint(new NotEqualConstraint(EXTREMADURA, CASTILLAYLEON));

addConstraint(new NotEqualConstraint(MADRID, CASTILLALAMANCHA));

addConstraint(new NotEqualConstraint(MADRID, CASTILLAYLEON));

addConstraint(new NotEqualConstraint(VALENCIA, CASTILLALAMANCHA));

addConstraint(new NotEqualConstraint(VALENCIA, ARAGON));

addConstraint(new NotEqualConstraint(ARAGON, CATALUNYA));

addConstraint(new NotEqualConstraint(ARAGON, CASTILLALAMANCHA));

addConstraint(new NotEqualConstraint(ARAGON, CASTILLAYLEON));

addConstraint(new NotEqualConstraint(LARIOJA, NAVARRA));

addConstraint(new NotEqualConstraint(ARAGON, NAVARRA));

addConstraint(new NotEqualConstraint(LARIOJA, CASTILLAYLEON));

addConstraint(new NotEqualConstraint(LARIOJA, PAISVASCO));

addConstraint(new NotEqualConstraint(NAVARRA, PAISVASCO));

addConstraint(new NotEqualConstraint(PAISVASCO, CANTABRIA));

addConstraint(new NotEqualConstraint(PAISVASCO, CASTILLAYLEON));

addConstraint(new NotEqualConstraint(CANTABRIA, CASTILLAYLEON));

addConstraint(new NotEqualConstraint(ASTURIAS, CANTABRIA));

addConstraint(new NotEqualConstraint(ASTURIAS, GALICIA));

addConstraint(new NotEqualConstraint(ASTURIAS, CASTILLAYLEON));

addConstraint(new NotEqualConstraint(CASTILLAYLEON, GALICIA));

}

}

MAPP COLORING

package aima.gui.applications.search.csp;

import java.awt.Color;

import aima.core.search.csp.Assignment;

import aima.core.search.csp.BacktrackingStrategy;

import aima.core.search.csp.CSP;

import aima.core.search.csp.CSPStateListener;

import aima.core.search.csp.Domain;

import aima.core.search.csp.ImprovedBacktrackingStrategy;

import aima.core.search.csp.MapCSP;

import aima.core.search.csp.MinConflictsStrategy;

import aima.core.search.csp.SolutionStrategy;

import aima.core.util.datastructure.FIFOQueue;

import aima.gui.framework.AgentAppController;

import aima.gui.framework.AgentAppEnvironmentView;

import aima.gui.framework.AgentAppFrame;

import aima.gui.framework.MessageLogger;

import aima.gui.framework.SimpleAgentApp;

import aima.gui.framework.SimulationThread;

/\*\*

\* Application which demonstrates basic constraint algorithms based on map

\* coloring problems. It shows the constraint graph, lets the user select a

\* solution strategy, and allows then to follow the progress step by step. For

\* pragmatic reasons, the implementation uses the agent framework, even though

\* there is no agent and only a dummy environment.

\*

\* @author Ruediger Lunde

\*/

public class MapColoringApp extends SimpleAgentApp {

/\*\* Returns an <code>CSPView</code> instance. \*/

@Override

public AgentAppEnvironmentView createEnvironmentView() {

return new CSPView();

}

/\*\* Returns a <code>MapColoringFrame</code> instance. \*/

@Override

public AgentAppFrame createFrame() {

return new MapColoringFrame();

}

/\*\* Returns a <code>MapColoringController</code> instance. \*/

@Override

public AgentAppController createController() {

return new MapColoringController();

}

// ///////////////////////////////////////////////////////////////

// main method

/\*\*

\* Starts the application.

\*/

public static void main(String args[]) {

new MapColoringApp().startApplication();

}

// ///////////////////////////////////////////////////////////////

// some inner classes

/\*\*

\* Adds some selectors to the base class and adjusts its size.

\*/

protected static class MapColoringFrame extends AgentAppFrame {

private static final long serialVersionUID = 1L;

public static String ENV\_SEL = "EnvSelection";

public static String STRATEGY\_SEL = "SearchSelection";

public MapColoringFrame() {

setTitle("Map Coloring Application");

setSelectors(new String[] { ENV\_SEL, STRATEGY\_SEL }, new String[] {

"Select Environment", "Select Solution Strategy" });

setSelectorItems(ENV\_SEL, new String[] {

"MAPA DE ESPAÑITA",

"Map of Australia NSW=BLUE (for LCV)",

"Map of Australia WA=RED (for LCV)"}, 0);

setSelectorItems(STRATEGY\_SEL, new String[] { "Backtracking",

"Backtracking + MRV & DEG",

"Backtracking + Forward Checking",

"Backtracking + Forward Checking + MRV",

"Backtracking + Forward Checking + LCV",

"Backtracking + AC3",

"Backtracking + AC3 + MRV & DEG + LCV",

"Min-Conflicts (50)" }, 0);

setEnvView(new CSPView());

setSize(800, 600);

}

}

/\*\*

\* Defines how to react on standard simulation button events.

\*/

protected static class MapColoringController extends AgentAppController {

protected CSPEnvironment env;

protected SolutionStrategy strategy;

protected FIFOQueue<CSPEnvironment.StateChangeAction> actions;

protected int actionCount;

protected MapColoringController() {

env = new CSPEnvironment();

actions = new FIFOQueue<CSPEnvironment.StateChangeAction>();

}

protected CSPView getCSPView() {

return (CSPView) frame.getEnvView();

}

/\*\* Prepares next simulation. \*/

@Override

public void clear() {

prepare(null);

}

/\*\*

\* Creates a CSP and updates the environment as well as its view.

\*/

@Override

public void prepare(String changedSelector) {

AgentAppFrame.SelectionState selState = frame.getSelection();

CSP csp = null;

CSPView view = getCSPView();

switch (selState.getValue(MapColoringFrame.ENV\_SEL)) {

case 0:

csp = new MapCSP();

break;

}

view.clearMappings();

view.setPositionMapping(MapCSP.ANDALUCIA, 5, 10);

view.setPositionMapping(MapCSP.MURCIA, 15, 3);

view.setPositionMapping(MapCSP.CASTILLALAMANCHA, 20, 15);

view.setPositionMapping(MapCSP.ARAGON, 30, 5);

view.setPositionMapping(MapCSP.EXTREMADURA, 35, 15);

view.setPositionMapping(MapCSP.MADRID, 30, 23);

view.setPositionMapping(MapCSP.VALENCIA, 33, 30);

view.setPositionMapping(MapCSP.ISLASBALEARES, 5, 10);

view.setPositionMapping(MapCSP.ISLASCANARIAS, 6, 12);

view.setPositionMapping(MapCSP.GALICIA, 5, 11);

view.setPositionMapping(MapCSP.PAISVASCO, 9, 30);

view.setPositionMapping(MapCSP.NAVARRA, 7, 11);

view.setPositionMapping(MapCSP.LARIOJA, 5, 50);

view.setPositionMapping(MapCSP.CANTABRIA, 5, 20);

view.setPositionMapping(MapCSP.CATALUNYA, 5, 80);

view.setPositionMapping(MapCSP.CEUTA, 10, 90);

view.setPositionMapping(MapCSP.MELILLA, 10, 100);

view.setPositionMapping(MapCSP.CASTILLAYLEON, 5, 110);

view.setPositionMapping(MapCSP.ASTURIAS, 5, 40);

view.setColorMapping(MapCSP.RED, Color.RED);

view.setColorMapping(MapCSP.GREEN, Color.GREEN);

view.setColorMapping(MapCSP.BLUE, Color.BLUE);

actions.clear();

actionCount = 0;

env.init(csp);

view.setEnvironment(env);

}

/\*\* Checks whether simulation can be started. \*/

@Override

public boolean isPrepared() {

return env.getCSP() != null

&& (!actions.isEmpty() || env.getAssignment() == null);

}

/\*\* Starts simulation. \*/

@Override

public void run(MessageLogger logger) {

logger.log("<simulation-log>");

prepareActions();

try {

while (!actions.isEmpty() && !frame.simulationPaused()) {

env.executeAction(null, actions.pop());

actionCount++;

Thread.sleep(200);

}

logger.log("Number of Steps: " + actionCount);

// logger.log(getStatistics());

} catch (InterruptedException e) {

// nothing to do here.

}

logger.log("</simulation-log>\n");

}

/\*\* Performs a simulation step. \*/

@Override

public void step(MessageLogger logger) {

prepareActions();

if (!actions.isEmpty()) {

env.executeAction(null, actions.pop());

actionCount++;

if (actions.isEmpty())

logger.log("Number of Steps: " + actionCount);

}

}

/\*\*

\* Starts the selected constraint solver and fills the action list if

\* necessary.

\*/

protected void prepareActions() {

ImprovedBacktrackingStrategy iStrategy = null;

if (actions.isEmpty()) {

SolutionStrategy strategy = null;

switch (frame.getSelection().getValue(

MapColoringFrame.STRATEGY\_SEL)) {

case 0:

strategy = new BacktrackingStrategy();

break;

case 1: // MRV + DEG

strategy = new ImprovedBacktrackingStrategy

(true, true, false, false);

break;

case 2: // FC

iStrategy = new ImprovedBacktrackingStrategy();

iStrategy.setInference(ImprovedBacktrackingStrategy

.Inference.FORWARD\_CHECKING);

break;

case 3: // MRV + FC

iStrategy = new ImprovedBacktrackingStrategy

(true, false, false, false);

iStrategy.setInference(ImprovedBacktrackingStrategy

.Inference.FORWARD\_CHECKING);

break;

case 4: // FC + LCV

iStrategy = new ImprovedBacktrackingStrategy

(false, false, false, true);

iStrategy.setInference(ImprovedBacktrackingStrategy

.Inference.FORWARD\_CHECKING);

break;

case 5: // AC3

strategy = new ImprovedBacktrackingStrategy

(false, false, true, false);

break;

case 6: // MRV + DEG + AC3 + LCV

strategy = new ImprovedBacktrackingStrategy

(true, true, true, true);

break;

case 7:

strategy = new MinConflictsStrategy(50);

break;

}

if (iStrategy != null)

strategy = iStrategy;

try {

strategy.addCSPStateListener(new CSPStateListener() {

@Override

public void stateChanged(Assignment assignment, CSP csp) {

actions.add(new CSPEnvironment.StateChangeAction(

assignment, csp));

}

@Override

public void stateChanged(CSP csp) {

actions.add(new CSPEnvironment.StateChangeAction(

csp));

}

});

strategy.solve(env.getCSP().copyDomains());

} catch (Exception e) {

e.printStackTrace();

}

}

}

/\*\* Updates the status of the frame after simulation has finished. \*/

public void update(SimulationThread simulationThread) {

if (simulationThread.isCanceled()) {

frame.setStatus("Task canceled.");

} else if (frame.simulationPaused()) {

frame.setStatus("Task paused.");

} else {

frame.setStatus("Task completed.");

}

}

}

}

1. Back= Bactraking [↑](#footnote-ref-1)
2. FC = Forward Checking [↑](#footnote-ref-2)
3. Back= Bactraking [↑](#footnote-ref-3)
4. FC = Forward Checking [↑](#footnote-ref-4)
5. Back= Bactraking [↑](#footnote-ref-5)
6. FC = Forward Checking [↑](#footnote-ref-6)