

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

УТВЕРЖДАЮ  
Директор Ассоциации  
«Искусственный интеллект в  
промышленности»

\_\_\_\_\_ Т. М.Супатаев  
\_\_\_\_\_ 2024

УТВЕРЖДАЮ  
Научный руководитель ИЦ СИИП  
Университета ИТМО

\_\_\_\_\_ А. В. Бухановский  
\_\_\_\_\_ 2024

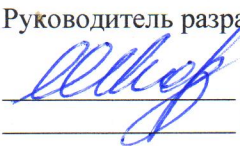
ИНСТРУМЕНТАРИЙ ВИЗУАЛИЗАЦИИ ВЕРОЯТНОСТНЫХ КОГНИТИВНЫХ КАРТ


ТЕКСТ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

RU.CНАБ.00853-02 12 ГГ-ЛУ

Представители  
Организации-разработчика

Руководитель разработки  
 Я.С. Коровин  
\_\_\_\_\_ 2024

Нормоконтролер  
 Е. В. Игнатова  
\_\_\_\_\_ 2024

Инв. № подл.	Подп. и дата	Взам. Инв №	Инв. № дубл.	Подп. и дата

УТВЕРЖДЕН  
RU.СНАБ.00853-02 12 ГГ-ЛУ

ИНСТРУМЕНТАРИЙ ВИЗУАЛИЗАЦИИ ВЕРОЯТНОСТНЫХ КОГНИТИВНЫХ КАРТ

ТЕКСТ ПРОГРАММЫ  
RU.СНАБ.00853-02 12 ГГ

ЛИСТОВ 9

Инв. № подл.	Подп. и дата	Взам. Инв №	Инв. № дубл.	Подп. и дата

## АННОТАЦИЯ

Документ содержит исходный код компонента “Инструментарий визуализации вероятностных когнитивных карт”. Этот компонент предназначен для применения совместно с компонентом адаптивной оптимизации выполнения производственных процессов с использованием вероятностных моделей и динамически изменяемой среды. Компонент входит в состав ПО, разрабатываемого в рамках мероприятия М1 плана Исследовательского центра в сфере искусственного интеллекта «Сильный ИИ в промышленности» (ИЦ ИИ) в рамках соглашения с АНО «Аналитический центр при Правительстве Российской Федерации» (ИГК 000000D730321P5Q0002), № 70-2021-00141.

Компонент предназначен для визуализации вероятностных нечетких когнитивных карт, созданных с помощью компонента адаптивной оптимизации выполнения производственных процессов с использованием вероятностных моделей и динамически изменяемой среды.

## 1 ОБЩИЕ СВЕДЕНИЯ

Текст программы размещен в репозитории <https://gitlab.actcognitive.org/itmo-sai-code/cogmapoptimizer>.

## 2 ИСХОДНЫЙ КОД

Файл CogDrawer.py.

```
import os
import json as js
import wx
import pydot
from math import sqrt
from PIL import Image
import webbrowser

temp_png = "temp.png"
temp_dot_to_png = "dot_to_png.png"
temp_dot = "temp.dot"

def ShowMessage(msg, title=""):
    dlg = wx.MessageDialog(None, msg, title, wx.OK | wx.ICON_INFORMATION)
    dlg.ShowModal()
    dlg.Destroy()

def FindVertexNumByID(list_of_vertices, id):
    lo = 0
    hi = len(list_of_vertices)
    while lo < hi:
        mid = (lo+hi)//2
        if id < list_of_vertices[mid][0]:
            hi = mid
        else:
            lo = mid+1
    return list_of_vertices[lo-1][1]

def GetVerticesImpact(edge):
    count = 0
    result = 0.0
    while True:
        weight = "weight-"+str(count)
        prob = "prob-"+str(count)
        if weight not in edge.keys():
            break
        w = edge[weight]
        p = edge[prob]
        result += w*p
        count += 1
    return result

def GetVerticesWeights(edge):
    count = 0
    result = ""
    # Multimap
    while True:
        weight = "weight-"+str(count)
        prob = "prob-"+str(count)
        if weight not in edge.keys():
            break
```

```

        break
    w = edge[weight]
    p = edge[prob]
    result += f"{w}/{p}, "
    count += 1
# Classic map
if result == "":
    w = edge["weight"]
    result += f"{w}, "
result = result[:len(result) - 2] # del ", " at the string end
return result

```

```

class MainForm(wx.App):
    # Init form
    def __init__(self, redirect=False, filename=None):
        self.cmj_file_name = ""
        self.dot_file_name = ""
        self.file_date = 0

        wx.App.__init__(self, redirect, filename)
        self.frame = wx.Frame(None, title="Graph editor")
        self.panel = wx.Panel(self.frame)
        self.panel.SetBackgroundStyle(wx.BG_STYLE_CUSTOM)

        fgs = wx.FlexGridSizer(2, 1, 10, 10)
        fgs.AddGrowableRow(0)
        self.GraphImage = wx.StaticBitmap(self.panel)
        fgs.Add(self.GraphImage, 1, wx.EXPAND)
        self.panel.SetSizer(fgs)

        self.Bind(wx.EVT_SIZE, self.OnSize)

        menuBar = wx.MenuBar()
        fileMenu = wx.Menu()
        openItem = wx.MenuItem(fileMenu, wx.ID_OPEN, text="Open file",
kind=wx.ITEM_NORMAL)
        fileMenu.Append(openItem)
        exitItem = wx.MenuItem(fileMenu, wx.ID_EXIT, text="Exit",
kind=wx.ITEM_NORMAL)
        fileMenu.Append(exitItem)
        menuBar.Append(fileMenu, "&File")
        self.frame.SetMenuBar(menuBar)
        self.Bind(wx.EVT_MENU, self.MenuHandler)

        self.timer = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.Timer, self.timer)

        self.frame.Show()
        self.frame.SetSize(wx.Size(800, 600))
        self.frame.Centre(wx.BOTH)

    def Timer(self, event):
        if self.cmj_file_name == "":
            return
        # Get file size
        real_date = os.path.getmtime(self.cmj_file_name)
        if os.path.isfile(temp_dot_to_png) and self.file_date != real_date:
            self.file_date = real_date
            self.BuildDOT(self.cmj_file_name)
            self.DrawGraph(temp_dot_to_png)

```

```

event.Skip()

def OnSize(self, event):
    if self.cmj_file_name == "":
        self.panel.SetSize(1, 1)
        return

    # Resize image via PIL scaling to panel size
    if not os.path.isfile(temp_dot_to_png):
        return

    im = Image.open(temp_dot_to_png)
    wi, hi = im.size
    wp, hp = self.panel.GetSize()
    im_to_panel = Image.new('RGB', (wp, hp), (255, 255, 255))
    if not (wp > wi and hp > hi):
        ratio = min(wp/wi, hp/hi)
        wi *= ratio
        wi = int(wi)
        hi *= ratio
        hi = int(hi)
        im.thumbnail((wi, hi), resample=Image.LANCZOS)
    pos_x = int((wp - wi) / 2)
    pos_y = int((hp - hi) / 2)
    im_to_panel.paste(im, (pos_x, pos_y))
    im_to_panel.save(temp_png, "PNG")

    self.DrawGraph(temp_png)
    event.Skip()

def MenuHandler(self, event):
    id = event.GetId()
    if id == wx.ID_OPEN:
        self.OpenFile()
        return
    if id == wx.ID_EXIT:
        self.Exit()

def Exit(self):
    self.timer.Stop()
    # Delete temp files
    if os.path.isfile(temp_png):
        os.remove(temp_png)
    if os.path.isfile(temp_dot_to_png):
        os.remove(temp_dot_to_png)
    if os.path.isfile(temp_dot):
        os.remove(temp_dot)
    exit(0)

def OpenFile(self):
    openFileDialog = wx.FileDialog(self.frame, "Open", "", "", "Cognitive
map (*.cmj)|*.cmj",
                                wx.FD_OPEN | wx.FD_FILE_MUST_EXIST)
    openFileDialog.ShowModal()
    self.cmj_file_name = openFileDialog.GetPath()
    openFileDialog.Destroy()
    # Force first refresh
    w, h = self.frame.GetSize()
    self.frame.SetSize(w+1, h+1)
    self.frame.SetSize(w, h)
    # Job
    img_file = self.BuildDOT(self.cmj_file_name)

```

```

self.DrawGraph(img_file)
# Timer
self.file_date = 0
self.timer.Start(100)
# Run editor
webbrowser.open(self.cmj_file_name)

def BuildDOT(self, cmj_file):
    # Read JSON
    with open(cmj_file, "r") as cog_file:
        cogmap_json = cog_file.read()
        json = js.loads(cogmap_json)

    tab = "    "
    dot = "digraph G {\n" + tab + "graph [size=\"12!\"]\n" + tab +
"rankdir=RL\n"

    # Vertices
    vert_count = len(json["Vertices"])
    vertices = []
    for i in range(vert_count):
        vertix = json["Vertices"][i]
        vert_name = vertix["fullName"]
        vert_id = vertix["id"]
        graph_name = f"A{i}"
        dot += tab + f"{graph_name}    [label=\"{vert_name}\"]\n"
color="#000000"]\n"
        vertices.append((vert_id, graph_name))
    vertices.sort(key=lambda key: key[0])
    dot += "\n"

    # Edges
    edges_count = len(json["Edges"])
    edges = json["Edges"]
    for i in range(edges_count):
        v1 = edges[i]["v1"]
        v2 = edges[i]["v2"]
        if v1 != v2:
            vert1 = FindVertixNumByID(vertices, v1)
            vert2 = FindVertixNumByID(vertices, v2)
            impact = GetVerticesImpact(edges[i])
            label = GetVerticesWeights(edges[i])
            color = "#FF0000" if impact < 0 else "#008000"
            dot += tab + f"{vert1} -> {vert2}    [label=\"{label}\"]\n"
color="{color}"]\n" fontcolor="{color}"]\n" \
            f"fontsize=\"8\"]\n"

    dot += tab + f"\n"

    row = round(sqrt(vert_count))
    count = 0
    rank = tab + "{ rank=same; "
    for i in range(vert_count):
        rank += f"A{i} "
        count += 1
        if count >= row:
            rank = rank[:len(rank) - 1] # del " " at the string end
            rank += " }\n"
            dot += rank
            rank = tab + "{ rank=same; "
            count = 0
    dot += rank

```



```

dot = dot[:len(dot) - 1]  # del " " at the string end
dot += " }\n"
dot += tab + f"\n"

dot += " }\n"
file = open(temp_dot, "w")
file.write(dot)
file.close()

graphs = pydot.graph_from_dot_file(temp_dot)
graph = graphs[0]
graph.write(temp_dot_to_png, "dot", "png")

return temp_dot_to_png

def DrawGraph(self, img_file):
    if not os.path.isfile(img_file):
        return
    # Set new name for dot-file
    # Draw graph in dot-format
    # Save graph as picture
    # Show picture on form
    img = wx.Image(img_file, wx.BITMAP_TYPE_ANY)
    bit_map = img.ConvertToBitmap()
    self.GraphImage.SetBitmap(bit_map)

if __name__ == '__main__':
    # Delete old temp files
    if os.path.isfile(temp_png):
        os.remove(temp_png)
    if os.path.isfile(temp_dot_to_png):
        os.remove(temp_dot_to_png)
    if os.path.isfile(temp_dot):
        os.remove(temp_dot)

    app = MainForm()
    app.MainLoop()

```

[illegible]