

УТВЕРЖДАЮ  
Научный руководитель ИЦ СИИП  
Университета ИТМО

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

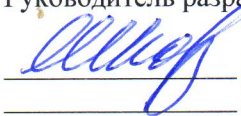
ИНСТРУМЕНТАРИЙ ПОДГОТОВКИ КОНКРЕТИЗИРУЮЩИХ ИИ-ЗАПРОСОВ  
ДЛЯ КОГНИТИВНЫХ КАРТ

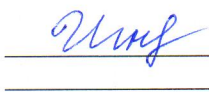
ТЕКСТ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

RU.СНАБ.00853-02 12 ВВ-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 ВВ-ЛУ

ИНСТРУМЕНТАРИЙ ПОДГОТОВКИ КОНКРЕТИЗИРУЮЩИХ ИИ-ЗАПРОСОВ  
ДЛЯ КОГНИТИВНЫХ КАРТ

ТЕКСТ ПРОГРАММЫ  
RU.СНАБ.00853-02 12 ВВ

ЛИСТОВ 8

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

## АННОТАЦИЯ

Документ содержит исходный код компонента “Инструментарий подготовки конкретизирующих ИИ-запросов для 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 интерпретации когнитивных карт, сгенерированных в качестве решений компонентом 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 Interpreter.py.

```
import sys
import os
import json as js

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 rreplace(where, what, replacement):
    return replacement.join(where.rsplit(what, 1))

def trunc_num(num, digits=5):
    result = num * (10**digits)
    result = int(result)
    result /= 10**digits
    return result

def ProcessCogMap(orig_filename, processed_filename):
    # Прочитать начальную когнитивную карту
    with open(orig_filename, "r") as cog_file:
        cogmap_json = cog_file.read()
        json = js.loads(cogmap_json)

    result = ""
    # Описание предметной области
    if "DomainDescription" in json.keys():
        domain_description = json["DomainDescription"]
        result = f"Есть граф, описывающий предметную область"
        ({domain_description.lower()}).\n"
    else:
```

```

result = f"Есть граф, описывающий некую предметную область.\n"

# Описание начального графа
# Описание вершин
result = result + "Граф состоит из множества вершин: "
vert_count = len(json["Vertices"])
vertices = []
for i in range(vert_count):
    vertix = json["Vertices"][i]
    vert_name = vertix["fullName"]
    vert_id = vertix["id"]
    result = result + f"вершина №{i+1} - \"{vert_name}\", "
    vertices.append((vert_id, vert_name)) # Запоминаем вершины
vertices.sort(key=lambda key: key[0])
result = result[:len(result)-2] # Удалить 2 последних символа - запятую с
пробелом
result = result + ".\n"

# Описание связей
result = result + "Вершины воздействуют друг на друга позитивно или нега-
тивно (если сила "
result = result + "воздействия положительная или отрицательная, соответ-
ственно). "
result = result + "Эти вершины связаны между собой следующим образом:\n"
edges_count = len(json["Edges"])
edges = json["Edges"]
for i in range(vert_count):
    verts = 0
    vertix = json["Vertices"][i]
    vli = vertix["id"]
    vert1 = FindVertixNumByID(vertices, vli)
    result = result + f"- вершина \"{vert1}\" воздействует на вершины "
    for j in range(edges_count):
        if i == j:
            continue
        v1j = edges[j]["v1"]
        v2j = edges[j]["v2"]
        if vli == v1j:
            vert2 = FindVertixNumByID(vertices, v2j)
            impact = GetVerticesImpact(edges[i])
            result = result + f"\n \"{vert2}\" с силой {trunc_num(impact)}, "
            verts += 1
    result = result[:len(result)-2] # Удалить последние 2 символа - запятую
и пробел
    result = result + ".\n"
    if verts < 2:
        result = rreplace(result, "воздействует на вершины ", "воздействует
на вершину ")

# Описание переработанного графа
result = result + "Данный граф был изменен, вследствие чего он принял следу-
ющий вид.\n"
# Прочитать переработанную когнитивную карту
with open(processed_filename, "r") as cog_file:
    cogmap_json = cog_file.read()
    json = js.loads(cogmap_json)

# Описание вершин
result = result + "Его вершины: "
vert_count = len(json["Vertices"])
vertices = []

```

```

for i in range(vert_count):
    vertix = json["Vertices"][i]
    vert_name = vertix["fullName"]
    vert_id = vertix["id"]
    result = result + f"вершина №{i+1} - \"{vert_name}\", "
    vertices.append((vert_id, vert_name)) # Запоминаем вершины
vertices.sort(key=lambda key: key[0])
result = result[:len(result)-2] # Удалить 2 последних символа - запятую с
пробелом
result = result + ".\n"

# Описание связей
result = result + "Вершины измененного графа связаны между собой следующим
образом:\n"
edges_count = len(json["Edges"])
edges = json["Edges"]
for i in range(vert_count):
    verts = 0
    vertix = json["Vertices"][i]
    vli = vertix["id"]
    vert1 = FindVertixNumByID(vertices, vli)
    result = result + f"- вершина \"{vert1}\" воздействует на вершины "
    for j in range(edges_count):
        if i == j:
            continue
        v1j = edges[j]["v1"]
        v2j = edges[j]["v2"]
        if vli == v1j:
            vert2 = FindVertixNumByID(vertices, v2j)
            impact = edges[j]["weight"]
            result = result + f"\n \"{vert2}\" с силой {trunc_num(impact)}, "
            verts += 1
    result = result[:len(result)-2] # Удалить последние 2 символа - запятую
и пробел
    result = result + ".\n"
    if verts < 2:
        result = result.replace("воздействует на вершины ", "воздействует
на вершину ")

    result = result + "Если разница в силе воздействия аналогичных вершин этих
двух графов меньше 5%, "
    result = result + "то можно считать, что сила не менялась и игнорировать эту
разницу.\n"

    result = result + "Вопрос: Объясни максимально подробно, с примерами, как
можно интерпретировать внесенные в "
    result = result + "начальный граф изменения и как реализовать такие измене-
ния?\n"

return result

def SaveResult(file_name, string):
    file = open(file_name, "w")
    n = file.write(string)
    file.close()

if __name__ == '__main__':
    # входные данные - файл когнитивной карты
    # выходные данные - текстовый файл с описанием когнитивной карты
    if len(sys.argv) < 3:

```

```
    print(f"Usage: {sys.argv[0]} <original_cognitive_map.cmj>  
<processed_cognitive_map.cmj>")  
    exit(-1)  
    o_filename = sys.argv[1]  
    p_filename = sys.argv[2]  
  
    print(f"Processing \"{o_filename}\" and \"{p_filename}\"...", end="")  
    res = ProcessCogMap(o_filename, p_filename)  
    print(f"{chr(8)}{chr(8)}{chr(8)}", end="")  
    print(f" done.")  
  
    result_filename = os.path.splitext(p_filename)  
    result_filename = result_filename[0]+".AI_description.txt"  
    print(f"Saving results to \"{result_filename}\"...", end="")  
    SaveResult(result_filename, res)  
    print(f"{chr(8)}{chr(8)}{chr(8)}", end="")  
    print(f" done.\n")
```



[illegible]