ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ ПОИСКА КРАТЧАЙШИХ ПУТЕЙ НА ГРАФАХ

Выполнил: Ткаченко Г.С.

Руководитель: Корнеев Г.А.

10 мая 2015 г.

Университет ИТМО

Проблема и задача

РЕШАЕМАЯ ПРОБЛЕМА

- Низкая производительность отдельных алгоритмов на специфичных графах
- Недостаточное разнообразие параллельных алгоритмов для поиска кратчайших путей

ПОСТАНОВКА ЗАДАЧИ

- Эффективное применение алгоритмов поиска кратчайшего пути на **многопроцессорных** архитектурах
- Разработка алгоритмов для поиска пути от одной вершины до всех (**one-to-many**)
- Разработка алгоритмов для поиска пути кратчайшего расстояния между каждой парой вершин (**many-to-many**)

ЗАДАЧА ONE-TO-MANY

Обзор решений

- Алгоритм Дейкстры
- Алгоритм Беллмана-Форда
- Алгоритм Джонсона (Дейкстра с потенциалами)
- Алгоритмы A* и D*

Классический Беллман-Форд

```
1: procedure ClassicBellmanFord(G, start)
2:     dist ← {∞...∞}
3:     dist[start] ← 0
4:     for i = 0 to |G.vertices| − 1 do
5:         for e ∈ G.edges do
6:             dist[e.to] ← min(dist[e.to], dist[e.from] + e.w)
7:     return dist
```

BFS Беллман-Форд

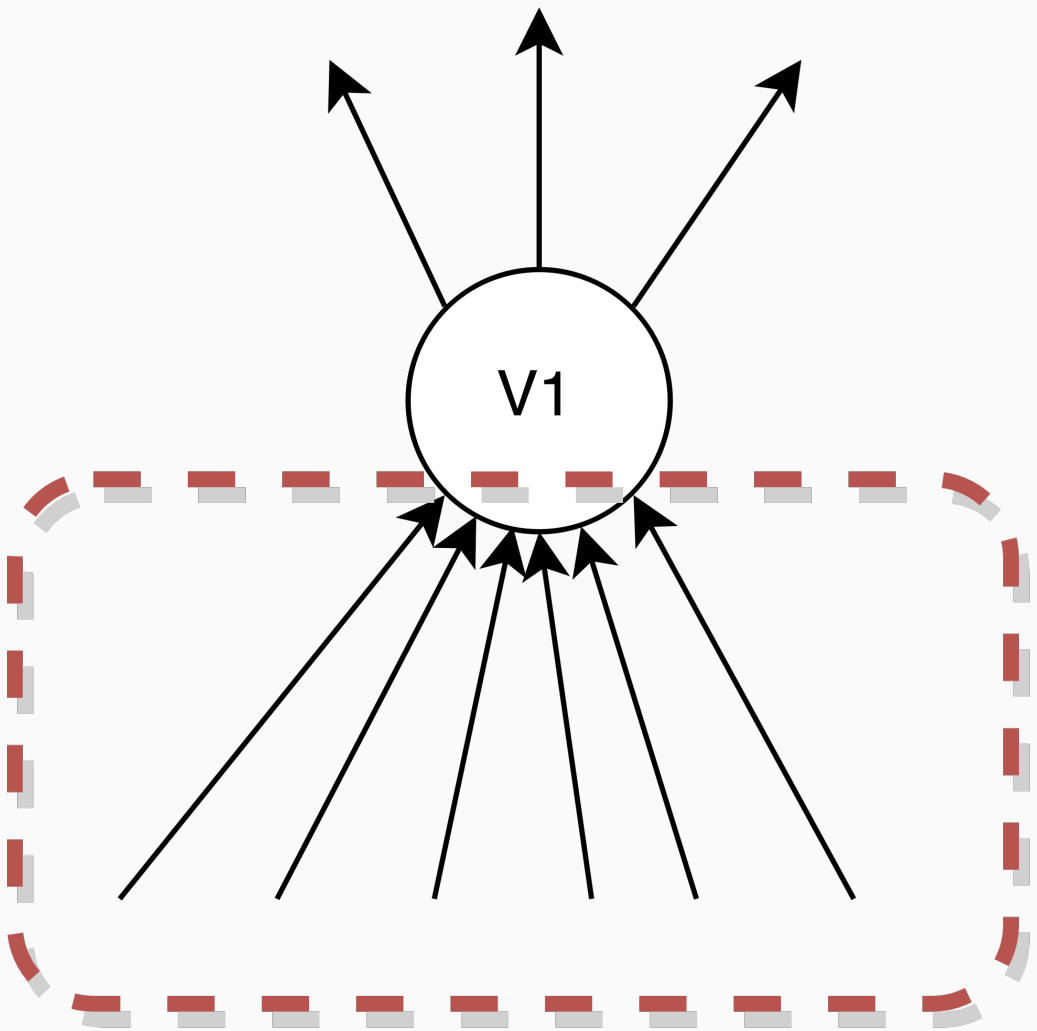
```
1:  procedure BFSBellmanFord(G, start)
2:      dist ← {∞...∞}
3:      dist[start] ← 0
4:      CurrentVertexSet ← {start}
5:      NextVertexSet ← ∅
6:      while CurrentVertexSet.empty() do
7:          NextVertexSet.clear()
8:          for v ∈ CurrentVertexSet do
9:              for e ∈ G.edgesFrom[v] do
10:                 if dist[e.to] > dist[e.from] + e.w then
11:                     dist[e.to] ← dist[e.from] + e.w
12:                     NextVertexSet.insert(e.to)
13:         CurrentVertexSet ← NextVertexSet
14:     return dist
```

Параллельный Беллман-Форд

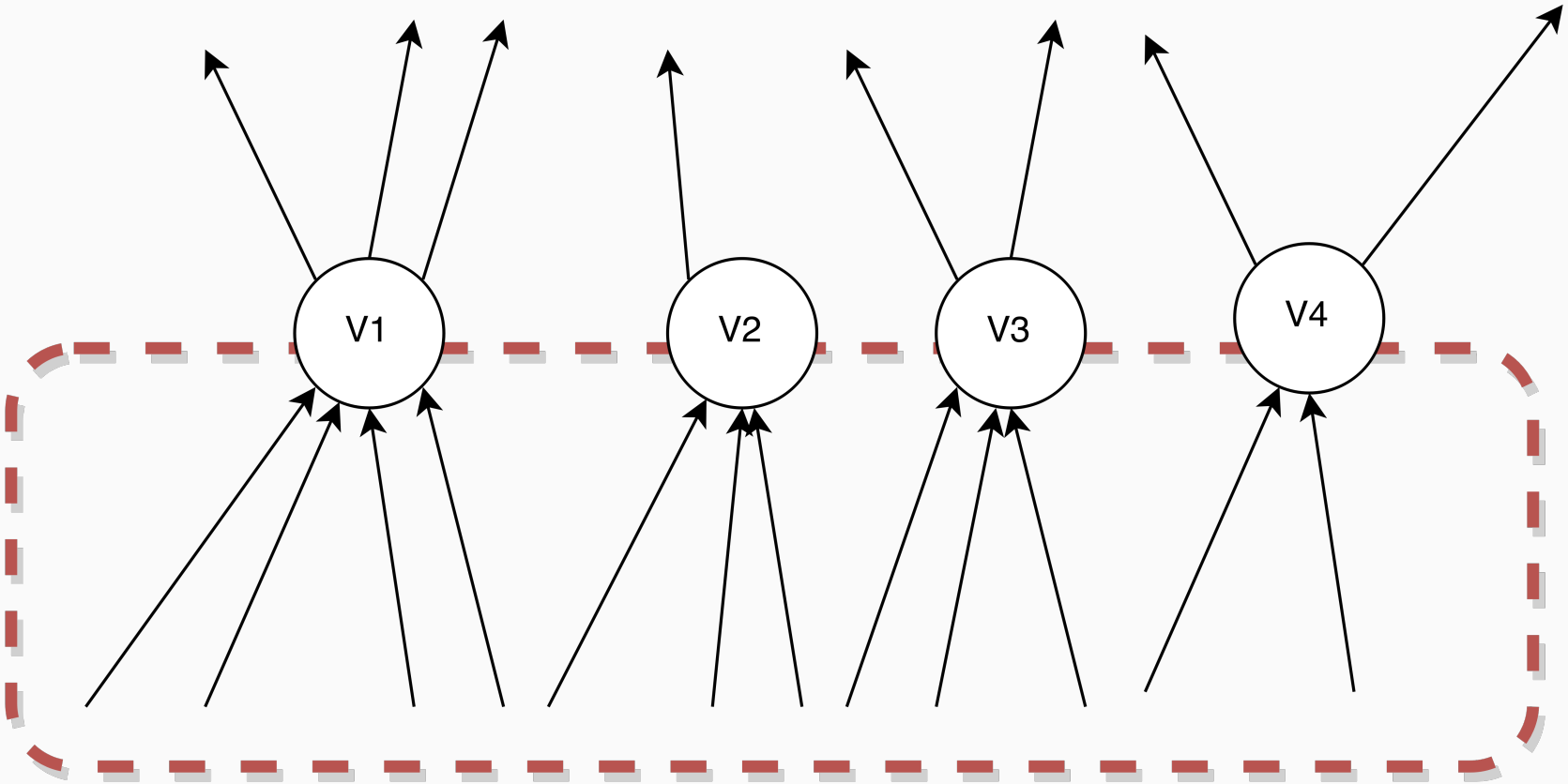
Три подхода

- Параллелизация по ребрам вершины
- Параллелизация по всем ребрам
- Использование параллельного обхода в ширину

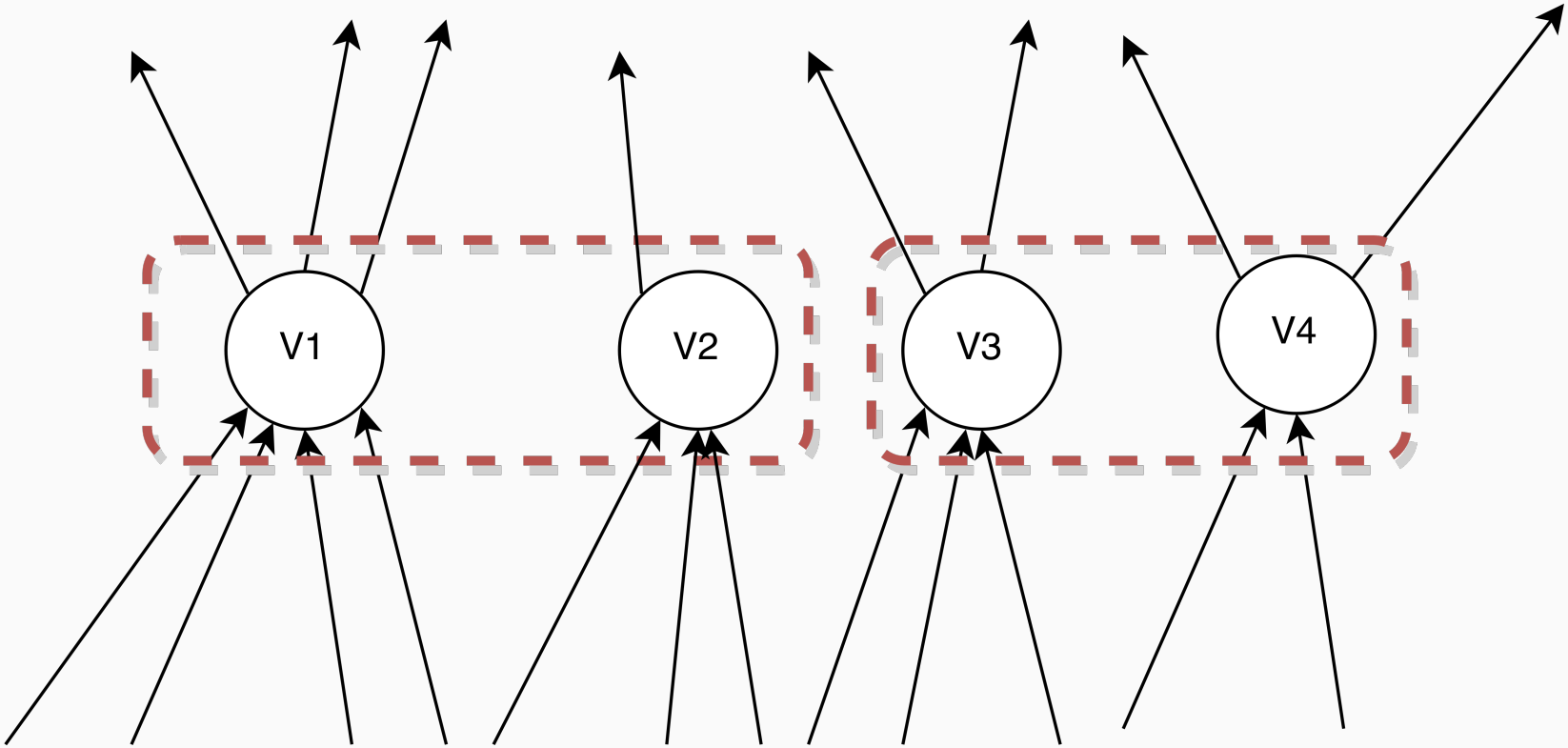
ПАРАЛЛЕЛИЗАЦИЯ ПО РЕБРАМ ВЕРШИНЫ



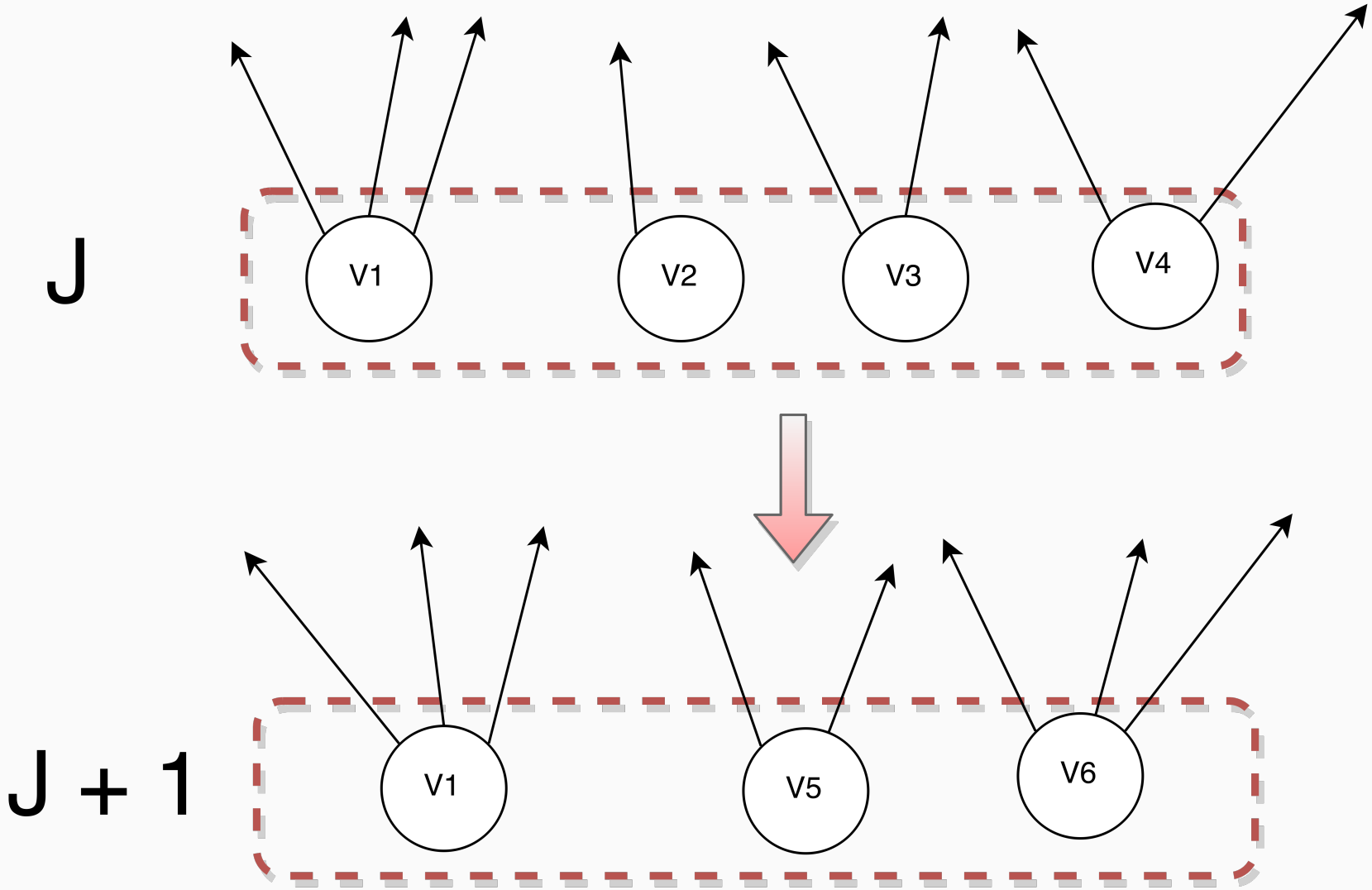
ПАРАЛЛЕЛИЗАЦИЯ ПО ВСЕМ РЕБРАМ



ПАРАЛЛЕЛИЗАЦИЯ ПО ВСЕМ РЕБРАМ



ИСПОЛЬЗОВАНИЕ ПАРАЛЛЕЛЬНОГО ОБХОДА В ШИРИНУ



ЗАДАЧА MANY-TO-MANY

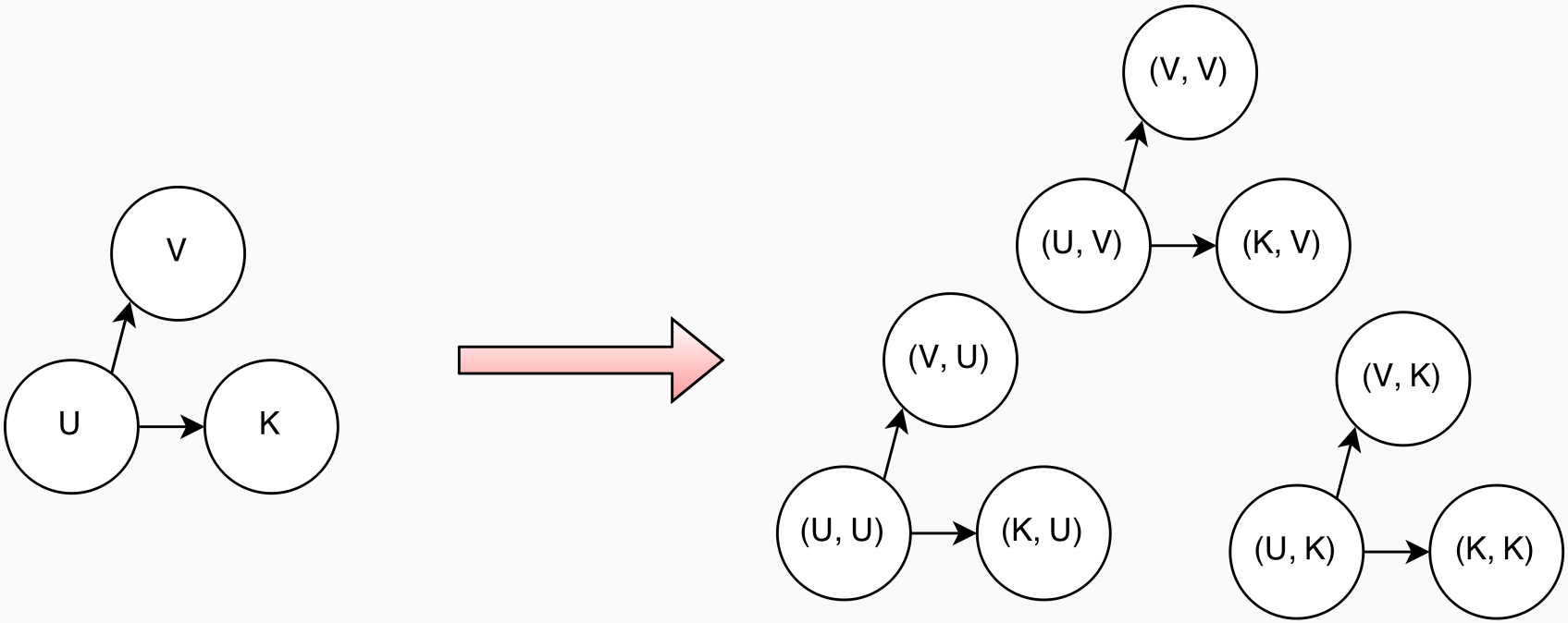
АЛГОРИТМ ФЛОЙДА

- В некоторых случаях классический алгоритм оказывается медленнее наивных алгоритмов
- Для каждой вершины можно использовать любой алгоритм поиска кратчайшего пути

НАИВНАЯ ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ

```
1:  procedure AllPairsPar1(G)
2:      return HandleVertices(G, 0, |G.vertices|)
3:
4:  procedure HandleVertices(G, startVertex, endVertex)
5:      if endVertex − startVertex < threshold then
6:          distances ← run Bellman-Ford for [startVertex, endVertex)
7:          return distances
8:      else
9:          midV ← (startVertex + endVertex)/2
10:         fork2(
                HandleVertices(G, startV, midV),
                HandleVertices(G, midV, endV));
```

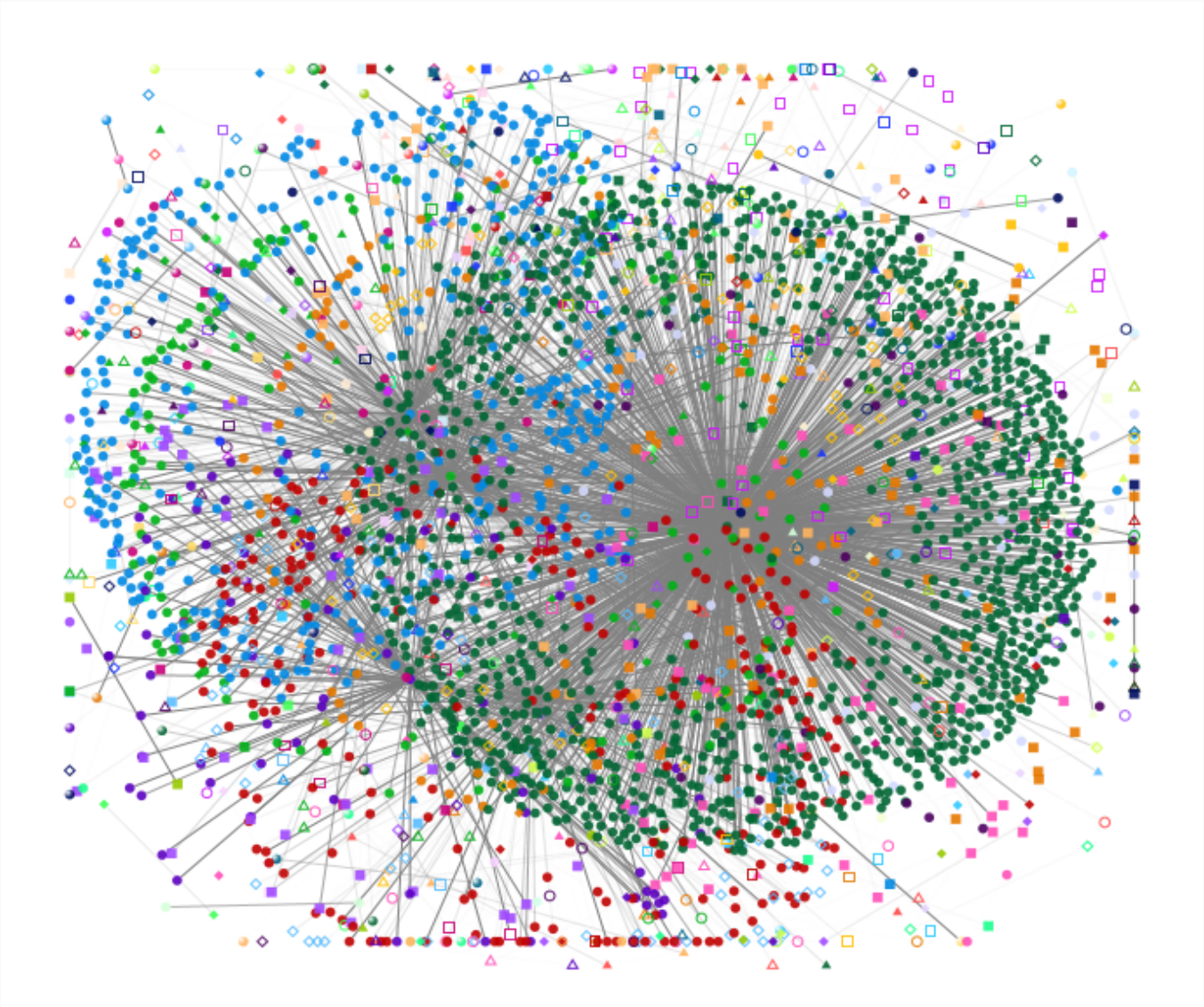
АЛГОРИТМ ДЛЯ ОБЪЕДИНЕННОГО ГРАФА



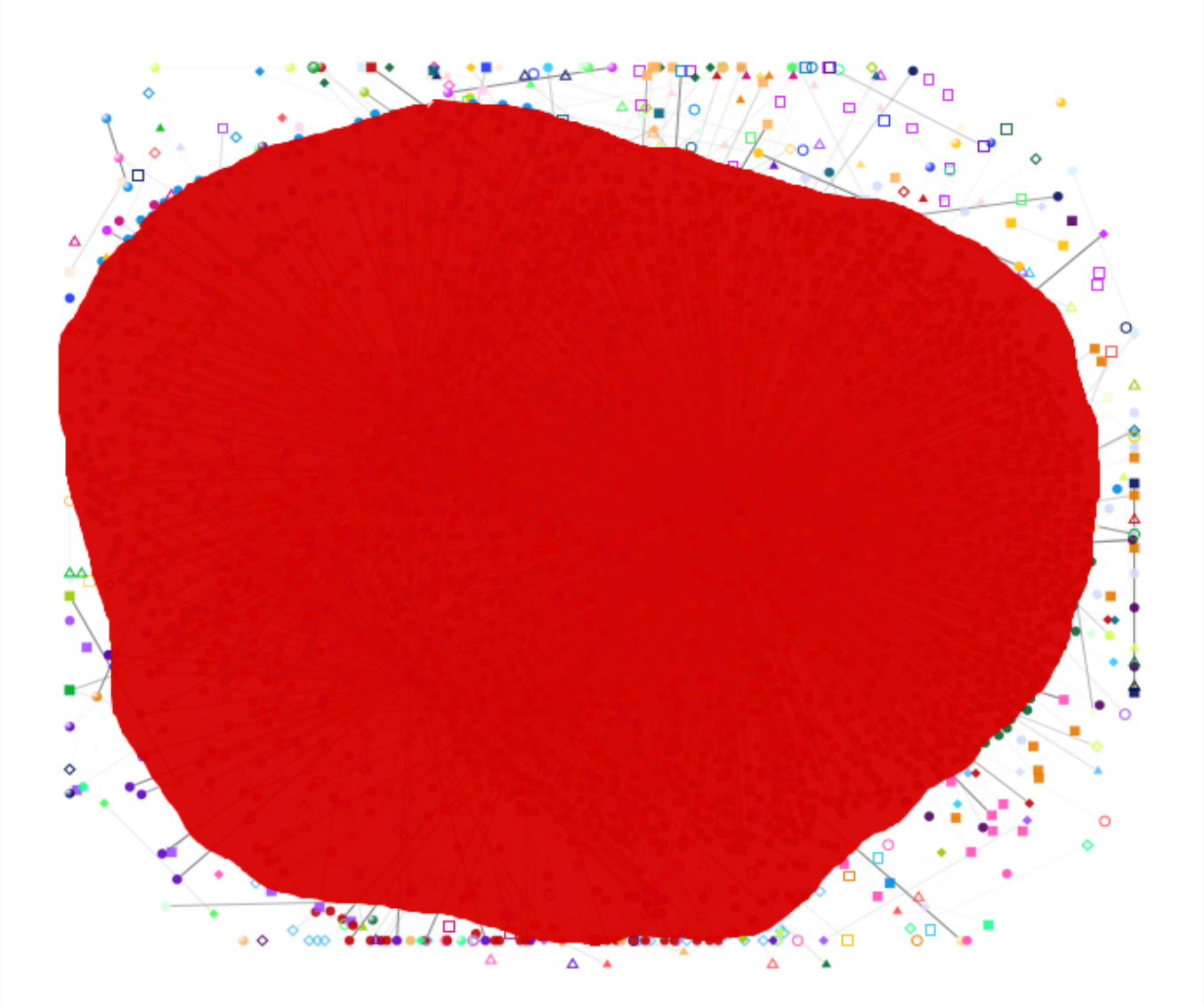
АЛГОРИТМ ДЛЯ СОЦИАЛЬНЫХ ГРАФОВ

- Основан на теории ”Шести рукопожатий”
- Работает не неориентированных невзвешенных социальных графах
- Использует идею динамического программирования

ИДЕЯ



ИДЕЯ



ДИНАМИКА ДЛЯ БОЛЬШЕГО МНОЖЕСТВА

- Посчитаем расстояние от базовой вершины до всех других
- Каждая из вершин u может находится только в $[dist[base][u] - K, dist[base][u] + K]$ слоях
- Будем поддерживать две динамики mask и calc. Причем mask[u][i] - сабсет вершин, расстояние от которых до u равно i. В свою очередь calc[u][i] - не более i

ПЕРЕСЧЕТ ЗНАЧЕНИИ ДИНАМИКИ

$$\text{mask}[v][i] = \neg\text{calc}[v][i-1] \wedge \bigvee_{\exists(u,v)\in E} \text{mask}[u][i-1] \tag{1}$$

$$\text{calc}[v][i] = \text{calc}[v][i-1] \vee \text{mask}[v][i] \tag{2}$$

ПРЕИМУЩЕСТВА ПОДХОДА

- Пересчет расстояний для группы вершин выполняется быстрее за счет битовых операций.
- Все битовые операции выполняются без выделения дополнительной памяти.
- Нет необходимости в построении следующего фронтира по предыдущему в процессе обработки
- Каждый из получившихся фронтиров довольно большой, что увеличивает его способность к
- Все остальные этапы также хорошо параллелятся

РЕЗУЛЬТАТЫ

СРАВНЕНИЕ ПАРАЛЛЕЛЬНЫХ ВЕРСИЙ БЕЛЛМАНА-ФОРДА

Algo №	Complete			BalancedTree		SquareGrid	
	TS	+	-	0.5	1	+	+-
1	2.43	4.65	nc	116.31	9.04	5.49	13.40
2	5.17	0.18	10.84	3.59	3.08	5.92	7.10
3	44.63	0.37	23.55	0.44	0.31	4.42	0.58

Таблица: Типичные графы

Algo №	RandomSparse			RandomDense		
	0.5+	0.5-	0.96+	0.5+	0.5-	0.96+
1	nc	nc	24.35	nc	nc	5.01
2	2.77	14.68	2.42	0.48	6.38	0.46
3	0.98	22.59	0.76	0.60	10.25	0.71

Таблица: Случайные графы

РАССТОЯНИЕ МЕЖДУ КАЖДОЙ ПАРОЙ ВЕРШИН СОЦИАЛЬНОГО ГРАФА

Алгоритм	Twitter graph
Наивная параллельная версия	427.217
Алгоритм для социальных графов	210.322

Таблица: Сравнение алгоритмов

ВЫВОДЫ

Вопросы?