A.1 Pseudocode of the DP2PN Module

This section shows the individual components of the DP2PN module in a Java inspired pseudocode style notation. The problem specific Main and State classes are generated automatically by JavaCC. They are shown here for the LINSRCS problem. All other classes are problem independent; they represent general data structures and functions needed across all DP problems. Classes that are rather trivial extensions of basic structures (StateList extends List, StateSet extends Set, and NodeSet extends Set, StateValueMap extends Map) are not shown here.

A.1.1 Main Class for LINSRCS

//This Main class has been  
//automatically generated by the DPspecificationParser  
//for the problem: LINSRCS

public class LINSRCSMain {  
    String problemName="LINSRCS";  
    private static double[] prob = {.2,.5,.3};  
    private static int N = prob.length;  

    void main() {  
        State goalState = new State(setOfAllItems);  
        StateValueMap baseStatesWithValues = new StateValueMap();  
        baseStatesWithValues.put(new State(emptySet),  
                new Double(0.0));

        DPInstance theInstance  
            = new DPInstance(goalState,baseStatesWithValues);
theInstance.setTransitionOperator("+");
theInstance.setMinOrMax("min");

//make table entries automatically
StateSet statesSeen = new StateSet(); //keep track for which
//states table entries
//have been made

//add goal state to statesSeen
statesSeen.add(goalState);
LinkedList stateQueue = new LinkedList();
//initialize queue with the goal state
stateQueue.addLast(goalState);
//loop as long as there is something in the queue
while (stateQueue.size() > 0) {
    //dequeue the first state and make it the current state
    State currentState = (State) stateQueue.removeFirst();
    //check whether the current state is a base state as
    //defined in DPFE_BASE_CONDITIONS section; if so, add
    //it and its value to baseStatesWithValues
    if (determineWhetherBaseState(currentState) == true) {
        baseStatesWithValues.put(currentState,
                                new Double(determineBaseStateValue(currentState)));
    }
    //if current state is not a base state...
    if (!baseStatesWithValues.containsKey(currentState)) {
        //create the decision set
        NodeSet _setInDDN0 = new NodeSet();
        { //extra block so _i reusable
            for (int _i=((Integer)currentState.getCoordinate(0)).intValue();
                _i<=((Integer)currentState.getCoordinate(1)).intValue()-1;
                _i++) {
                _setInDDN0.add(new Integer(_i));
            }
        } //close extra block
        NodeSet decisionSet = _setInDDN0;
        //loop over all decisions now
        for (Iterator it=decisionSet.iterator();it.hasNext();) {
            Integer currentDecisionCandidate = (Integer)it.next();
            Decision d = new Decision(currentDecisionCandidate.intValue());
            d.setDecisionPrefix("k=");
            //determine the successor states of current state
StateList successorStates = calculateTransformation(currentState, d);
//determine the transition weights given the current state
ArrayList transitionWeights = calculateTransitionWeights(currentState, d);
//generate table entry
DPInstanceTableEntry entry = new DPInstanceTableEntry(
currentState, d, calculateReward(currentState, d), successorStates, transitionWeights);
theInstance.addTableEntry(entry);
//enqueue unseen destination states by iterating over successorStates
for (Iterator it2 = successorStates.iterator(); it2.hasNext();)
{
State currentSuccessor = (State) it2.next();
if (!statesSeen.contains(currentSuccessor)) {
stateQueue.addLast(currentSuccessor);
//mark state as seen
statesSeen.add(currentSuccessor);
}
}
} //end of for loop over the decisions
} //end of if
} //end of loop once queue is empty

//Build the BellmanNet from the instance.
BellmanNet bn = theInstance.buildBellmanPNTopDown(problemName);
//Output the BellmanNet as incidence matrix.
bn.toIncidenceMatrix();

 }//end of main() method

double calculateReward(State s, Decision d) {
    double result;
    result = (N+1-size(((NodeSet)((NodeSet)s.getCoordinate(0)).clone())))
        *prob[d.getDecision()];
    return result;
}
StateList calculateTransformation(State s, Decision d) {
    StateList result = new StateList();
    NodeSet items = ((NodeSet)
        ((NodeSet) s.getCoordinate(0)).clone());
    NodeSet _setExplicit2 = new NodeSet();
    _setExplicit2.add(new Integer(d.getDecision()));
    items.removeAll(_setExplicit2);
    result.add(new State(items));
    return result;
}

double determineBaseStateValue(State s) {
    if (s.getCoordinate(0) == s.getCoordinate(1)) return 0.0;
    return Double.NaN; //NaN denotes: not a base state
}

boolean determineWhetherBaseState(State s) {
    if (Double.isNaN(determineBaseStateValue(s))) {
        return false;
    }
    return true;
}

}//end of MCMMain class

A.1.2 State Class for LINSRCS

//This State class has been
//automatically generated by the DPspecificationParser
//for the problem: LINSRCS

class State {
    List theState;

    //Constructor for State class
    State(NodeSet items) {
        theState = new ArrayList();
        theState.add(items);
    } //end of constructor

    //Determine the lexicographical order of two states.
    //Iterate over instance variable theState
    //and compare each coordinate with state2.
int compareTo(Object state2) {
    if states are equal return 0;
    if theState<state2 return -1;
    if theState>state2 return 1;
}

//Returns the i-th coordinate of this state
Object getCoordinate(int i) {
    return theState.get(i);
}

//Set i-th coordinate in the state to object o
public void setCoordinate(int i, Object o) {
    theState.set(i,o);
}

}//end of State class

A.1.3 Decision Class

//For most DP problems a decision is essentially an int
//plus a String prefix like "k=" or "goto"

public class Decision {

    private String decisionPrefix;
    private int decision;

    //constructor
    Decision(int d) {
        decision=d;
        decisionPrefix="";
    }

    //accessor method for the int core of the decision
    //without the prefix
    int getDecision() {
        return decision;
    }

    //accessor method to set the decisionPrefix
    void setDecisionPrefix(String s) {
        decisionPrefix=s;
    }
}
A.1.4 DPInstanceTableEntry Class

//Represents info for the DPInstance class
//format:
// state--decision--reward--List of nextStates--List of weights

class DPInstanceTableEntry {
    State state;
    Decision decision;
    double reward;
    StateList nextStates;
    ArrayList transitionWeights;

    //constructor and basic accessor methods not shown here
}

A.1.5 DPInstance Class

//DP instance represents all the information necessary to
//build a complete state diagram for the DP instance

class DPInstance {
    ArrayList stateDecisionRewardTransformationTable;
    private String operatorForTransition;
        //e.g. "+" for sum transition, or "*" for mult transition
    private String minOrMax; //min or max problem?
    private State goalState;
    private StateValueMap baseStatesWithValues; //store base states
        //along with values

    //The constructor creates a DP instance with no table entries.
    //Add entries with add() method.
    public DPInstance(State goal, StateValueMap bases) {
        ...
    }

    //Accessor methods not shown
    ...

    void addTableEntry(DPInstanceTableEntry e){
        ...
    }
}
stateDecisionRewardTransformationTable.add(e);
}

//The following is the central method where the BellmanNet
//is built and returned.
BellmanNet buildBellmanPNTopDown(String netName) {
    //initialize place and transition counters
    int sp=0; //for state places (not used for naming,
    //only for graphical layout)
    int ip=0; //for intermediate places (i.e. places
    //not associated with a state)
    int ep=0; //for enabling places(allow eval
    //transitions to only fire once)
    int st=0; //for evaluation transitions
    //the type of operator ("+","*" ) is specified
    //in the instance variable operatorForTransition
    int mt=0; //for min transitions

    StateSet statesBuilt = new StateSet();
    //to keep track which states are already built
    //in the Bellman Net

    //create an empty Bellman net
    BellmanNet bn=new BellmanNet(netName);

    //build the goal state
    Place goalPlace=new StatePlace(goalState.toString());
    bn.addPlace(goalPlace);
    statesBuilt.add(goalState);

    LinkedList stateQueue = new LinkedList();
    //initialize queue with the goal state
    stateQueue.addLast(goalState);

    //loop as long as there is something in the queue
    while (stateQueue.size()>0) {
        //dequeue the first state and make it the current state
        //it has already been built when it was discovered
        State currentState=(State) stateQueue.removeFirst();
        if (!baseStatesWithValues.containsKey(currentState)) {
            //regular state?
            //count how many times currentState appears on left
            //hand side of the table now
            int stateAppearanceInTheTableCounter=0;
            for(Iterator i
=stateDecisionRewardTransformationTable.iterator();
   i.hasNext();) {
   DPInstanceTableEntry anEntry
      = (DPInstanceTableEntry) i.next();
   if (anEntry.getState().equals(currentState)) {
      stateAppearanceInTableCounter++;
   }
   } //end of for
   if (stateAppearanceInTableCounter>0) {
      // Making sure that current state
      // has successors. Flawed BellmanNets might have
      // states that do not have successor states,
      // but are not base states either.
      mt++; //update the counter for min/max transitions
      MTransition newMinMaxTransition
         = new MTransition("mt"+mt,minOrMax);
      newMinMaxTransition.setNumberOfArguments
         (stateAppearanceInTableCounter);
      newMinMaxTransition.addInscription();
      bn.addTransition(newMinMaxTransition);
      Arc arcFromMinMaxTransToState
         =new Arc(newMinMaxTransition,
            bn.getPlace(currentState.toString()),
            "ordinary");
      arcFromMinMaxTransToState.addInscription("y");
      //y is output var. of min/max transition
      bn.addArc(arcFromMinMaxTransToState);
   } //end of if block making sure that currentState
      //has successor states
   } //end of if block making sure that currentState
      //is not a base state
   else { //currentState is a base state
      double tokenValue
         = (baseStatesWithValues.get(currentState));
      //get a reference for the place:
      Place basePlace=bn.getPlace(currentState.toString());
      basePlace.addToInitialMarking(tokenValue);
   }
   //loop through through the left column of the table
   int inputArcForMinMaxTransCounter=0;
   //counter to label the arcs x1,x2,etc.
   for(Iterator i
      =stateDecisionRewardTransformationTable.iterator();
      i.hasNext();) {
A.1 Pseudocode of the DP2PN Module

```java
DPInstanceTableEntry currentEntry = (DPInstanceTableEntry) i.next();
if (currentEntry.getState().equals(currentState)) {
    // found the current state in the left column
    ip++;
     // update counter for intermediate places
    Place intermedPlace = new IntermediatePlace("p"+ip,
        currentEntry.getDecision().toString());
    bn.addPlace(intermedPlace);
    Arc arcFromIntermedPlaceToMinMaxTrans = new Arc(intermedPlace, // bn.getPlace("p"+ip),
        bn.getTransition("mt"+mt),
        "ordinary");
    inputArcForMinMaxTransCounter++;
    // update arc counter
    arcFromIntermedPlaceToMinMaxTrans.addInscription("x"+inputArcForMinMaxTransCounter);
    bn.addArc(arcFromIntermedPlaceToMinMaxTrans);
    st++; // update counter for E(val)-transition
    ETransition evalTrans = new ETransition("st"+st,
        operatorForTransition,
        currentEntry.getReward());
    bn.addTransition(evalTrans);
    Arc arcFromETransToIntermedPlace = new Arc(evalTrans, // bn.getTransition("st"+st),
        intermedPlace, // bn.getPlace("p"+ip),
        "ordinary");
    arcFromETransToIntermedPlace.addInscription("y");
    // y is output var. of eval. transition
    bn.addArc(arcFromETransToIntermedPlace);
    // now make enabling places with a single black token
    // as initial marking to allow eval transitions to
    // only fire once
    ep++;
    // update counter for enabling places
    Place enablingPlace = new EnablingPlace("ep"+ep);
    bn.addPlace(enablingPlace);
    Arc arcFromEnablingPlaceToEvalTrans = new Arc(enablingPlace,
        evalTrans,
        "ordinary");
    bn.addArc(arcFromEnablingPlaceToEvalTrans);
```

//make placeS for the newly found stateS and enqueue
//THEM, if THEY do not already exist
StateList destinationStates = currentEntry.getNextStates();
State currentDestination;
//loop over all destination states now
int inputArcForEtransCounter=0;
//counter to label the arcs x1,x2,etc.
for(Iterator i2=destinationStates.iterator(); i2.hasNext(); ) {
currentDestination=(State) i2.next();
if(!statesBuilt.contains(currentDestination)) {
sp++;
//update counter for state places
Place statePlace =new StatePlace(currentDestination.toString());
bn.addPlace(statePlace);
statesBuilt.add(currentDestination);
//now enqueue it
stateQueue.addLast(currentDestination);
}
//make an arc (regardless whether state
//is new or not)
Arc doubleArcBetweenStateAndEtrans =new Arc(
   bn.getPlace(currentDestination.toString()),
evalTrans,
   "double");
inputArcForEtransCounter++;
doubleArcBetweenStateAndEtrans.addInscription("x"+inputArcForEtransCounter);
bn.addArc(doubleArcBetweenStateAndEtrans);
}
//At the end of the above for-loop we know how many
//arcs go into the ETransition, so we can build its
//inscription now
evalTrans.setNumberOfVariables(inputArcForEtransCounter);
evalTrans.addInscription();
}
//end of if
}
//end of for-loop over table entries
}
//end of while

return bn;
}//end of buildBellmanPNTopDown() method
A.1.6 BellmanNet Class

class BellmanNet {

    String title; //holds the title of the net
    ArrayList placeList; //a list of Place objects
    ArrayList transitionList; //a list of Transition objects
    ArrayList arcList; //a list of Arc objects

    //useful to have is an adjacency matrix. It can be calculated
    //using the method calculateAdjacencyMatrix()
    //the indices run from:
    // 0,...,|transitionList|-1,|transitionList|,...,
    // |transitionList|+|placeList|-1
    //adjacencyMatrix[x][y]=1 means that there is an arc
    //from x to y
    int[][] adjacencyMatrix;

    //An incidence matrix is used for the file I/O of Bellman nets
    // (for output via the method toIncidenceMatrix())
    //We use the instance variable incidenceMatrix to store the
    // mere numbers. It can be calculated using the method
    // calculateIncidenceMatrix()
    //Columns are labeled with transitions, rows with places
    int[][] incidenceMatrix;

    ArrayList topologicallySortedTransitionList;

    //constructor creates an initially empty BellmanNet
    BellmanNet(String title) {};

    //add a place
    void addPlace(Place p) {
        placeList.add(p);
    }

    //add a transition
    void addTransition(Transition t) {
        transitionList.add(t);
    }

    //add an arc
void addArc(Arc a) {
    arcList.add(a);
}

//get a place by index
Place getPlace(int index) {
    return placeList.get(index);
}

//get a transition by index
Transition getTransition(int index) {
    return transitionList.get(index);
}

//get a place by name
Place getPlace(String placeName) {
    //iterate through placeList
    for(Iterator i=placeList.iterator(); i.hasNext();){
        Place currentPlace=(Place) i.next();
        if (currentPlace.getName().equals(placeName)) {
            return currentPlace;
        }
    }
    return null; //not found
}

//get a transition by name
Transition getTransition(String transitionName) {
    //iterate through transitionList
    for(Iterator i=transitionList.iterator(); i.hasNext();){
        Transition currentTransition=(Transition) i.next();
        if (currentTransition.getName().equals(transitionName)) {
            return currentTransition;
        }
    }
    return null; //not found
}

//this helper method topologically sorts all the transitions
//from transitionList and puts the sorted list into
//topologicallySortedTransitionList.
void topologicallySortTransitions() {...}

void calculateAdjacencyMatrix() {...}
void calculateIncidenceMatrix() {...}

ArrayList getPresetIgnoreSelfLoopsAndEnablingPlaces(NetNode node) {...}

ArrayList getPostsetIgnoreSelfLoops(NetNode node) {...}

//Convert the BellmanNet to a comma separated //exportable format based on the incidence matrix. String toIncidenceMatrix() {
    calculateIncidenceMatrix(); //update the instance variable
    //incidenceMatrix

    String result;
    //make the first line with transition names
    for (Iterator j=transitionList.iterator(); j.hasNext();) {
        result.append(j.next().getName());
    }
    //outer loop iterates over the places
    for (Iterator i=placeList.iterator(); i.hasNext();) {
        Place currentPlace = i.next();
        if(!((currentPlace instanceof EnablingPlace)) {
            //ignore enabling places
            if(currentPlace instanceof StatePlace) {
                result.append("s,");
            }
            else { //currentPlace is intermediate place
                result.append("i,");
            }
        }
        result.append(currentPlace.getName()); //append place
        //name
        //inner loop iterates over the transitions
        for(int j=0; j<transitionList.size(); j++) {
            result.append(incidenceMatrix
                [placeList.indexOf(currentPlace)][j]);
        }
        //make the init entry for INIT/DEC column
        if(currentPlace instanceof StatePlace) {
            //if currentPlace is a base state
            if(currentPlace.getInitialMarking()!=null) {
                result.append(currentPlace.getInitialMarking());
            }
        }
        else { //currentPlace is intermediate place
            //make the DEC entry for INIT/DEC column
            if cip has a decision entry
        }
    }
}
if (currentPlace.getDecision() != null) {
    result.append(currentPlace.getDecision());
}
}
// end of if
}
// end of outer for loop over the places
// second to last line produces the transition types
for (Iterator j=transitionList.iterator(); j.hasNext();)
{
    Transition currentTransition=j.next();
    if (currentTransition instanceof MTransition) {
        result.append(currentTransition.getMinOrMaxDesignator());
    }
    else { // ETransition
        result.append(currentTransition.getArithmeticOperator());
    }
}
// last line produces the constants for ETransitions
for (Iterator j=transitionList.iterator(); j.hasNext();)
{
    Transition currentTransition=j.next();
    if (currentTransition instanceof MTransition) {
        // MTransitions do not have constants, leave blank
    }
    else { // ETransition
        result.append(currentTransition.getConstantValue());
    }
}
return result;
}

} // end of class
A.2 DP2PN System Files

The system file Main1.txt is as follows.

```java
public static void main(String[] args) throws IOException {
    //set default value for name of logfile that records details about how the Bellman net is built
    String outputFileName = "buildBNlog.txt";
    //determine the correct current working directory,
    //so that it works for
    //case 1: if this class is launched directly with "java" from within
    //this directory (and therefore currentWorkingDir==user.dir)
    //case 2: if this class is launched with
    // "java -cp thisDir className" from its parent directory
    // or via
    // "rt.exec(java -cp thisDir className)" from a class located in
    // the parent directory
    final String subDirName = "DP2PN" + problemName;
    //needs to be hardcoded, no way to find out this name here, if called from parent dir
    String currentWorkingDir = System.getProperty("user.dir");
    //now append subDirName, if launched from parent dir which is (most likely) the case if currentWorkingDir does not end in
    //subDirName
    if (!currentWorkingDir.endsWith(subDirName)) {
        currentWorkingDir = currentWorkingDir + "+" + subDirName;
    }
    //create an output directory for the Bellman net
    File newDir = new File(currentWorkingDir + "\" + problemName + "BN\");
    newDir.mkdirs(); //make the directory
    //Out writes to screen and file at the same time
    //to create log file
    Out.pw = new PrintWriter(new FileWriter(
        new File(currentWorkingDir + "\" + problemName + "BN\" + outputFileName)));
    //Out5 writes to file
    //to create the incidence matrix of the Bellman net in .CSV format
    //the file is written into the subdirectory "PN2Java"+problemName
    Out5.pw = new PrintWriter(new FileWriter(
        new File(currentWorkingDir + "\" + problemName + "BN\" + problemName + "BN.csv")));
    Out.putln("Starting...");
}
```

The system file Main2.txt is as follows.

```java
//determine the successor states of current state
StateList successorStates = calculateTransformation(currentState, d);
//determine the transition weights given the current state
ArrayList transitionWeights = calculateTransitionWeights(currentState, d);
//generate table entry
DPInstanceTableEntry entry = new DPInstanceTableEntry(
    currentState, d, calculateReward(currentState, d), successorStates, transitionWeights);
theInstance.addTableEntry(entry);
//enqueue unseen destination states by iterating over successorStates
//for TSP only one iteration in the following loop
//for MatMul two iterations
for (State currentSuccessor : successorStates) {
    if (!statesSeen.contains(currentSuccessor)) {
        ...
The system file Main3.txt is as follows.

```
beginning of the fixed code Main3.txt for Main.java

//determine the successor states of current state
StateList successorStates=calculateTransformation(currentState,d);
//determine the transition weights given the current state
ArrayList transitionWeights = calculateTransitionWeights(currentState,d);
//generate table entry
DPInstanceTableEntry entry = new DPInstanceTableEntry(currentState,d,
calculateReward(currentState,d),successorStates,transitionWeights);
theInstance.addTableEntry(entry);
//enqueue unseen destination states by iterating over successorStates
//for TSP only one iteration in the following loop
//for MatMul two iterations
for(Iterator it2=successorStates.iterator(); it2.hasNext();) {
    State currentSuccessor = (State) it2.next();
    if(!statesSeen.contains(currentSuccessor)) {
        stateQueue.addLast(currentSuccessor);
        statesSeen.add(currentSuccessor);//mark state as seen
    }
} //end of for loop over the decisions
} //end of if
else { //base state
    //do nothing
} } //end of loop once queue is empty
//print the instance
theInstance.print();
Out.println();
//build the Bellman PN from the instance
//if there are no transition weights, use the ordinary method
if(!(theInstance.hasTransitionWeights())) { 
    BellmanNet bn=theInstance.buildBellmanPNTopDown(problemName);
    Out5.println(bn.toIncidenceMatrix()); //write the BellmanNet as incidence
    //matrix.
} else { //there are transition weights, use the newly crafted method
    BellmanNet bn=theInstance.buildBellmanPNTopDownWithWeights(problemName);
    Out5.println(bn.toIncidenceMatrix()); //write the BellmanNet as incidence
    //matrix.
}

//finish the output files
Out.println("End.");
Out.close(); //close the output stream (the log file)
Out5.close(); //close the 5th output stream (the BellmanNet
//incidence matrix in .CSV)
} //end of main() method

end of the fixed code Main3.txt for Main.java
```

A Supplementary Material

```
stateQueue.addLast(currentSuccessor);
stateSeen.add(currentSuccessor);//mark state as seen
}
) }
} //end of for loop over the decisions
} //end of if
else { //base state
    //do nothing
} } //end of loop once queue is empty
//print the instance
theInstance.print();
Out.println();
//build the Bellman PN from the instance
//if there are no transition weights, use the ordinary method
if(!(theInstance.hasTransitionWeights())) { 
    BellmanNet bn=theInstance.buildBellmanPNTopDown(problemName);
    Out5.println(bn.toIncidenceMatrix()); //write the BellmanNet as incidence
    //matrix.
} else { //there are transition weights, use the newly crafted method
    BellmanNet bn=theInstance.buildBellmanPNTopDownWithWeights(problemName);
    Out5.println(bn.toIncidenceMatrix()); //write the BellmanNet as incidence
    //matrix.
}
```

The system file Main3.txt is as follows.
else { //there are transition weights, use the newly crafted method
    BellmanNet bn=theInstance.buildBellmanPNTopDownWithWeights(problemName);
    Out5.println(bn.toIncidenceMatrix()); //write the BellmanNet as incidence matrix.
}

//finish the output files
Out.println("End.");
Out.close(); //close the output stream (the log file)
Out5.close(); //close the 5th output stream (the BellmanNet incidence matrix in .CSV)
}

} //end of main() method

 ViewChild="Main";

//end of the fixed code Main.txt for Main.java

//end of the fixed code Main3.txt for Main.java
A.3 Output from PN2XML

In this section, the XML output from the PN2XML module is shown. In the form presented, such files (with the suffix .xrn) can be imported into the PN software system Renew version 1.6 [32]. Renew has the capabilities to display PNs (and thus the Bellman nets produced) graphically and also to simulate PNs. Unfortunately, the graphical layout is not optimized and looks rather crude for large PNs. Very often however, Renew’s automatic net layout feature (when choosing the Automatic Net Layout option of the Layout menu) rearranges the nodes of a PN nicely, especially for smaller nets. For space reasons, we include only the XML file for SPA1 (see Sect. 9.4.2).

A.3.1 High-Level Bellman Net XML file for SPA1

```xml
<?xml version="1.0"?>
<!DOCTYPE net SYSTEM "http://www.renew.de/xrn1.dtd">
<net id="N" type="hlnet">
  <place id="I1">
    <graphics>
      <size w="20" h="20"/>
      <offset x="30" y="150"/>
      <fillcolor><RGBcolor r="112" g="219" b="147"/></fillcolor>
      <pencolor><RGBcolor r="0" g="0" b="0"/></pencolor>
      <textcolor><RGBcolor r="0" g="0" b="0"/></textcolor>
    </graphics>
    <annotation id="I2" type="name">
      <text>(0)</text>
      <graphics>
        <size w="20" h="16"/>
        <textsize size="12"/>
        <offset x="0" y="15"/>
        <fillcolor><transparent/></fillcolor>
        <pencolor><transparent/></pencolor>
        <textcolor><RGBcolor r="0" g="0" b="0"/></textcolor>
      </graphics>
    </annotation>
  </place>
  <place id="I3">
    <graphics>
      <size w="20" h="20"/>
      <offset x="30" y="240"/>
      <fillcolor><RGBcolor r="112" g="219" b="147"/></fillcolor>
      <pencolor><RGBcolor r="0" g="0" b="0"/></pencolor>
      <textcolor><RGBcolor r="0" g="0" b="0"/></textcolor>
    </graphics>
    <annotation id="I4" type="name">
      <text>p1</text>
      <graphics>
        <size w="20" h="16"/>
        <textsize size="12"/>
        <offset x="0" y="15"/>
        <fillcolor><transparent/></fillcolor>
        <pencolor><transparent/></pencolor>
        <textcolor><RGBcolor r="0" g="0" b="0"/></textcolor>
      </graphics>
    </annotation>
  </place>
  <place id="I5">
    <graphics>
      <size w="20" h="20"/>
      <offset x="30" y="330"/>
      <fillcolor><RGBcolor r="112" g="219" b="147"/></fillcolor>
      <pencolor><RGBcolor r="0" g="0" b="0"/></pencolor>
      <textcolor><RGBcolor r="0" g="0" b="0"/></textcolor>
    </graphics>
    <annotation id="I6" type="name">
      <text>(1)</text>
      <graphics>
        <size w="20" h="16"/>
        <textsize size="12"/>
        <offset x="0" y="15"/>
        <fillcolor><transparent/></fillcolor>
        <pencolor><transparent/></pencolor>
        <textcolor><RGBcolor r="0" g="0" b="0"/></textcolor>
      </graphics>
    </annotation>
  </place>
</net>
```
B.1 System Requirements for DP2PN2Solver

The DP2PN2Solver tool should run on any computer system on which a java compiler and runtime systems (javac and java) is installed. (We have only tested the tool using JDK 1.4.2 and JDK 1.5.) To obtain numerical solutions in a spreadsheet or graphically, a spreadsheet system (such as Excel) or a Petri net tool (such as Renew) is also required.

B.1.1 Java Environment

Make sure that JDK 1.4.2 or 1.5 is installed on your system. (DP2PN2Solver might also work with earlier and later versions, but this has not been tested.) You may download it for free from http://www.java.sun.com

DP2PN2Solver needs the compiler “javac”, so having merely a Java Runtime Environment (JRE) installed will not be sufficient. The compiler “javac” should be universally accessible, so make sure to include the directory where the binary of “javac” resides to your current path.

For example, on Windows XP you might add the following to your path by going through the sequence “Start—Control Panel—System—Advanced—Environment Variables” and then under “System variables” (or “User variables” if you are not the admin), edit the variable “path” by prepending the prefix c:\j2sdk1.4.2_01\bin; to the path. (We assumed that the binary javac.exe is located in c:\j2sdk1.4.2_01\bin.)

You can easily check whether the compiler “javac” is universally accessible by typing javac on the command line from any directory of your choice; if you always get a response that explains the usage of “javac” then the path is set correctly; if you get a response that “javac” is not recognized as a command, then the path is not set correctly.
B.2 Obtaining DP2PN2Solver

The DP2PN2Solver software can be downloaded from the following websites:
http://natsci.eckerd.edu/~mauchh/Research/DP2PN2Solver
http://www2.hawaii.edu/~icl/DP2PN2Solver

free of charge. Version 6 of the software comes as a zipped file named DP2NP2SolverV6.zip. (Over time, the version numbers may have progressed.)

When unzipped, the folder will contain:

1. four folders (named bellman_net, BN2SolverV6, DP2PNv6, and pn2java) that contain the software and sample programs,
2. a read me file that contains special notices (regarding rights, disclaimers, etc.), including a reference to the book, and
3. a UserGuide that provides usage and reference information, explaining how to install and use the DP2PN2Solver tool and giving some implementation details.

The UserGuide includes instructions for compiling gDPS source programs into Bellman nets and for obtaining numerical solutions from these Bellman nets. Suggestions for debugging gDPS programs using the tool, as well as a list of diagnostic error messages, are also included.

(Note: This Appendix duplicates what is in the UserGuide.)

B.3 Installation of DP2PN2Solver

This section describes how and where to deploy the files to your computer.

B.3.1 Deployment of the Files

Take the downloaded ZIP file and unzip it into a directory of your choice (the installation directory). Make sure that when you unzip and extract, you preserve the pathname information for the files (check your zip utility for that) — what you definitely do not want to happen is that after unzipping all files are in a single flat directory.
Throughout these instructions let us assume that the installation directory that contains the software is named DP2PN2Solver. (You may pick another name if you wish.) In Windows systems, the software might have been installed to C:\Program Files\DP2PN2Solver, for example. Note that this manual also applies to UNIX-like systems; the DOS/Windows specific separation character '\\' (backslash) will have to be read as '/\' (slash) instead, and the DOS/Windows typical path separation character ';\' (semicolon) has to be read as '::' (colon).

After the unzip is completed the directory DP2PN2Solver contains the following four subdirectories

bellman_net
BN2SolverV5
DP2BNv5
pn2java

The directories bellman_net and pn2java are Java packages that contain parts of the implementation. While they might be of interest to developers, they need not concern ordinary users of DP2PN2Solver.

The directory DP2BNv5 contains the software for the module DP2PN; this is also the directory where you have to place your gDPS source files.

The directory BN2SolverV5 contains the software for the module PN2Solver; this is also the directory where you have to place your Bellman net source file, which you probably created using the DP2PN module.

B.4 Running DP2PN2Solver

This section describes how to invoke DP2PN2Solver’s two major modules DP2PN and PN2Solver.

B.4.1 The DP2PN Module

Preparations

Switch to the directory

DP2PN2Solver\DP2BNv5

Make sure you have your gDPS source file ready to be parsed in this directory. Remember, the gDPS source file is the one that contains all the information of your DP problem instance. Usually it needs to be created manually by the DP modeler. (With the exception of integer linear programming problems, for which a preprocessor exists.) You will notice that there are already some example gDPS source files present in this directory, ending with the suffix ".dp", e.g.
act.dp
...
 tspAlt.dp

For details on how to create a gDPS source file, please refer to Chap. 3 and Chap. 4.

The following other files in this directory (ending in .class and in .txt) make up the DP parser and should not be modified:

Main1.txt
Main2.txt
Main3.txt
State.txt
CodeCreator.class
DPFEdata.class
DPspecificationParser$1.class
DPspecificationParser$JCalls.class
DPspecificationParser$LookaheadSuccess.class
DPspecificationParser.class
DPspecificationParserConstants.class
DPspecificationParserTokenManager.class
FileCopy.class
Out.class
Out2.class
Out3.class
ParseException.class
SimpleCharStream.class
StreamGobbler.class
SymbolTableEntry.class
Token.class
TokenMgrError.class

Invocation of DP2PN

The gDPS2BN parser is invoked as follows, where the name of the gDPS source specification file (e.g. mcm.dp) is provided as a command line argument; optionally, a name for the parser log file can be specified.

java DPspecificationParser mcm.dp [parserLogFileName]

As an alternative, on DOS/Windows systems the batch file DP2BN.bat can simplify the invocation:

DP2BN mcm.dp [parserLogFileName]
Consequences of Invocation

This will create a directory for your problem instance, named DP2PNXXX where XXX is the name given in the “NAME” section of the gDPS specification, e.g. BST, MCM, etc. The contents of this new directory are explained below; we will from now on assume that the name of the problem instance is MCM.

Note that the name of the gDPS source specification file does not necessarily need to match the name given in the “NAME” section of this gDPS specification (e.g. MCM), but it seems good practice to match them anyway.

The parser log file is stored by default in

\[\text{DP2PN2Solver}\backslash\text{DP2BNv5}\backslash\text{dpParserLog.txt}\]

but the log file name can be changed to something else by specifying \([\text{parserLogFileName}]\) in the invocation launch.

Now look at the directory

\[\text{DP2PN2Solver}\backslash\text{DP2BNv5}\backslash\text{DP2PNMCM}\]

which contains some intermediate data files that might be useful for debugging, if you encountered a problem parsing or compiling your gDPS source file. Otherwise they are of no further interest to the normal user. If the gDPS source is parsed, compiled, and successfully translated into a Bellman net, then the subdirectory \(\text{MCMBN} (\text{XXXBN} \text{ in general})\) will hold the output, i.e. the Bellman net, and a log file of the Bellman net building process in:

\[\text{DP2PN2Solver}\backslash\text{DP2BNv5}\backslash\text{DP2PNMCM}\backslash\text{MCMBN}\]

\[\text{MCMBN.csv} \quad \text{DP2PNlog.txt}\]

The first file \(\text{MCMBN.csv}\) is the desired Bellman net, which can be fed into the PN2Solver software module.

This concludes the use of the DP2PN module. Starting from a gDPS source, we have produced a Bellman net. The next section deals with the use of the PN2Solver module, that automatically produces solver code from a Bellman net.

B.4.2 The PN2Solver Module

Preparations

The output of DP2PN can and should be used as the input for PN2Solver, so now manually copy or move the Bellman net you just produced (e.g. \(\text{MCMBN.csv}\)) to the directory

\[\text{DP2PN2Solver}\backslash\text{BN2SolverV5}\]
which already contains twelve .class files that make up the Bellman net parser. So before you invoke the Bellman net parser, the directory will look like this:

DP2PN2Solver\BN2SolverV5
  BNspecificationParser.class
  BNspecificationParserConstants.class
  BNspecificationParserTokenManager.class
  Out.class
  Out2.class
  Out3.class
  Out4.class
  Out6.class
  ParseException.class
  SimpleCharStream.class
  Token.class
  TokenMgrError.class
  MCMBN.csv

In our distribution, there may already be additional sample Bellman net files in the directory, all having a name of the form XXXBN.csv.

Invocation of PN2Solver

After establishing the Bellman net file MCMBN.csv in DP2PN2Solver\BN2SolverV5 and changing to this directory we are ready to invoke the BN2Solver module; we provide the name of the Bellman net file MCMBN.csv as a command line argument as follows (do not omit the classpath information, otherwise you will get an error message):

java -classpath ;... BNspecificationParser MCMBN.csv

As an alternative, on DOS/Windows systems the batch file BN2Solver.bat can simplify the invocation:

BN2Solver MCMBN.csv

Consequences of Invocation

This produces a directory named MCMSolverCode that contains three solvers where each solver uses a different technology to produce a final result. In DP2PN2Solver\BN2SolverV5\MCMSolverCode the first solver is

MCM.csv
B.5 Creation of the gDPS Source File

This is the spreadsheet solver; load this file into a spreadsheet application such as Microsoft Excel, update the cells, and you get the solution to the DP problem instance.

The second solver is

MCM.xrn

This is the Petri net solver; import this file (which is in PNML standard format) into a PN application such as Renew, simulate the net, and you get the solution to the DP problem instance.

The third solver is the Java solver

MCMJavaSolver.java

The Java Solver file is automatically compiled to

MCMJavaSolver.class

and executed by the invocation from Section B.4.2 and the resulting solution tree of the problem instance can be found in the file

MCMSolutionTree.txt

This file is the desired output providing not only the optimal function value but also the optimal decision policy for the DP problem instance.

The file

PN2SolverLog.txt

contains a log of the transformation process from a Bellman net to the solver files.

If for some reason you would like to trace the automated compilation and execution of the JavaSolver, see the file

runJavaSolver.bat

which contains the necessary steps to compile and launch the JavaSolver:

javac -classpath ..\.. MCMJavaSolver.java
rem for running, do not forget to include the current directory
java -classpath ..\..\.. MCMJavaSolver

B.5 Creation of the gDPS Source File

A gDPS source file is a plain text file ending in the suffix .dp. It can be created and modified with a simple text editor. Details on how to create a gDPS source files are given in Chapter 3. Numerous gDPS examples are shown in Chapter 4. We refer the reader to these chapters.
B.6 Debugging gDPS Code

B.6.1 Omission of Base Cases

Suppose in INTKSSCA, instead of correctly specifying the base cases as

\[
\begin{align*}
\text{DPFE\_BASE\_CONDITIONS:} & \\
& f(k,s) = 0.0 \text{ WHEN } (s==0); \\
& f(k,s) = 99990.0 \text{ WHEN } ((k==M1)\&\&(s>0));
\end{align*}
\]

we accidentally omit the base cases (2,0) and (3,0) by specifying

\[
\begin{align*}
\text{DPFE\_BASE\_CONDITIONS:} & \\
& f(k,s) = 0.0 \text{ WHEN } ((k==M1)\&\&(s==0)); \\
& f(k,s) = 99990.0 \text{ WHEN } ((k==M1)\&\&(s>0));
\end{align*}
\]

then there is no feasible decision to take from the states (2,0) and (3,0) and hence the recursive process terminates, yet (2,0) and (3,0) are not declared as base states.

The DP2PN module will not report an error in this case, and produce a PN. (This is not a bug of the DP2PN module, which has performed its job of translating the gDPS source into a PN — the error becomes apparent after performing consistency checks on the PN.) In the resulting PN, the states (2,0) and (3,0) are source places without a proper initial marking.

One way to detect this, would be to examine the intermediate PN produced by the DP2PN module.

Another way of getting valuable hints for debugging is to examine the file buildBNlog.txt which is located in the same directory as the resulting Bellman net. The file contains the base states and their initial values and also the state-decision-reward-transformation-table, which shows the states (2,0) and (3,0) appearing as successor states in the transformation column, but not as states from which transformations originate (i.e. they do not appear in the state-column). It also shows that the states (2,0) and (3,0) are not among the base states.

If it is attempted to invoke BN2Solver upon the flawed PN, for the Java solver we get a bunch of error messages, e.g.

```
errStream>intKSscaSolverCode\intKSscaJavaSolver.java:44: cannot resolve symbol
errStream>symbol : variable I41
errStream>location: class intKSscaJavaSolver
errStream>operandList.add(I41);
errStream>^
errStream>intKSscaSolverCode\intKSscaJavaSolver.java:47: cannot resolve symbol
errStream>symbol : variable I41
errStream>location: class intKSscaJavaSolver
errStream> I41.getValue()+210.0,
errStream>^
```
These error messages do not provide a good starting point for debugging. The debugging process should be initiated one step earlier, as mentioned above.

B.6.2 Common Mistakes

Space Before Minus

The DP2PN module’s lexicographical analysis scans negative numbers as tokens, which sometimes leads to an ambiguity when subtraction expressions are desired and there is no whitespace between the minus operator and the expression to be subtracted. In this case an error is reported. For example,

\[
\text{TRANSFORMATION\_FUNCTION: } t1(m, \text{dummy}) = (m-1);
\]

causes an error that looks something like:

Exception in thread "main" ParseException:
  Encountered "-1" at line 29, column 32.
  Was expecting one of:
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    

Avoid this error by adding a space after the minus sign, as in

\[
\text{TRANSFORMATION\_FUNCTION: } t1(m, \text{dummy}) = (m - 1);
\]

and now it works.

Forgotten Braces in Singleton

If a set expression involves a singleton, it is easy to forget the curly braces around the single element. But those are necessary to correctly identify the expression as a set. For example, with the integer decision variable \(d\),

\[
\begin{align*}
\text{TRANSFORMATION\_FUNCTION:} \\
tLeft(k,S,i,j,d) & = (k+1, S \text{ SETMINUS d, i, begintime}[d]); \\
tRight(k,S,i,j,d) & = (k+1, S \text{ SETMINUS d, endtime}[d], j );
\end{align*}
\]
causes the following error when attempting to construct the Bellman net:

```
errStream>DP2PNact2\act2Main.java:239: cannot resolve symbol
errStream>symbol : constructor NodeSet (Decision)
errStream>location: class NodeSet
errStream>  NodeSet _globalSet2=new NodeSet(d);
errStream> ^
errStream>DP2PNact2\act2Main.java:246: cannot resolve symbol
errStream>symbol : constructor NodeSet (Decision)
errStream>location: class NodeSet
errStream>  NodeSet _globalSet3=new NodeSet(d);
errStream> ^
errStream> [total 1502ms]
errStream>2 errors
```

The error is not caught by the syntax parser, because if \( d \) were a set variable, syntactically there would be nothing wrong. The correct formulation would be:

```
TRANSFORMATION_FUNCTION:
  tLeft(k,S,i,j,d) = (k+1, S SETMINUS {d}, i, begintime[d]);
  tRight(k,S,i,j,d) = (k+1, S SETMINUS {d}, endtime[d], j );
```

### B.7 Error Messages of DP2PN2Solver

In addition to obvious syntax errors, the following error messages are reported by the DP2PN module.

- **Illegal type in state section.** This error is reported if a component of the state is not of type \( \text{int} \) or \( \text{Set} \).
- **Illegal type of decisionSetGlobalFunctionalArgument.** This error is reported if a variable used in the decision set is not of type \( \text{int} \) or \( \text{Set} \).
- **A DPFE_BASE_CONDITION functional does not match the one declared in GOAL section.** This error is reported if there is a mismatch of the functional name used in the goal statement and the one used in a base condition.
- **Base section functional does not match the one declared in GOAL section.** This error is reported if there is a mismatch of the functional name used in the goal statement and the one used in a base statement.
- **DPFE functional does not match the one declared in GOAL section.** This error is reported if there is a mismatch of the functional name used in the goal statement and the one used in the DPFE.
- **Decision variable in DPFE does not match the one declared after DECISION_VARIABLE.** This error is reported if there is a mismatch of the decision variable name declared in the decision variable section and the one used in the DPFE.
• *Decision set identifier in DPFE does not match the one declared after DECISION_SPACE.* This error is reported if there is a mismatch of the decision set identifier used in the decision space section and the one used in the DPFE.

• *More than one reward functional in DPFE.* The current version of DP2PN2Solver requires exactly one reward functional to present in the DPFE. This error is reported if there is more than one reward functional.

• *Recursive functional mismatch in DPFE.* There must be exactly one name that is used as the functional in the recurrence relation. This error is reported if there is a mismatch of the functional names used within the DPFE.

• *In REWARD_FUNCTION section, functional identifier does not match the one in DPFE.* This error is reported if there is a mismatch of the functional name used for the reward function in the DPFE and the one used in the reward function section.

• *Illegal type of rewardFunctionGlobalFunctionalArgument.* This error is reported if a variable used in the reward function section is not of type int or Set.

• *In TRANSFORMATION_FUNCTION a functional appears that is not present in DPFE.* This error is reported if a functional name is used in the transformation function section, but not in the DPFE.

• *Illegal type of transformationFunctionGlobalFunctionalArgument.* This error is reported if a variable used in the transformation function section is not of type int or Set.

• *Illegal type of transformationFunctionSetGlobalFunctionalArgument.* This error is reported if a variable used in the transformation function section is a state coordinate of illegal type.

• *In TRANSITION_WEIGHTS a functional appears that is not present in DPFE.* All weight functions defined in the transition weight section must be used in the DPFE.
References


activated transition, 206, 208
Activity Selection Problem, 64
acyclic Petri net, 209
add-in, 330
all-pairs shortest paths, 29, 50
ALLOT, 49
allotment, 47, 80
Alphabetic Radix-Code Tree Problem, 51
AMPL, 4
annotation, 207, 285
APSP, 50
APSPFW, 50
ARC, 51
arc, 9
artificial intelligence, 78
ASMBAL, 52
Assembly Line Balancing, 52
ASSIGN, 54
assignment problem, 46, 47, 54
attribute, 207, 285
B-transition, 216
Backus-Naur form, 117
base case, 9
base condition, 12
base state, 9
base state place, 215
Bellman, 5
Bellman net, 4, 205, 211, 247
  high-level, 215, 251, 288
  low-level, 212
Bellman-Ford algorithm, 28, 30
big-O, 10
binary search tree, 55
binary tree, 47, 94
bioinformatics, 48
black token, 216
BNF, 117
boundary condition, 9
bounded Petri net, 209
branch, 9
BST, 55
buildBNlog.txt, 252
builder module, 259
Calculation Object, 255
characteristic vector, 78
clustering problem, 46
code generation, 38
color, 208
colored Petri net, 207, 208
comma separated values, 252
comment, in gDPS, 106
complexity, 22, 31
covergence, 22
cost, 11
COV, 57
COV as KSINT Problem, 70
covering, 57, 70
covering problem, 46
CPN, 208
CSV, 252
curse of dimensionality, 31
curve fitting, 88
dataflow architecture, 335
Interval Scheduling Problem, 64
intractable, 31
intractable problem, 99, 334
INTVL, 64
INVENT, 66
inventory, 66, 83
Inventory Problem, 66
inventory problem, 47
inverted linear search, 14
INVEST, 67
investment, 67
investment problem, 47
INVESTWLV, 68
Java, 103, 106
Jug-Pouring Problem, 82
k-bounded Petri net, 209
knapsack, 46
Knapsack Problem, 69
KS01, 69
KSCOV, 70
KSINT, 49, 70
label, 285
LCS, 71
least squares, 89
linear programming, 3, 46, 47
linear search, 12, 14, 81
LINGO, 4
LINSRC, 73
live Petri net, 209
live transition, 209
LLBN, 212
Longest Common Subsequence, 71
Longest Simple Path, 74
LOT, 73
Lot Size Problem, 73
low-level Bellman net, 212
low-level Petri net, 206
LSP, 74
M-transition, 217
marking, 206
reachable, 209
markup language, 207
matching problem, 54
Matrix Chain Multiplication, 75
maximal link, 75
MCM, 75
Method S, 12, 13
Method W, 13
Minimum Maximum Problem, 75
Minimum Weight Spanning Tree Problem, 77
MINMAX, 75, 79
multiplicative DP, 80
multistage decision process, 69, 77
multistage graph, 86
MWST, 77
net, 206
next-state, 9, 11, 115, 205, 261, 275
NIM, 78
node, 9
NodeSet, 108
nonoptimization, 48, 62
nonoptimization problems, 33
nonserial, 65
nonserial DPFE, 9, 47
O (big-O), 10
objective function, 6, 11
occur, transition, 209
ODP, 49, 80
operator, 11
Optimal Allotment Problem, 49
Optimal Assignment Problem, 54
Optimal Binary Search Tree, 55
Optimal Covering Problem, 57
Optimal Distribution Problem, 80
Optimal Investment Problem, 67
Optimal Linear Search Problem, 12, 73
Optimal Permutation Problem, 81
Optimal Production Problem, 83
Optimal Selection Problem, 94
optimal substructure property, 5, 74
optimization problem, 6
optimization, 6
order, of DPFE, 9
pagination, 92
parser module, 259
partitioning problem, 46
path state, 21
path-state, 75
PERM, 81
permutation, 13, 45, 54, 81
persistent Petri net, 209
Petri net, 4, 5, 37, 205
colored, 207, 208
high-level, 207
low-level, 206
markup language, 207
place/transition net, 206
predicate/transition net, 207, 208
tool, 210
place, 206
place/transition net, 206
PN2Java, 247
PN2Spreadsheet, 247
PN2XML, 247
PNML, 207
policy, 10
postset, 206
POUR, 82
pouring, 82
predicate/transition net, 207, 208
preset, 206
Principle of Optimality, 5
priority algorithm, 32
probabilistic DP, 32, 47, 50, 68, 79, 83, 84
Process Scheduling Problem, 97
PROD, 83
PRODRAP, 84
production, 68, 83, 84
production problem, 47
profit, 11
Program Segmentation Problem, 91
proper DPFE, 34, 329, 334
proper LLBN, 214
PrTnN, 208
radix code tree, 51
rank, of partition, 46
RDP, 84
reachable marking, 209
reconstruction, 10, 17
recurrence relation, 22, 34, 48, 60, 62
recursion, 36
recursive functional equation, 8
regression, 89
Reject Allowances Problem, 84
relaxation, 22, 29
reliability, 84
Reliability Design Problem, 84
Renew, 210, 324
REPLACE, 85
replacement, 85
Replacement Problem, 85
replacement problem, 47
reverse arc, 216
reward function, 11, 115
routing, 45
safe Petri net, 209
scheduling, 46, 57, 61, 64, 87, 97
scheduling problem, 46
SCP, 86
SDRT, 36
second-order DPFE, 47
SEEK, 87
Seek Disk Scheduling Problem, 87
SEGLINE, 88
Segmented Curve Fitting, 88
SEGPAGE, 91
SELECT, 94
selection problem, 46
self-loop, in graph, 25
self-loop, in Petri net, 216
separable function, strongly, 7
separable function, weakly, 8
sequencing, 45
sequencing problem, 46
set notation, in gDPS, 108
set selection problem, 94
shortest path, 17
shortest path problem, 23
Shortest Path, Acyclic Graph, 95
Shortest Path, Cyclic Graph, 95
shortest processing time, 15
simplex, 3, 330
solution tree, 277, 293, 322
solver code, 38, 42
SPA, 24, 45, 95
spanning tree, 46, 77
SPC, 95
fixed time, 27
relaxation, 28
successive approximation, 25
spreadsheet, 38, 281, 321, 335
SPT, 15, 97
stage, 19
Stagecoach Problem, 86
state, 11
state place, 215
state space, 11
state transition system, 9
State-Decision-Reward-Transformation Table, 36
string processing, 48
strongly separable function, 7
successive approximation, 22

tableaux, 3
tape storage, 82
tape storage problem, 15
target-state DPFE, 10
temporal, 58
terminal state, 9, 11
text processing, 48
token, 206
tool, Petri net, 210
tower, 62
Tower of Hanoi, 33
Tower of Hanoi Problem, 62
transformation function, 11, 115
transition, 206
activated, 206, 208
enabled, 206, 208
fire, 207, 209
occur, 209
transition function, 11, 115
transition weights, 66
TRANSP, 98
transportation, 98
Transportation Problem, 98
transportation problem, 47
Traveling Salesman Problem, 29, 31, 99
tree
binary, 47, 94, 222
binary search, 55
Huffman code, 51
K-ary, 211
radix code, 51
spanning, 46, 77
TreeSet, 108
TSP, 99

user guide, 359
vertex, 9
virtual time, 19, 20

weakly separable function, 8
weights, 48
weights, of transition, 12
Winning in Las Vegas, 68

XML, 207, 324
XRN, 288, 324