Chondro Documentation

Release 0.2.3

Przemyslaw Szufel
Bogumił Kamiński
Michał Jakubczyk

November 21, 2016

CHAPTER 1

Decision tree sensitivity analysis with Chondro

1.1 Library overview

The goal of this document is to describe the Chobdro library that accompanies our paper titled *"A unified framework for global sensitivity analysis of decision trees"*.

Chondro – is an analytical engine that implements the decision tree (DT) sensitivity analysis (SA) algorithms described in the above article. All the methods support both for separable and non-separable decision trees. Chondro has been developed with the Python3 and has been tested with Anaconda 2.3.0 running Python3 version 3.4.4.

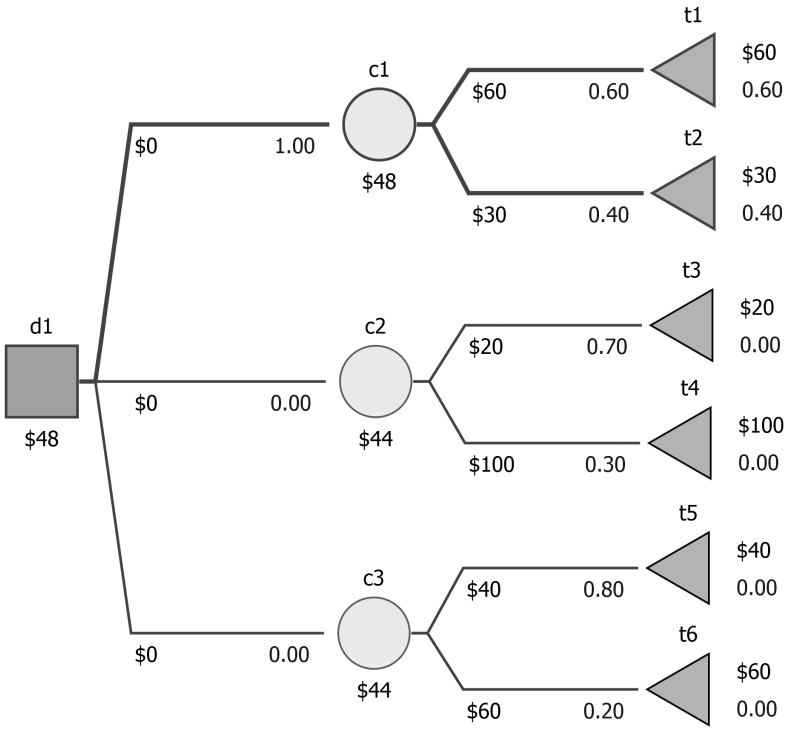
The basic Chondro's functionality is described in Table 1.1. The software can load files stored by the SilverDecisions (available at `http://www.silverdecisions.pl`) or uses internal JSON format. A DT is presented as a Python dictionary structure with each node described with a `type` (choice,decision,final), `id`, `value` (pay-off), and a list (`nodes`) containing child nodes. The probability values `p` (for separable DTs) or identifiers `pi` (for non-separable DTs) are stored in children nodes of a chance node – full JSON schema specification for DT representation is available at section. 2.2. For a quick overview please see a sample DT insee Figure 1.1. Chondro supports non separable DTs through injection of probability values as a dictionary. In order to perform stability and perturbation analysis for non-separable decision trees a function that generates probability dictionary on the base of fundamental probabilities should be provided to a respective algorithm. An example of such function has been presented in Listing 1.3.

It should be noted that Chondro heavily relies on the Python `fractions` package for numerical computing and hence enables calculation and comparison of the exact values for P-optimal decisions. In this way we managed to avoid numerical problems when expected values at different nodes are equal.

1.2 Quick start - sensitivity analysis of separable trees

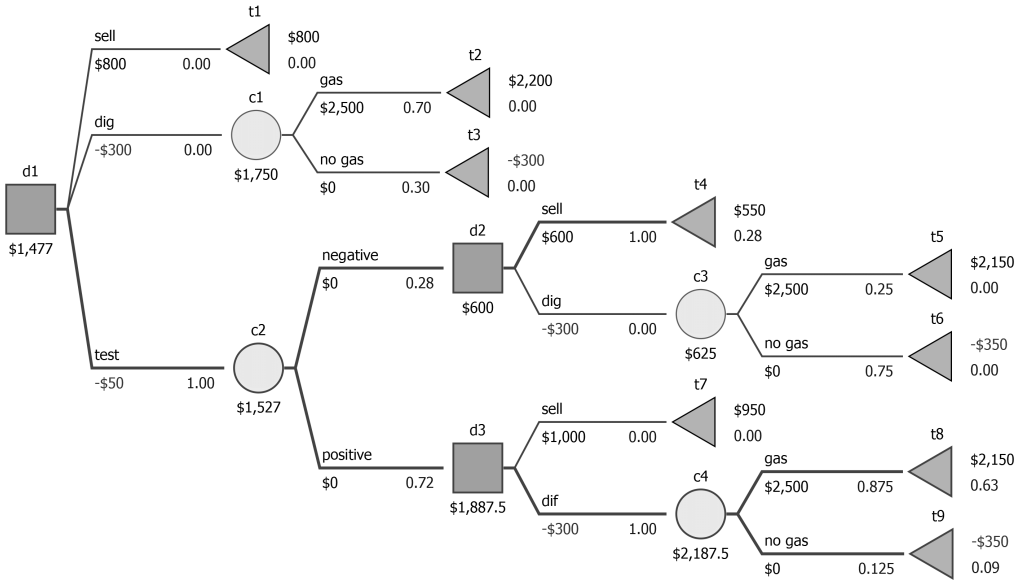
A typical example session with Chondro might consist of the following steps:

1. create a JSON representation of a DT (either by saving a DT from SilverDecisions or manually creating a JSON file)
2. Use the `load_tree` function to load a DT to memory
3. Use the `solve_tree` function to calculate optimal decision for the DT. The function supports non-separable trees by accepting a probability dictionary.
4. perform the sensitivity analysis
    - use `find_stability` to calculate the stability
    - use `find_perturbation_mode` to calculate $P_{mode,\varepsilon}$
    - use `find_perturbation_pessopty` to calculate $P_{\min,\varepsilon}$ and $P_{\max,\varepsilon}$




```
{
  "tree": {
    "type":"decision", "id":"d1",
    "nodes": [
      {
        "type":"chance","id":"c1",
        "nodes": [
          {"p":"0.6","type":"final","id":"t1","value": "60" },
          {"p":"0.4","type":"final","id":"t2","value": "30" }
        ]
      },
      {
        "type":"chance","id":"c2",
        "nodes": [
          {"p":"0.7","type":"final","id":"t3","value": "20" },
          {"p":"0.3","type":"final","id":"t4","value": "100" }
        ]
      },
      {
        "type":"chance","id":"c3",
        "nodes": [
          {"p":"0.8","type":"final","id":"t5","value": "40" },
          {"p":"0.2","type":"final","id":"t6","value": "60" }
        ]
      }
    ]
  }
}
```

Figure 1.1: A separable decision tree and a corresponding JSON representation. The probability and payoff values are given as string rather than number values in order to enable a proper conversion with the `fractions` module. The full JSON schema decision tree specification see Listing 2.1.




```
{
    "tree": {
        "type": "decision", "id": "d1",
        "nodes": [
            {
                "type": "final","label":"sell","id": "t1","value": "800"
            },
            {
                "type": "chance","label":"dig","id": "c1","value": "−300",
                "nodes": [
                    {
                        "pi": "gas", "label":"gas",
                        "type": "final", "id": "t2", "value": "2500"
                    },
                    {
                        "pi": "no_gas", "label":"no_gas",
                        "type": "final", "id": "t3", "value": "0"
                    }
                ]
            },
            {
                "type": "chance","label":"test","id": "c2","value": "−50",
                "nodes": [
                    {
                        "pi": "neg._test", "label":"negative",
                        "type": "decision", "id": "d2",
                        "nodes": [
                            {
                                "type": "final",
                                "label":"sell",
                                "id": "t4",
                                "value": "600"
                            },
(...)
```

Figure 1.2: A non-separable decision tree and a part of the corresponding JSON representation.

function	description	output
load_tree	loads a DT from SilverDecisions or internal *.json file	a Python dictionary holding a decision tree (see Listing 2.1 for full DT representation specification)
save_tree	saves a DT to a JSON file	file saved to disk follows format presented in Listing 2.1
solve_tree	recursively calculates an optimal value for a given separable or non-separable DT	expected value for a tree and P-optimal decisions for every node
print_tree	prints a tree to the standard output	textual representation of a DT – e.g. see 1.2
find_stability	finds stability for a separable or non-separable DT	stability coefficient for every optimal decision in a given DT
find_perturbation_mode	performs a sweep over ε values in order to find epsilon ranges for P_{mode} perturbation	dictionary of ε ranges and P-optimal decisions for perturbations
find_perturbation_pessopty	performs a sweep over the full grid (non-separable DT) or ε (separable DT) values and finds epsilon ranges for $P_{\max,\varepsilon}$ and $P_{\min,\varepsilon}$ perturbations	dictionary of ε ranges and P-optimal decisions for perturbations

Table 1.1: Functionality of the Chondro decision tree sensitivity analysis module

Listing 1.1 presents a sample code to solve the decision tree. We first start by loading the module (line 1). Next a JSON file is loaded with the load_tree function. It should be noted that this function supports JSON files in both internal dictionary format as well as files that can be exported from SilverDecisions software (available at http://www.silverdecisions.pl). The function solve_tree (line 5) returns a tuple where the first element is the expected value of DT and the second dictionary of optimal decisions.

It should be noted that in Chondro a decision tree is represented as a python dictionary - a direct representation of the JSON file presented in the Figure 1.1. We assume that each node in a tree can be identified by a path represented as a Python tuple of indices. The root node is represented by () (an empty tuple) the first node (c1 in Listing 1.2) is represented by (0,) (one element tuple) while the node t3 could be presented as (1,0). Node paths are used to for represent P-optimal decisions. Further in the text the tuple of node indices representing a path in the graph will be called node_path_tuple. It should be noted that all sensitivity analysis methods (find_stability, find_perturbation_mode and find_perturbation_pessopty) provide a use_labels parameter. If it is set to True (the default value) Chondro uses label attribute (see section 2.2) of decision tree nodes rather than indices to represent paths in the tree. For example consider a DT in Figure 1.2. The P-optimal decision can be presented a tuple of two node_path_tuples either as ((2, 0, 0), (2, 1, 1)) or (("test", "negative", "sell"), ("test", "positive", "dig")). The P-optimal decision consists of two elements because the test performed at c2 node can have two results - can be either positive or negative. Further in the text a tuple of node_path_tuples will be called *decision_path_tuple*.

The function solve_tree returns the optimal decision paths in the form of a following dictionary:

```
{ node_path_tuple :  [list of P-optimal node indices] }
```

It should be noted that the function solve_tree changes the state of the tree object – now additionally it stores the data on optimal solution. The function print_tree enables printing a human-readable textual representation of a DT to the console (line 9). A sample textual tree representation can be found in Listing 1.2.

Once a decision tree is loaded and solved a sensitivity analysis (SA) can be performed. The first step is calculating the stability value and $P_{mode,\varepsilon}$. The function find_stability (line 11) returns results in the form of the following dictionary of P-optimal decisions:

```
{ decision_path_tuple :  stability_epsilon }
```

In the next step, the $P_{mode,\varepsilon}$ perturbations is calculated (line 14). The output of the

`find_perturbation_mode` function is the following:

```
{ (e1,e2) :  [list of decision_path_tuples] }
```

where `e1` and `e2` represent some range of ε values where the set of P-optimal decisions does not change (please node that a decision tree can have more than one P-optimal decision and hence a list is here used).

Finally, $P_{\min,\varepsilon}$ and $P_{\max,\varepsilon}$ are calculated with the function `find_perturbation_pessopty` (line 17). The results will be returned in the following format:

```
{ "max_min" :  { (e1,e2) :  [list of decision_path_tuples] },
 "max_max" :  { (e1,e2) :  [list of decision_path_tuples] } }
```

Both elements of the above dictionary (`"max_min"` and `"max_max"`) are analogous to the output of the `find_perturbation_mode` function. The value of `"max_min"` represents $P_{\min,\varepsilon}$ perturbation sensitivity while the value of `"max_max"` represents $P_{\max,\varepsilon}$ perturbation sensitivity.

It should be noted that for separable tree analysis Chondro supports the $s\colon \mathcal{C} \to [0,1]$ function representing whether a given chance node should be subject to sensitivity analysis. Simply add the `s` property to any chance node in a separable tree. The default value is $\mathtt{s} = 1$ i.e. all chance nodes in a separable tree will be considered in sensitivity analysis of a DT.

Listing 1.1: Source code for separable probability

```
1  from chondro import *
2
3  file_name = "example_separable_Fig5.json"
4  tree = load_tree(file_name)
5  ev,dec=solve_tree(tree)
6  print ("DT␣has␣been␣solved,␣the␣expected␣value␣is␣ev:␣"+str(ev)+ \
7         "␣reachable␣decisions:␣"+str(get_reachable(dec)))
8
9  print_tree(tree)
10
11 stabi = find_stability(tree,precision=Fraction("1/10000") )
12 print ("DT␣stability", stabi)
13
14 ress = find_perturbation_mode(tree,precision=Fraction("1/1000"))
15 print("DT␣perturbation␣mode",ress)
16
17 ress = find_perturbation_pessopty(tree,precision=Fraction("1/1000"))
18 for key in ress.keys():
19         print ("P_"+key, ress[key])
```

Listing 1.2: Textual representation of decision tree from Figure 1.1 in Chondro. For each chance node the probabilities `p` are given and the expected values `ev` have been calculated. The asterix (*) represents the optimal decision(s). It should be noted that Chondro uses `fractions` package and hence the results can be calculated utilizing full precision.

```
d1:decision
  *c1:chance (ev=48)
    t1:p=3/5 final [60]
    t2:p=2/5 final [30]
   c2:chance (ev=44)
    t3:p=7/10 final [20]
    t4:p=3/10 final [100]
   c3:chance (ev=44)
    t5:p=4/5 final [40]
    t6:p=1/5 final [60]
```

1.3 Sensitivity analysis of non-separable trees

The Chondro library is capable of processing both separable and non separable trees. Due to much larger computational complexity of non-separable the library has different internal implementation of stability and perturbation algorithms for both tree types. A sample non-separable decision tree has been presented in Figure 1.2.

The support for non-separable trees is achieved by providing to the function `solve_tree` an additional parameter `derived_probs_dict` that contains a dictionary of key-probability values that can be injected into `pi` fields in a decision tree. Hence, there are two differences in processing non-separable DTs compared to separable ones:

- probabilities in the decision tree are represented as keys rather than values and use `pi` fields instead of `p` fields.
- in order to perform sensitivity analysis a function needs to be provided that transforms fundamental probabilities into dictionary of key-value pairs that can be injected by Chondro into `pi` fields in a DT

Calculating stability and perturbation requires performing a sweep over a set of fundamental probabilities and providing a function transferring those probabilities into a key-probability dictionary. Hence, the methods `find_stability`, `find_perturbation_pessopty` and `find_perturbation_mode` require providing two additional parameters:

- *derived_probs_lambda* - a function that calculates derived probabilities on the base of fundamental ones. The function should return a dictionary where keys are corresponding to `pi` values in a decision tree.
- *fundamental_probs* - a list of initial vales of fundamental probabilities

An example function that calculates probabilities on the base of fundamental ones has been presented in (e.g. see Listing 1.3.

The initial values for fundamental probabilities is represented as a list of events where each event is described by a list of outcome probabilities. Moreover, if there are n possible outcomes of an event the probabilities of $n-1$ should be only passed - the last n-th probability will be automatically calculated. For example suppose that we consider two fundamental probabilities result of throwing a coin and a result of throwing a four-sided dice. In that case the fundamental probabilities in Chondro will be presented as: `[[0.5],[0.25,0.25,0.25]]`.

In Listing 1.3 an example processing of a non-separable decision tree has been presented. Firstly, fundamental probabilities values need to be defined (line 4). Those values can be used to calculate a P-optimal decision (line 5). In the line 7 we defined s values representing whether a particular event (for which fundamental probabilities have been given) should be a subject of sensitivity analysis. Finally, we perform the stability analysis - in our computations we limit the maximum considered value of ε to 1.

Listing 1.3: Source code for function transforming separable probabilities (`probs`) into non-separable probabilities at nodes of the tree presented in Figure 1.2

```
def tree_derived_probs_lambda(probs):
        p=dict()
        p["gas"] = probs[0][0]
        sensitivity = probs[1][0]
        specifity = probs[2][0]
        p["no_gas"] = 1-p["gas"]
        p["pos._test"]=sensitivity*p["gas"]+  \
        (1-specifity)*p["no_gas"]
        p["neg._test"]=1-p["pos._test"]
        p["gas|pos._test"]=sensitivity*p["gas"]/p["pos._test"]
        p["no_gas|pos._test"]=1-p["gas|pos._test"]
        p["gas|neg._test"]=(1-sensitivity)*p["gas"]/  \
        p["neg._test"]
        p["no_gas|neg._test"]=1-p["gas|neg._test"]
        return p
```

Listing 1.4: Calculating sensitivity analysis for a separable decision tree with Chondro. We have assumed that the function tree_derived_probs_lambda has been defined as presented in Listing 1.3.

```
from chondro import *

tree = load_tree("exaple_non_separable_fig7.json")
fund_probs = [[Fraction("7/10")],[Fraction("9/10")],[Fraction("7/10")]]
solve_tree(tree,tree_derived_probs_lambda(fund_probs))
print_tree(tree)
s = [1,0.1,0.1]
stabi = find_stability(tree,tree_derived_probs_lambda, \
          fund_probs,precision=Fraction("1/100"),s=s,max_epsilon=1) )
print ("stability", stabi)
ress = find_perturbation_mode(tree,tree_derived_probs_lambda,\
         fund_probs,precision=Fraction("1/100"),s=s,max_epsilon=1))
print("mode perturbation",ress)
ress = find_perturbation_pessopty(tree,tree_derived_probs_lambda,\
          fund_probs,precision=Fraction("1/100"),s=s,max_epsilon=1) )
for key in ress.keys():
    print ("P "+key, ress[key])
```

Reference manual

2.1 List of available functions

chondro.**bisect**(*fun*, *a*, *b*, *precision=1e-06*, *divide=2.0*)

Implements bisection with a given precision.

fun - a boolean function

a - start value

b - end value

precision - precision parameter

divide - division parameter

examples:

bisect(lambda x: x*x-5 > 0, 0.0,5.0)

bisect(lambda x: x*x-5 > 0, Fraction(0),Fraction(5),divide=Fraction(2))

chondro.**bisect_change**(*fun*, *start_value*, *a*, *b*, *precision=1e-06*)

Implements bisection with a given precision - searching for a change in function value.

fun - a boolean function

start_value - a starting value of the function

a - start value

b - end value

precision - precision parameter

returns x where the change was observed and the new function value

chondro.**find_gamma_probs**(*probs*, *e*)

Finds a gamma value for the most probable values.

probs - a list of probabilities

e - epsilon

examples:

find_gamma_probs([Fraction("1/4"),Fraction("3/8"),Fraction("3/8")], Fraction(0.249999))

find_gamma_probs([Fraction("1/4"),Fraction("3/8"),Fraction("3/8")], Fraction(0.25))

find_gamma_probs([Fraction("1/4"),Fraction("3/8"),Fraction("3/8")],0.1)

`chondro.`**`find_perturbation_mode`**(*tree*, *derived_probs_lambda=None*, *grouped_fundamental_probs=None*, *precision=Fraction(1, 100)*, *max_epsilon=None*, *s=None*, *use_labels=True*)

Finds peturbation stability for mode-perturbation type The grid contains the original probabilities as well as the corner cases (probabilities equal to zero and one).

tree - a decision tree represented as a dictionary

derived_probs_lambda - a function that calculates derived probabilities on the base of fundamental ones

grouped_fundamental_probs - a list of groups of fundamental probabilities

precision - step value for the gamma parameter sweep

max_epsilon - maksimum epsilon value considered in the computation

s - sensitivity list for fundamental probabilities

use_labels - decisions will be presented as labels rather than indices

Output format:

```
{ (e1,e2) :  [list of decision_path_tuples] }
```

where `e1` and `e2` represent some range of ε values where the set of $P_{mode,\varepsilon}$ does not change (please node that a decision tree can have more than one P-optimal decision and hence a list is here used).

`chondro.`**`find_perturbation_pessopty`**(*tree*, *derived_probs_lambda=None*, *grouped_fundamental_probs=None*, *precision=Fraction(1, 10)*, *max_epsilon=None*, *s=None*, *use_labels=True*)

Performs a full grid sweep search in order to find $P_{\min,\varepsilon}$ and $P_{\max,\varepsilon}$ values. The grid contains the original probabilities as well as the corner cases (probabilities equal to zero and one). If only epsilon stability is selected the search stops after finding the first set of fundamental probabilities that changes the decision.

tree - a decision tree represented as a dictionary

derived_probs_lambda - a function that calculates derived probabilities on the base of fundamental ones

fundamental_probs - a list of groups of fundamental probabilities

step - step value for the parameter sweep

max_epsilon - maksimum epsilon value for parameter sweep generation

s - the s parameter for fundamental probabilities

use_labels - decisions will be presented as labels rather than indices

Output format:

```
{ "max_min" :  { (e1,e2) :  [list of decision_path_tuples] },
  "max_max" :  { (e1,e2) :  [list of decision_path_tuples] } }
```

The value of `"max_min"` represents $P_{\min,\varepsilon}$ perturbation sensitivity while the value of `"max_max"` represents $P_{\max,\varepsilon}$ perturbation sensitivity. `e1` and `e2` represent some range of ε values where the set of P-optimal decisions does not change (please node that a decision tree can have more than one P-optimal decision and hence a list is here used).

`chondro.`**`find_stability`**(*tree*, *derived_probs_lambda=None*, *grouped_fundamental_probs=None*, *precision=Fraction(1, 10)*, *max_epsilon=None*, *s=None*, *use_labels=True*)

Finds the stability coefficient for every optimal decision in a given separable or nonseparable tree.

tree - a decision tree represented as a dictionary

derived_probs_lambda - a function that calculates derived probabilities on the base of fundamental ones

fundamental_probs - a list of groups of fundamental probabilities

precision - precision (for separable) or sweep step (for non separable trees)

max_epsilon - maximum value of an epsilon in the sweep

s - sensitivity list for fundamental probabilities

use_labels - decisions will be presented as labels rather than indices

Returns results in the form of the following dictionary of P-optimal decisions:

```
{ decision_path_tuple :  stability_epsilon }
```

chondro.**full_sweep**(*grouped_probs*, *step=Fraction(1, 5)*, *max_epsilon=None*, *s=None*)
Generates a parameter sweep for a given set of grouped probabilities The corner probabilities 0 and 1 are included in the sweep if they are in the max_epsilon range.

The sum of probabilities in any group does not exceed 1.

grouped_probs - a list of probabilities groups [[p1, p2],[p3],[p4,p5]]

step - step value used to generate the parameter sweep

max_epsilon - maximum value of an epsilon in the sweep

chondro.**get_decision_name**(*tree*, *path_tuple*)
Transforms a given *path_tuple* to a tuple of node labels from a given decision *tree*

chondro.**get_reachable**(*dec*)
For a given decision strategy dictionary creates a copy with removed optimal decisions that are not reachable.

dec - decision dictionary

example:

get_reachable({():[0,1],(1,):[2,3],(2,):[1,2],(2,2):[0]}) will return {(): [0, 1], (1,): [2, 3]} because the nodes (2,) and (2,2) cannot be reached - at the root node the only optimal decisions are 0 and 1

chondro.**go_fractions**(*node*)
Recursively enables Fractions for a dictionary decision tree. The parameter types of 'p' and 'value' fields are changed fractions

node - tree or subtree

chondro.**load_tree**(*file_name*, *use_fractions=True*)
Loads a decision tree defintion from a give file. The file can be either a plain dict saved to JSON or a SilverDecisions file. If a SilverDecisions file has several trees only the first decision tree in a file is read.

file_name - a name of the file

use_fractions - should "fractions" module be used for computational accurancy

chondro.**node_branching**(*node*, *__branching=()*)
Creates a set of all possible decisions for a given decision tree.

node - root node of a decision tree

chondro.**perturb**(*probs*, *v*, *epsilon*, *maximize*)
Calculates a peturbation for a given parameter set

probs - list initial probabilities

v - payoff values

epsilon - max perturbation

maximize - maximize (True) or minimize payoff (False)

chondro.**print_tree**(*node*, *__level=0*)
Recursively prints a decision tree subtree.

node - tree root or sub-node

__level - internal parameter of the function - level of the tree

chondro.**save_tree**(*tree*, *file_name*, *overwrite=False*)

Saves the tree definition to a file.

tree - a dictionary representation of a decision tree

file_name - a name of the file

overwrite - should the file be overwritten if it exists

chondro.**solve_tree**(*node*, *derived_probs_dict=None*, *node_stability_type=None*, *decision=None*, *__branching=()*, *__best_path=None*)

Recursive method to find an optimal value for a decision tree.

node - tree or sub-tree to be calculated

derived_probs_dict - dictionary of probabilities for non-separable trees (if present a node will use those probability values instead of 'p')

node_stability_type - a lambda (probs,evs,best_path,s) to recalculate probabilities at chance nodes

decision - decision dictionary to calculate ev of a tree for a particular decision

__branching - internal method parameter - current position in the tree

__best_path - internal method parameter - best paths in the tree

The function returns a tuple: `(expected_value, optimal_decisions_dictionary)`. The optimal decision dictionary is presented as the following:

```
{ node_path_tuple :  [list of P-optimal node indices] }
```

WARNING! The function has the following side effects on the tree parameter:

- expected values of chance nodes will be saved in the 'ev' field
- children of a decision node will store a 'best' boolean field with `True` value indicating P-optimal decisions

chondro.**tree_branching_to_tuple**(*tree_branching*, *use_names_tree=None*)

Converts a full decision dictionary to a tuple of tuples.

tree_branching - list of dictionaries generated with tree_sweep

use_names_tree - use labels from decision tree rather than indices to represent nodes

example tree_branching_to_tuple({():[1],(1,): [2],(1,2):[3,4]}) yields (((1, 2, 3),), ((1, 2, 4),))

chondro.**tree_decision_paths_to_tuple**(*tree_sweep_decision*)

Converts a decision dictionary to a list decision.

example:

tree_decision_paths_to_tuple({():[1],(1,): [2],(1,2):[3]})

returns ((1, 2, 3),)

chondro.**tree_sweep**(*decision_dictionary*)

Transforms a *decision_dictionary* to list of dictionaries, with single decisions in each node.

example:

tree_sweep({(1,):[1,2]})

returns [{(1,): [1]}, {(1,): [2]}]

2.2 Decision tree format specification

Listing 2.1 presents a specification JSON schema a decision tree. It should be noted that `p` and `value` fields accept both `number` and `string` values. The textual values enable entering rational numbers in `fractions`

module format (e.g. `1/7`) and thus avoiding loss of precision. It should be noted that when using `string` to represent numbers in a DT the function `load_tree` requires using the default value (`True`) of the parameter `use_fractions`.

The `p` value in chance node children is used to represent possibilities as real numbers while the `pi` value is used to represent a field in a dictionary that will be passed to the `solve_tree` function.

The fields `ev` and `best` are added to the tree when the `solve_tree` is called in order to indicate expected values on the chance nodes and best paths in a DT. It should be noted that the function `save_tree` does not store values of those fields.

Listing 2.1: JSON schema for the decision tree representation.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "definitions": {
    "node": {
      "type": "object",
      "properties": {
        "value": { "type": ["string","number"] },
        "id": { "type": "string" },
        "label": { "type": "string" }
      }
    },
    "probabilities": {
      "type": "object",
       "properties": {
          "p": { "type":  ["string","number"] },
          "pi": { "type":  "string"}
       },
       "oneOf" : [{"required" : ["p"]},{"required" : ["pi"]}]
    },
    "best": {
      "type": "object",
       "properties": {"best": { "type":  "boolean" }}
    },
    "node-decision": {
      "allOf": [
       {"$ref": "#/definitions/node" },
       { "properties" : {
         "type":{"type":"string","enum": ["decision"]},
         "nodes": {
           "type": "array",
           "items": { "oneOf": [
                  {"allOf": [
                    { "$ref": "#/definitions/node-decision" },
                    { "$ref": "#/definitions/best" }
                  ]},
                  {"allOf": [
                    { "$ref": "#/definitions/node-chance" },
                    { "$ref": "#/definitions/best" }
                  ]},
                  {"allOf": [
                    { "$ref": "#/definitions/node-final" },
                    { "$ref": "#/definitions/best" }
                  ]}
                ]
           }
         }
       }}
```

```
      ],
      "required": ["type","nodes"]
    },
    "node-final": {
      "allOf": [
        {"$ref": "#/definitions/node" },
        { "properties" : {
          "type":{"type":"string","enum": ["final"]}
        }}
      ],
      "required": ["type","value"]
    },
    "node-chance": {
      "allOf": [
        {"$ref": "#/definitions/node" },
        { "properties" : {
          "type":{"type":"string","enum": ["chance"]},
          "s":{"type":"number"},
          "ev":  {"type":"number"},
          "nodes": {
            "type": "array",
            "items": { "oneOf": [
              {"allOf": [
                { "$ref": "#/definitions/node-decision" },
                { "$ref": "#/definitions/probabilities" }
              ]},
              {"allOf": [
                { "$ref": "#/definitions/node-chance" },
                { "$ref": "#/definitions/probabilities" }
              ]},
              {"allOf": [
                { "$ref": "#/definitions/node-final" },
                { "$ref": "#/definitions/probabilities" }
              ]}
            ]
            }
          }
        }}
        ],
        "required": ["type","nodes"]
    }
  },

  "type": "object",
  "properties": {
    "tree": { "oneOf": [
              { "$ref": "#/definitions/node-decision" },
              { "$ref": "#/definitions/node-chance" },
              { "$ref": "#/definitions/node-final" }
              ]
            }
  },
  "required": ["tree"]
}
```