Решение задачи составления производственного расписания с помощью параллельного случайного локального поиска на GPU

П.А. Борисовский

Институт математики им. С.Л. Соболева СО РАН (Омский филиал)

Исследование выполнено за счет гранта Российского научного фонда No 22-71-10015,

Задача составления производственного расписания

Постановка задачи[a].

$\boldsymbol{J}$ – множество работ, $\boldsymbol{I}$ – множество машин.

Каждая работа должна быть выполнена ровно один раз.

$\boldsymbol{p_{ij}}$ – длительность выполнения работы $\boldsymbol{j}$ на машине $\boldsymbol{i}$.

$[\boldsymbol{e_j}, \boldsymbol{d_j}]$ – временное окно для окончания работы $\boldsymbol{j}$.

$\boldsymbol{s_j}$ – максимально возможное запаздывание работы $\boldsymbol{j}$.

$\boldsymbol{z_i}$ – «стоимость» запуска любой работы на машине $\boldsymbol{i}$.

[a] J. Berndorfer, S. N. Parragh. Modeling and solving a real world machine scheduling problem with due windows and processing set restrictions. Procedia Computer Science, V. 200, 2022, P. 1646–1653. https://doi.org/10.1016/j.procs.2022.01.365

Задача составления производственного расписания

Критерий.

$$\boldsymbol{L + 0.001 \cdot Z \to \min},$$

где $\boldsymbol{L}$ – суммарное запаздывание
$\boldsymbol{Z}$ – суммарная стоимость запуска всех работ.

Случайный локальный поиск (Hillclimbing, (1+1)-EA)

1. Построить начальное решение $\boldsymbol{S^{(0)}}$.

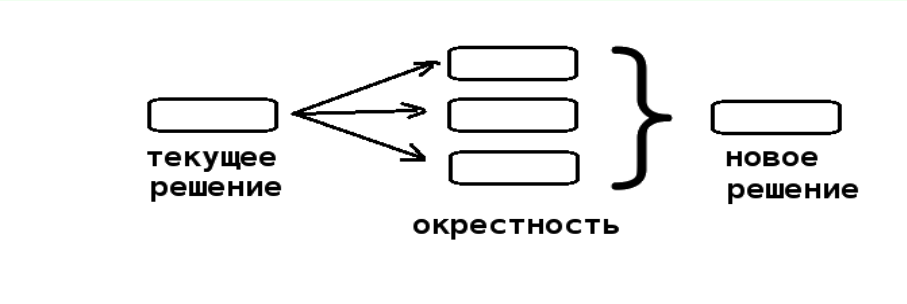
2. На каждой итерации $\boldsymbol{t}$ выполнять:

   2.1. Сгенерировать случайное решение $\boldsymbol{S'}$ из окрестности текущего решения $\boldsymbol{S^{(t)}}$ (т.е. применить небольшие случайные изменения $\boldsymbol{S^{(t)}}$).

   2.2. Если $\boldsymbol{f(S') < f(S^{(t)})}$, то $\boldsymbol{S^{(t+1)} := S'}$, иначе $\boldsymbol{S^{(t+1)} := S^{(t)}}$.

Параллельный случайный локальный поиск

1. Параллельный просмотр окрестности



Алгоритм (1+K)-EA

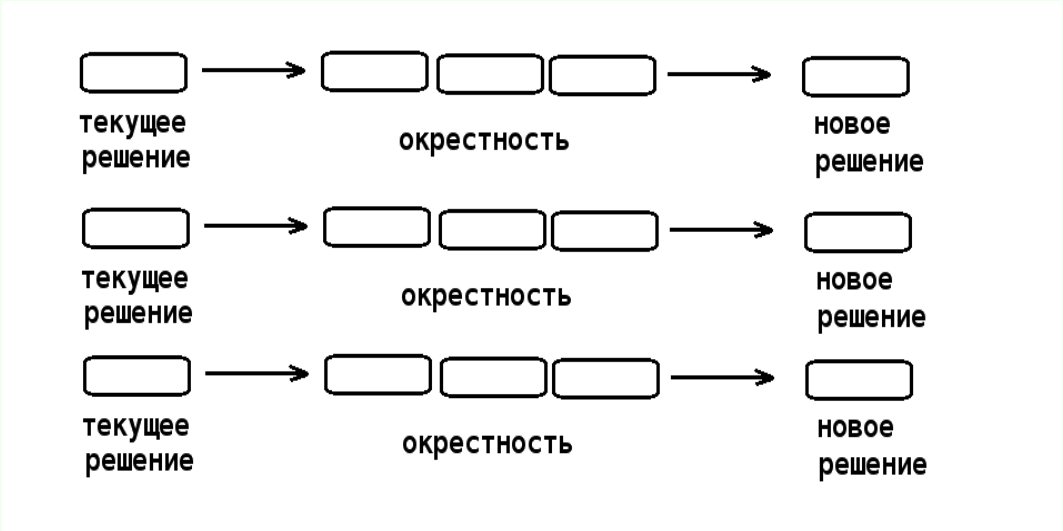
1. Построить начальное решение $S^{(0)}$.
2. На каждой итерации t выполнять:

2.1. Сгенерировать K случайных решений $S'_1, ..., S'_K$ из окрестности текущего решения $S^{(t)}$. Из них выбрать лучшее S^*.

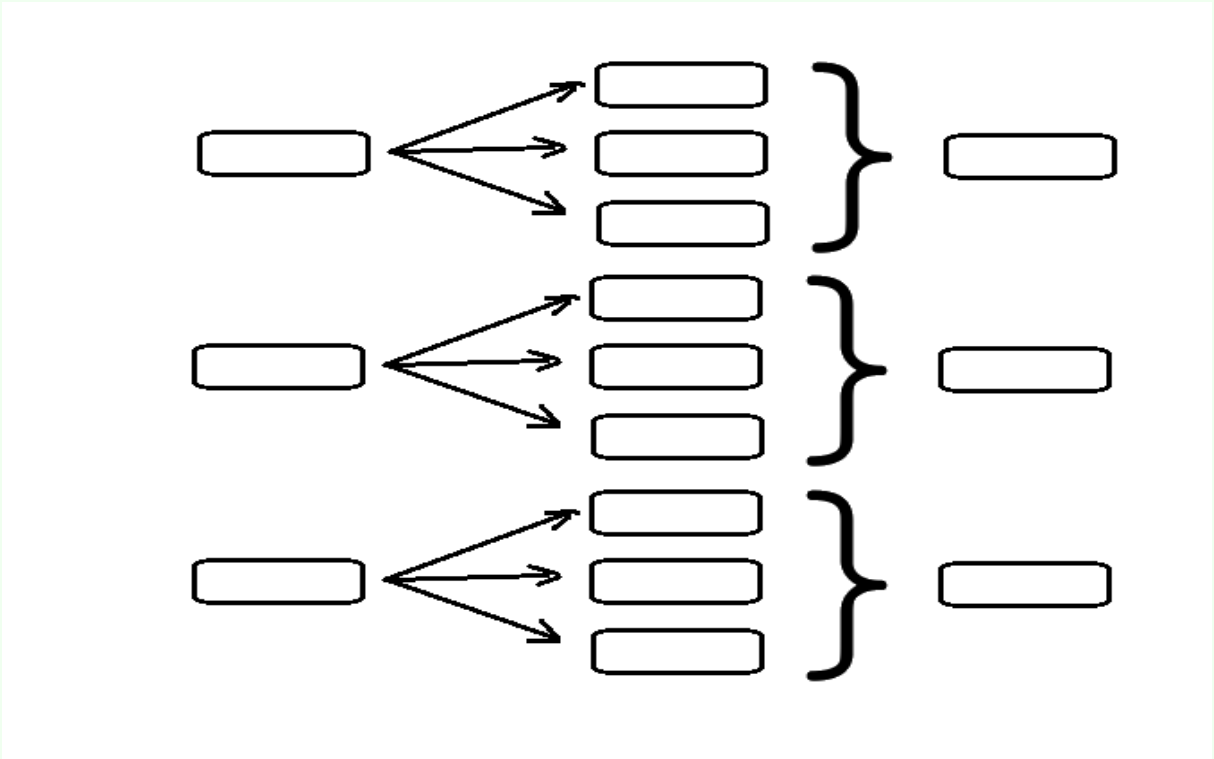
2.2. Если $f(S^*) < f(S^{(t)})$, то $S^{(t+1)} := S^*$, иначе $S^{(t+1)} := S^{(t)}$.

Параллельный (случайный) локальный поиск

2. Локальный поиск с перезапуском

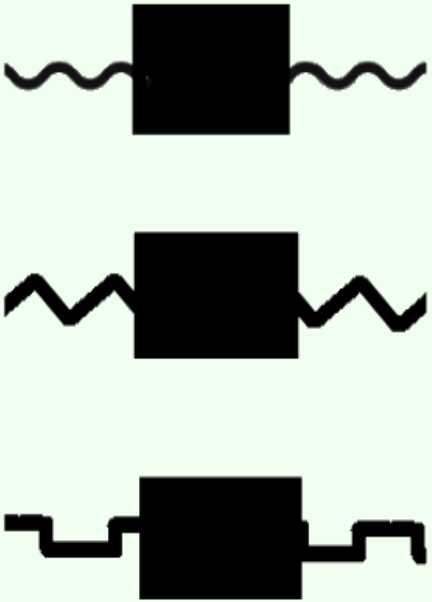


Параллельный локальный поиск с перезапуском



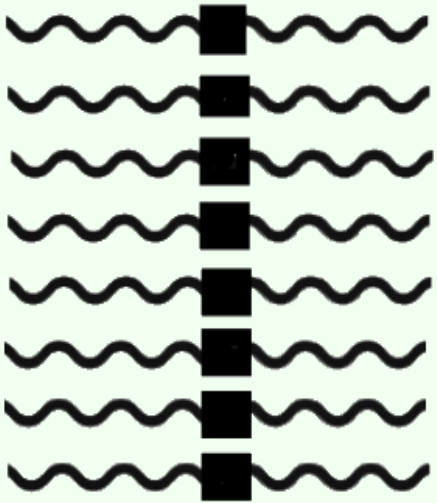
CPU

множественный поток команд,
множественный поток данных

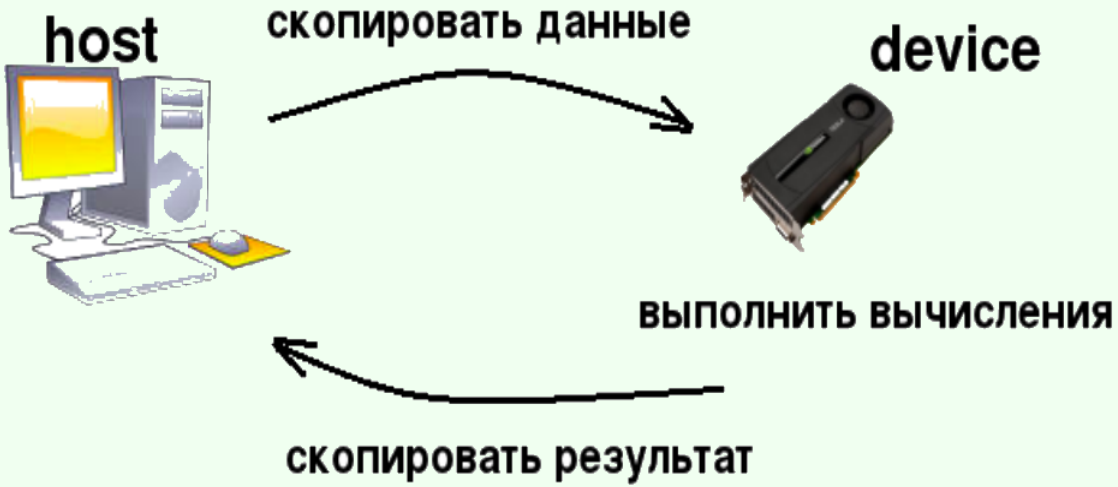


GPU

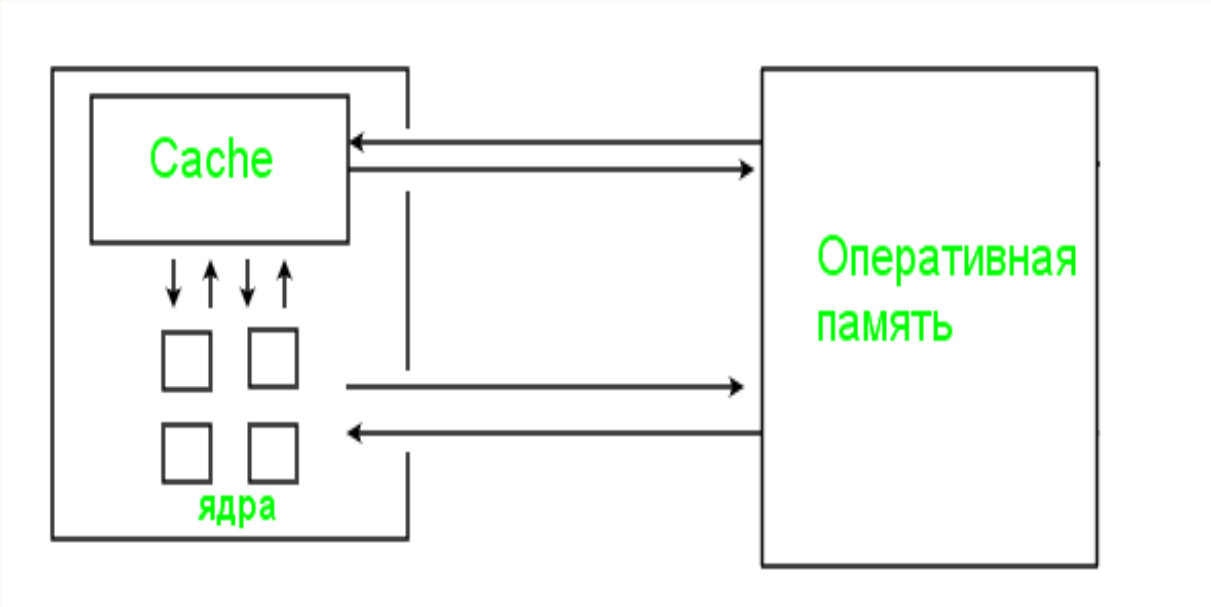
одиночный поток команд,
множественный поток данных



Взаимодействие с центральным процессором



Для достижения высокой скорости необходимо как можно меньше обращаться к оперативной памяти и максимально использовать кэш.

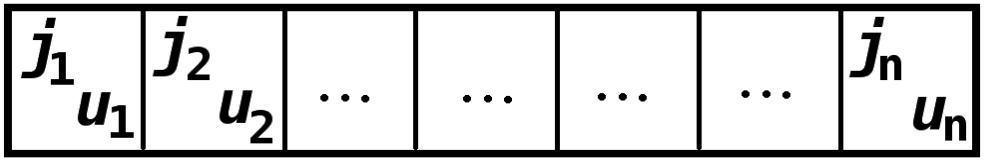


Представление решений

$(j_1, j_2, ..., j_n)$ – перестановка работ
$(u_1, u_2, ..., u_n)$ – упорядоченный список машин, например

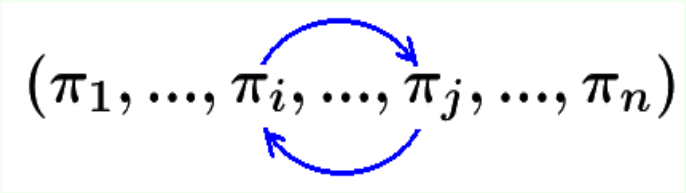
(1,1,2,2,2,3,3,3,3).

Решение хранится в виде массива `int[n]`. Для уменьшения обращений к памяти, оба значения (j_k, u_k) записываются в одну ячейку (в старшие и младшие 16 бит).

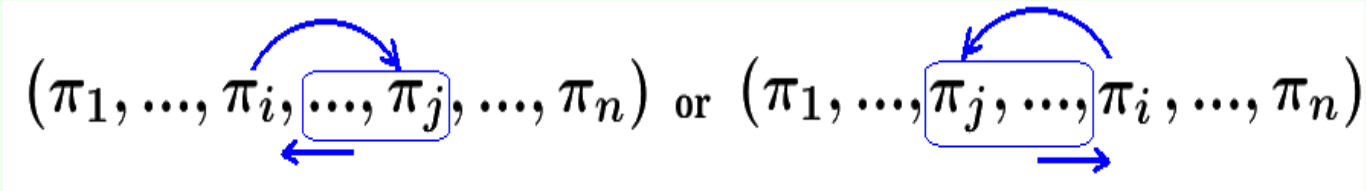


Операторы мутации

1. **Swap**



2. **Insertion**



Тестовый пример

500 работ, 150 машин.

«Популяция» из 64 решений, для каждого создаются 512 потомков, используется только оператор Insertion. Таким образом, на каждой итерации LS обрабатываются 30720 решений. LS запускается на 100000 итераций.

Время выполнения.

CPU INTEL Xeon 4210: **930** сек.

CPU AMD EPYC 7502: **794** сек.

GPU Tesla V100 : **66** сек.

Ускорение в 12 - 14 раз.

Тестовый пример

500 работ, 150 машин.

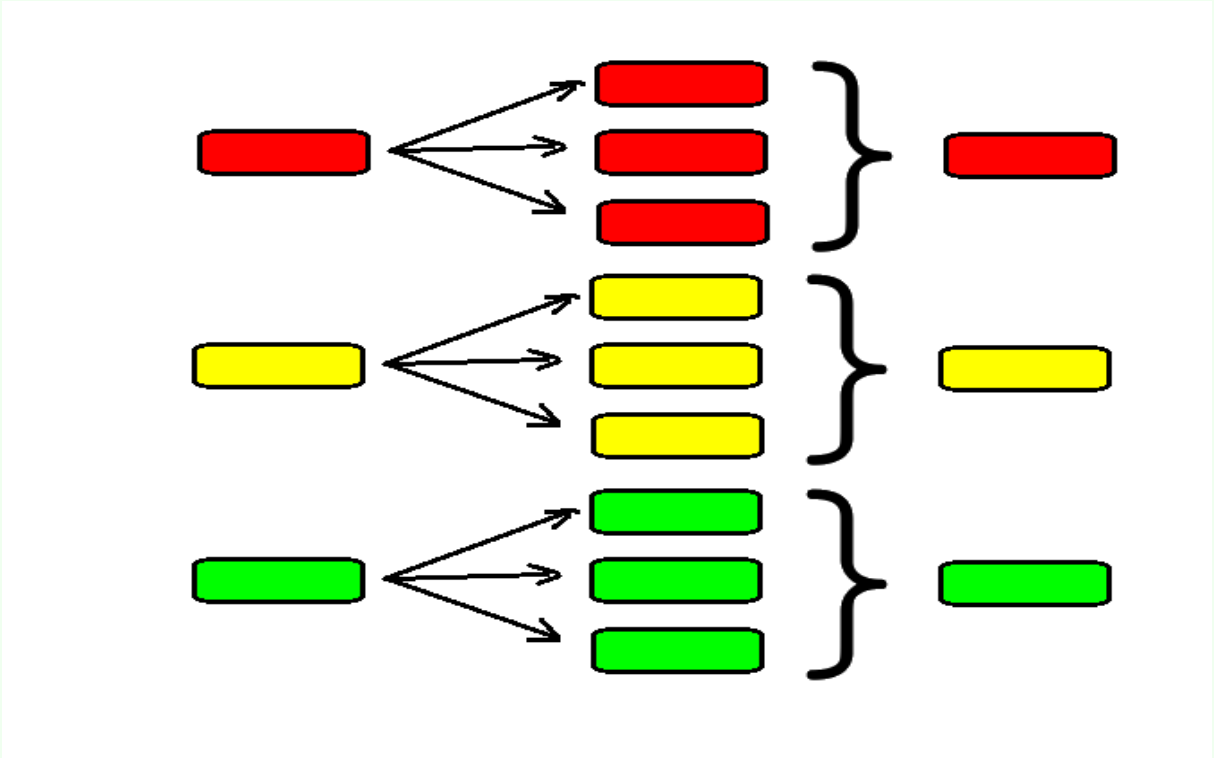
Вычисления на GPU без параллельности. Одно родительское решение, один потомок, 10 млн итераций.

Время выполнения.

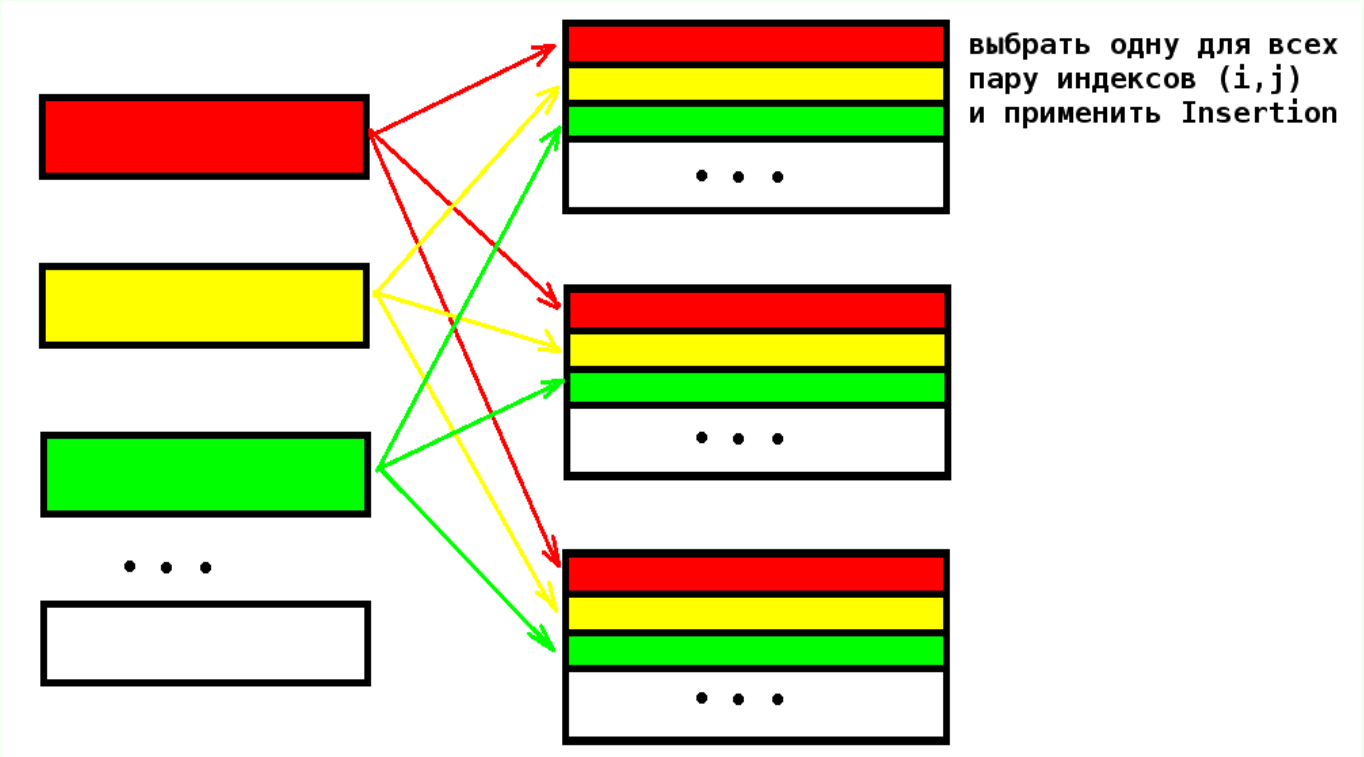
CPU AMD EPYC 7502: **19** сек.

GPU Tesla V100 : **2480** сек.

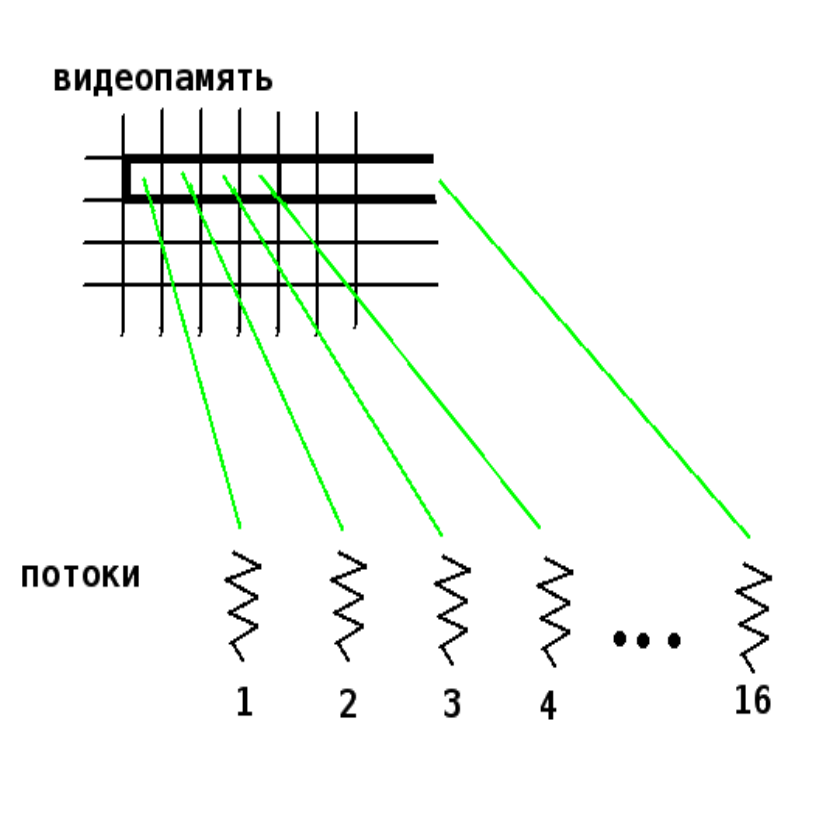
Параллельный локальный поиск



Другая группировка потомков

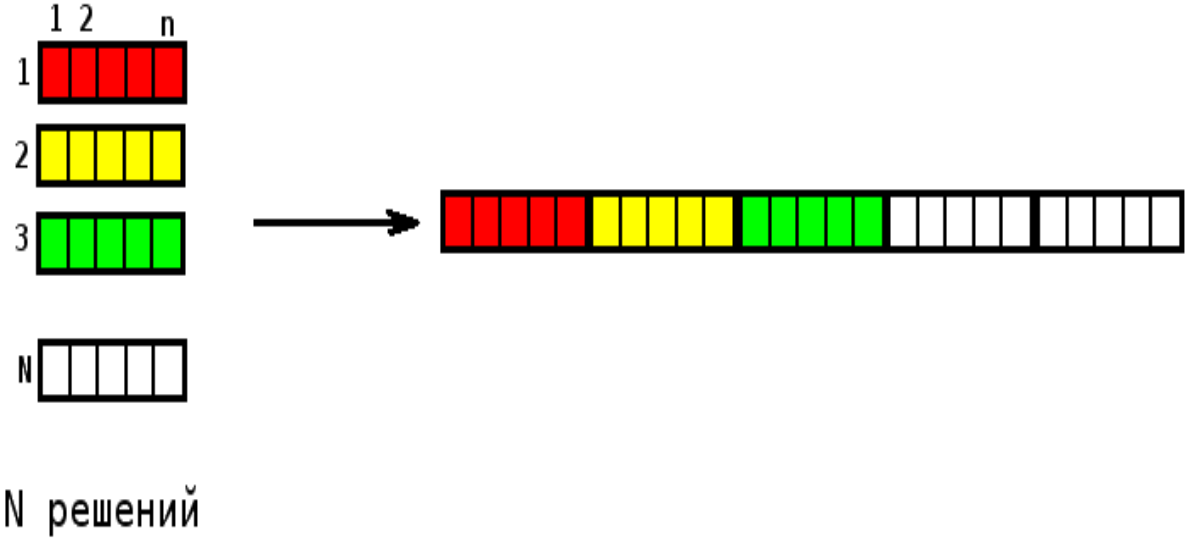


Объединение запросов к памяти (memory coalescing)



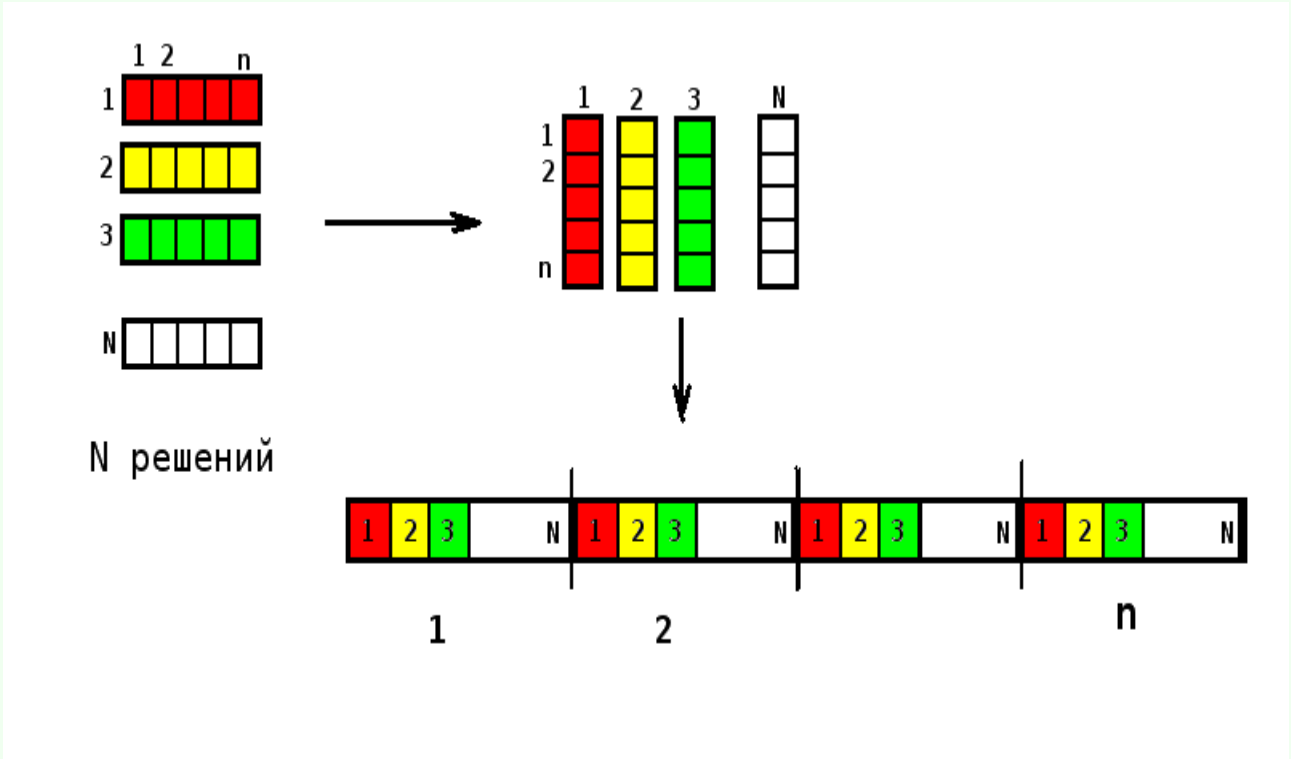
Хранение множества решений в памяти

Неправильный способ.



Хранение множества решений в памяти

Правильный способ.



Время выполнения на GPU: **9** сек. Ускорение по сравнению с CPU примерно в 100 раз.

Сравнение с Gurobi

Тестовые примеры с количеством работ $n = 500$ и машин $m = 150$.

Параметры алгоритма: число родительских решений $N = 256$, число потомков $K = 512$, количество итераций – сто тысяч.

Время выполнения с такими настройками составляло 260 секунд. Для каждой задачи было сделано 10 независимых запусков алгоритма.

Gurobi запускался с ограничением по времени счета в пять часов и с использованием до восьми ядер CPU.

Результаты

№	Gurobi LB	Gurobi UB	LS min	LS avg	LS max
1	121.6	402.1 (230%)	122.5 (0.2%)	122.7	123.4
2	128.8	128.9 (0.1%)	129.4 (0.5%)	129.4	129.4
3	140.3	480.8 (242%)	141.1 (1.3%)	142.2	144.1
4	84.2	84.3 (0.1%)	84.7 (0.6%)	84.7	84.7
5	70.5	70.6 (0.1%)	71.1 (3.2%)	72.8	74.1
6	119.6	119.7 (0.1%)	120.2 (0.8%)	120.6	121.3
7	124.2	124.3 (0.1%)	124.8 (0.6%)	124.9	125.8
8	70.4	70.4 (0%)	70.9 (2.1%)	71.9	72.9
9	191.7	520.1 (171%)	193.5 (1.6%)	194.8	196.6
10	68.9	453.4 (558%)	69.7 (1.4%)	69.9	71.7

Заключение

- Вычисления на GPU могут успешно применяться при решении задач оптимизации в производстве.
- В сравнении с известными аналогами рассмотренный подход позволяет получать похожие по качеству решения, но за значительно меньшее время.
- Особо стоит отметить простоту реализации алгоритма, что позволяет использовать его для широкого класса задач.
- Представляется перспективным опробовать данный подход в комбинации с другими эвристиками, такими как генетический алгоритм или поиск с запретами.

Спасибо за внимание!