1 Алгоритм верификации состязательной устойчивости времени доставки сумки по отношению к эмбеддингам вершин для всей конвейерной сети

1.1 Анализ графа сети

Данный алгоритм основывается на уже разработанном алгоритме верификации [1]. Можно выполнить верификацию на всех парах вершин в графе конвейерной сети, число которых равняется $\binom{n}{2}$, где n — размер графа. Но мы можем улучшить производительность алгоритма, минимизируя количество пар. Рассмотрим пример графа, представленный на рисунке 1.

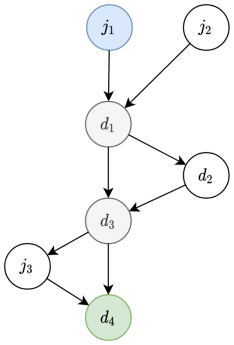

Рисунок 1 — Пример конвейерной сети

Мы считаем, что разделители хорошо обучены, поэтому сумка обычно движется по кратчайшему пути. Тогда если устойчивость для пары вершин (j_1, d_4) доказана, мы можем считать, что она соблюдается для суффиксов кратчайшего пути: $(d_1, d_4), (d_3, d_4)$ — так как соответствующие нейронные сети будут получать одинаковые эмбеддинги на вход.

Задача поиска ожидаемого времени доставки параметризируется множеством разделителей, поэтому время доставки для исходной пары и для суффикса соответственно можно формализовать следующим образом: $\{d_1, d_3\} \to t_{j_1}$, $\{d_3\} \to t_{d_3}$. Задача для суффикса является подзадачей для исходной пары. Нам также следует учитывать, что если пара не является суффиксом, она может повторять задачу другой пары, как в случае с (j_1, d_4) и (j_2, d_4).

Чтобы из множества пар, на которых необходимо запускать верификацию, исключить наибольшее число элементов, будем искать пару, которая имеет наибольший кратчайший путь и содержит наибольшее число суффиксов.

1.2 Описание алгоритма

Псевдокод алгоритма приведен в листинге 1.

Листинг 1 — Псевдокод алгоритма

```
1: procedure Pairs(g)
2:     V ← Nodes(g)
3:     for pair ∈ V² do
4:         p ← DijkstraPath(g, pair)
5:         if ∃ p ∧ diverter ∈ p then
6:             P ← p
7:         end if
8:     end for
9:
10:    Sort(P)                                  ▷ по возрастанию кол-ва узлов
11:    while P ≠ ∅ do
12:        p ← Pop(P)
13:        RemoveSuffixes(P, p)
14:        pair ← (p[0], p[−1])
15:        if Diverters(pair) ∉ Diverters(WithSink(res, p[−1])) then
16:            res ← pair
17:        end if
18:    end while
19:
20:    return res
21: end procedure
22:
23: procedure FullEmbAdvVerif(g)
24:    for p ∈ Pairs(g) do
25:        m ← CreateAbsorbingMC(g, p[1])          ▷ если конечный узел не
26:                                                ▷ является стоком, то необходимо
27:                                                ▷ удалить все его исходящие ребра
28:        s ← CreateEDTSolver(m, p[0])
29:        if NontrivDvtrs(s) = ∅ then
30:            continue
31:        end if
32:
33:        t⊤ ← BoundEstimation(s)
34:        for k ∈ (0.95, 0.99, 1.01, 1.05) do
35:            t_b ← t⊤ · k
36:            res ← EmbAdvVerif(t_b)
37:        end for
38:    end for
39: end procedure
```

1. Для каждой пары узлов графа конвейерной сети находим кратчайший путь (4). Если путь существует и содержит хотя бы один разделитель, то сохраняем его (5–6).

2. Извлекаем самый длинный путь и удаляем все его суффиксы из множества найденных путей (10–13). Если множество разделителей данной пары не совпадает с каким-либо множеством разделителей сохраненных пар, которые имеют тот же сток, то сохраняем эту пару (14–

16).

3. Если найденная пара имеет нетривиальные разделители, то находим для нее оценку времени доставки и выполняем верификацию (24–36).

1.3 Результат работы

Результаты выполнения алгоритма приведены в таблице 1

Таблица 1 — Результат полной верификации сети

	$t_\top$	t_b	Вериф.	Прод., с
$o_1 \to d_7$	40.10	38.10	−	0.09
		39.70	−	0.19
		40.50	+	0.78
		42.11	+	0.52
$o_1 \to d_8$	50.10	47.60	−	0.09
		49.60	−	0.11
		50.60	+	0.74
		52.61	+	0.34
$o_1 \to i_0$	50.15	47.64	−	0.09
		49.65	−	0.11
		50.65	+	0.77
		52.66	+	0.53
$o_0 \to i_0$	40.20	38.19	−	0.10
		39.80	−	0.12
		40.60	+	0.94
		42.21	+	0.66
$o_0 \to i_1$	53.10	50.45	−	0.11
		52.57	−	0.11
		53.63	+	0.76
		55.76	+	0.41
$o_1 \to i_1$	43.25	41.09	−	0.15
		42.82	−	0.17
		43.68	+	2.85
		45.41	+	1.80
$o_1 \to i_2$	53.06	50.41	−	0.12
		52.53	−	0.16
		53.59	+	4.54
		55.71	+	1.64
$o_0 \to i_2$	53.07	50.42	−	0.09
		52.54	−	0.11
		53.60	+	0.63
		55.72	+	0.28

$o_1 \to i_3$	43.10	40.95	−	0.14
		42.67	−	0.17
		43.53	+	2.27
		45.26	+	1.45
$o_0 \to i_3$	43.15	40.99	−	0.10
		42.72	−	0.11
		43.58	+	0.91
		45.31	+	0.54
$d_3 \to i_3$	43.10	40.95	−	0.10
		42.67	−	0.12
		43.53	+	0.82
		45.26	+	0.56
$o_1 \to j_0$	30.10	28.60	−	0.09
		29.80	−	0.11
		30.40	+	0.89
		31.61	+	0.52

Список литературы

[1] Мультиагентные алгоритмы маршрутизации на основе глубоких нейронных сетей с подкреплением и их верификация. — Санкт-Петербург. — 2020. — 108 с.