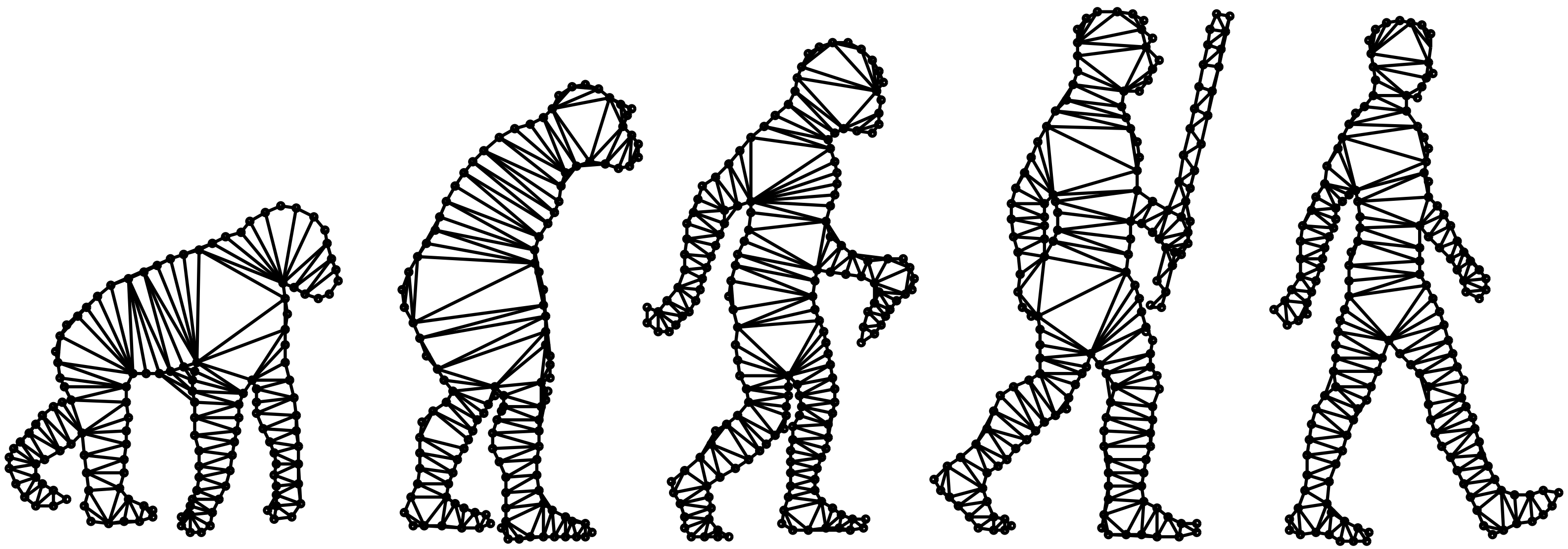
Karlsruhe Institute of Technology

# Evolutionary Hypergraph Partitioning

**Presentation · December 12, 2017**
**Robin Andre**

INSTITUTE OF THEORETICAL INFORMATICS ·

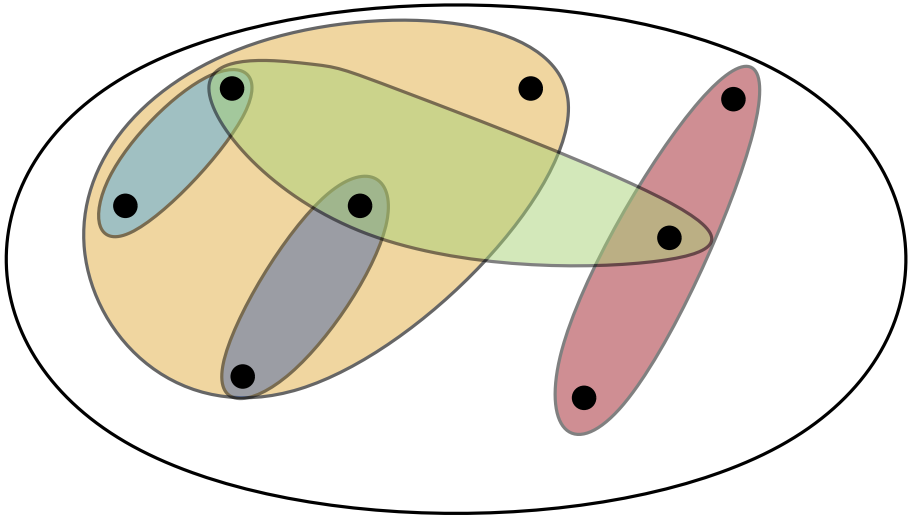


# Hypergraph Partitioning

## Hypergraph Partitioning

A $k$ partition of a hypergraph is $H = V_1 \uplus V_2 \uplus \ldots \uplus V_k$

A partition is balanced if $\forall\ 1 \leq i \leq k\ :\ c(V_i) \leq (1+\epsilon)\lceil \frac{c(V)}{k} \rceil$



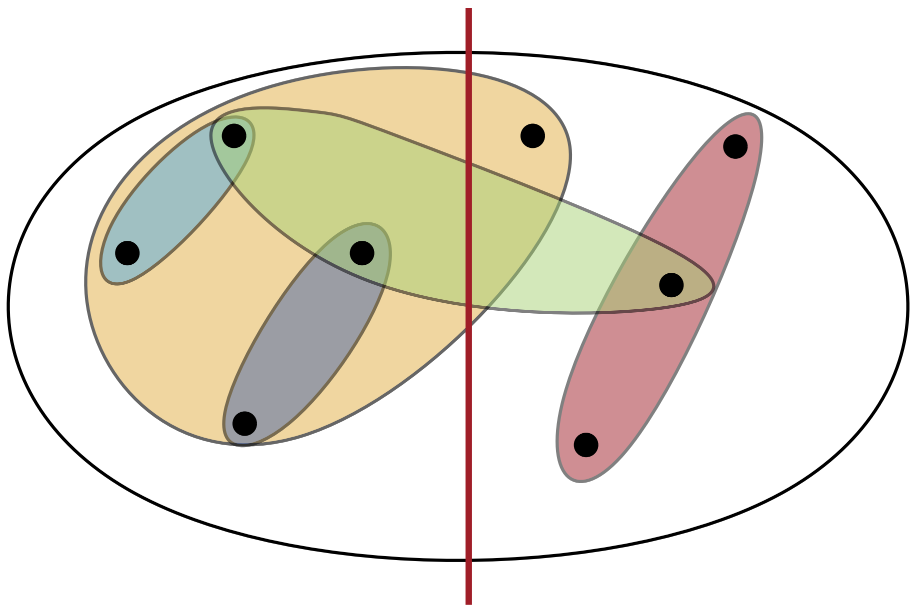
# Hypergraph Partitioning

## Hypergraph Partitioning

A $k$ partition of a hypergraph is $H = V_1 \cup V_2 \cup \ldots \cup V_k$

A partition is balanced if $\forall\ 1 \le i \le k\ :\ c(V_i) \le (1+\epsilon)\lceil \frac{c(V)}{k} \rceil$

$k = 2;\ cut = 2;\ (\lambda - 1) = 2$



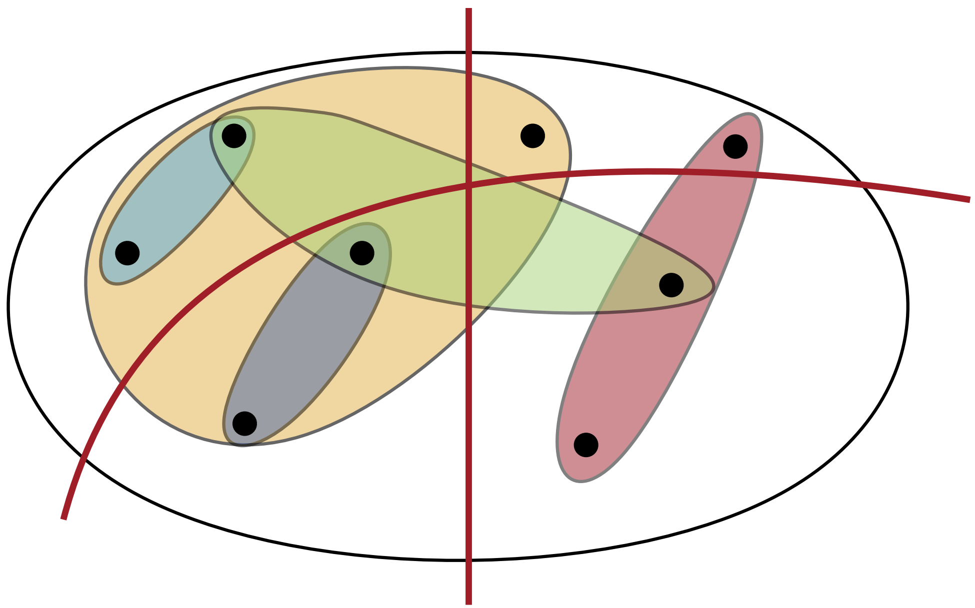
# Hypergraph Partitioning

## Hypergraph Partitioning

A $k$ partition of a hypergraph is $H = V_1 \mathbin{\dot\cup} V_2 \mathbin{\dot\cup} \ldots \mathbin{\dot\cup} V_k$

A partition is balanced if $\forall\ 1 \leq i \leq k\ :\ c(V_i) \leq (1+\epsilon)\lceil \frac{c(V)}{k} \rceil$

$k = 4;\ cut = 3;\ (\lambda - 1) = 5$

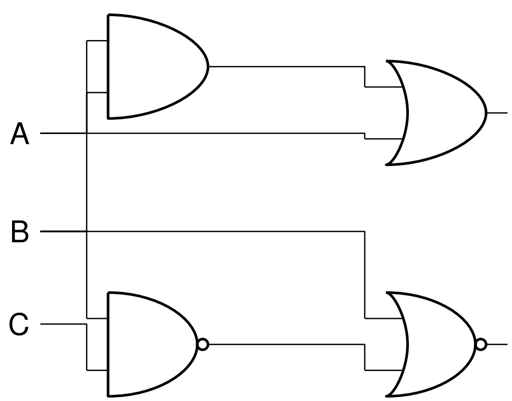


# Motivation



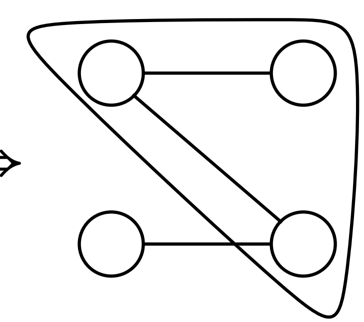
- Hypergraph partitioning is NP-hard
- Many applications benefit from the best possible solution
- Evolutionary Algorithms are generating high quality solutions

VLSI Design

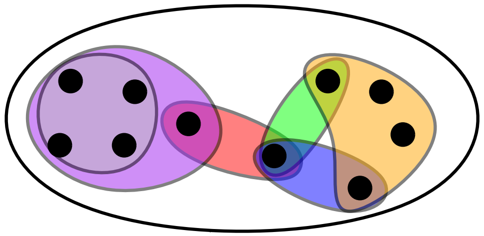



Scientific Computing

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \Rightarrow$$



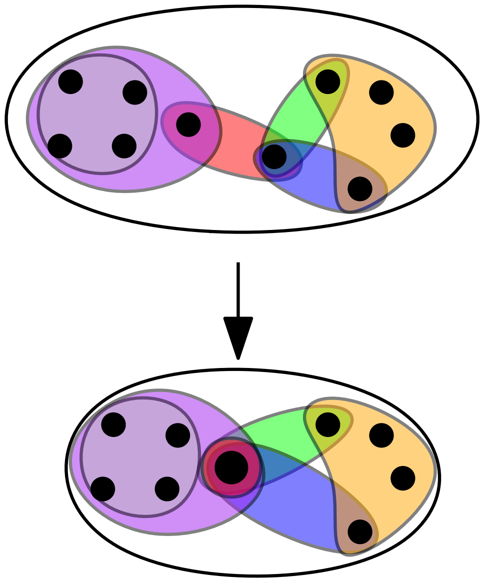
# KaHyPar - Multilevel Partitioner



Coarsening:

- $H$ is reduced to a smaller problem $H_C$
- $n$-level coarsening
- $H_C$ should be sufficiently small

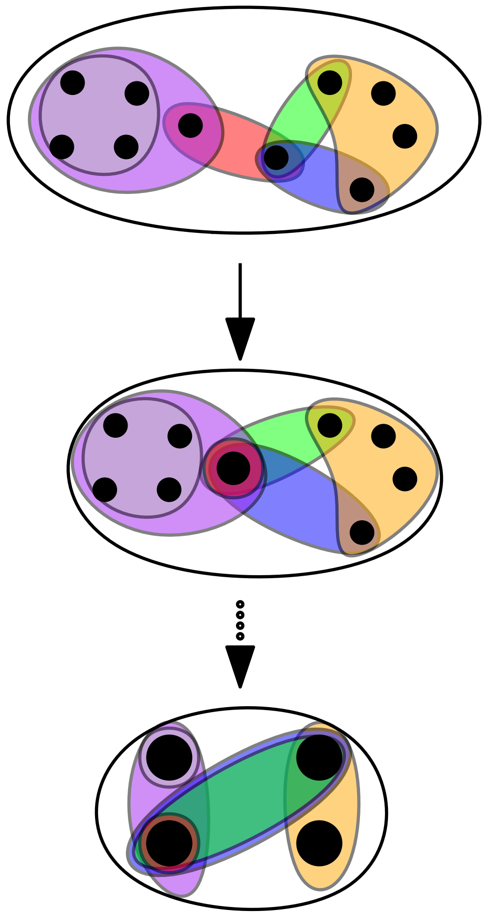
# KaHyPar - Multilevel Partitioner



Coarsening:

- $H$ is reduced to a smaller problem $H_C$
- $n$-level coarsening
- $H_C$ should be sufficiently small

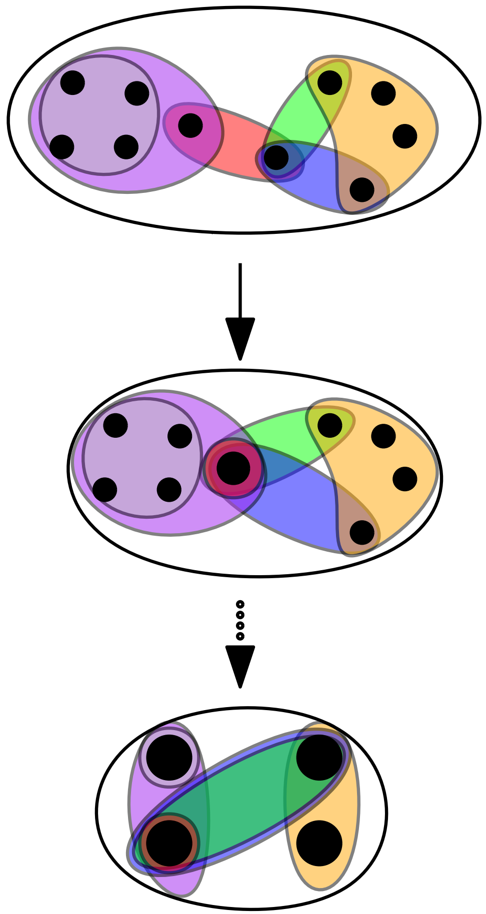
# KaHyPar - Multilevel Partitioner



Coarsening:

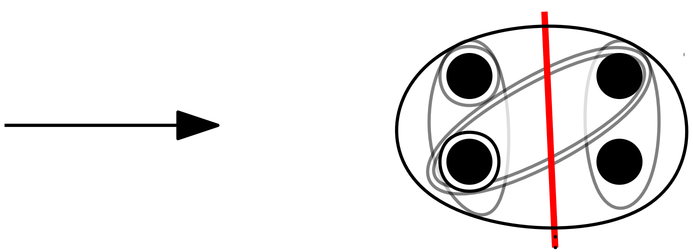
- *H* is reduced to a smaller problem $H_C$
- *n*-level coarsening
- $H_C$ should be sufficiently small

# KaHyPar - Multilevel Partitioner

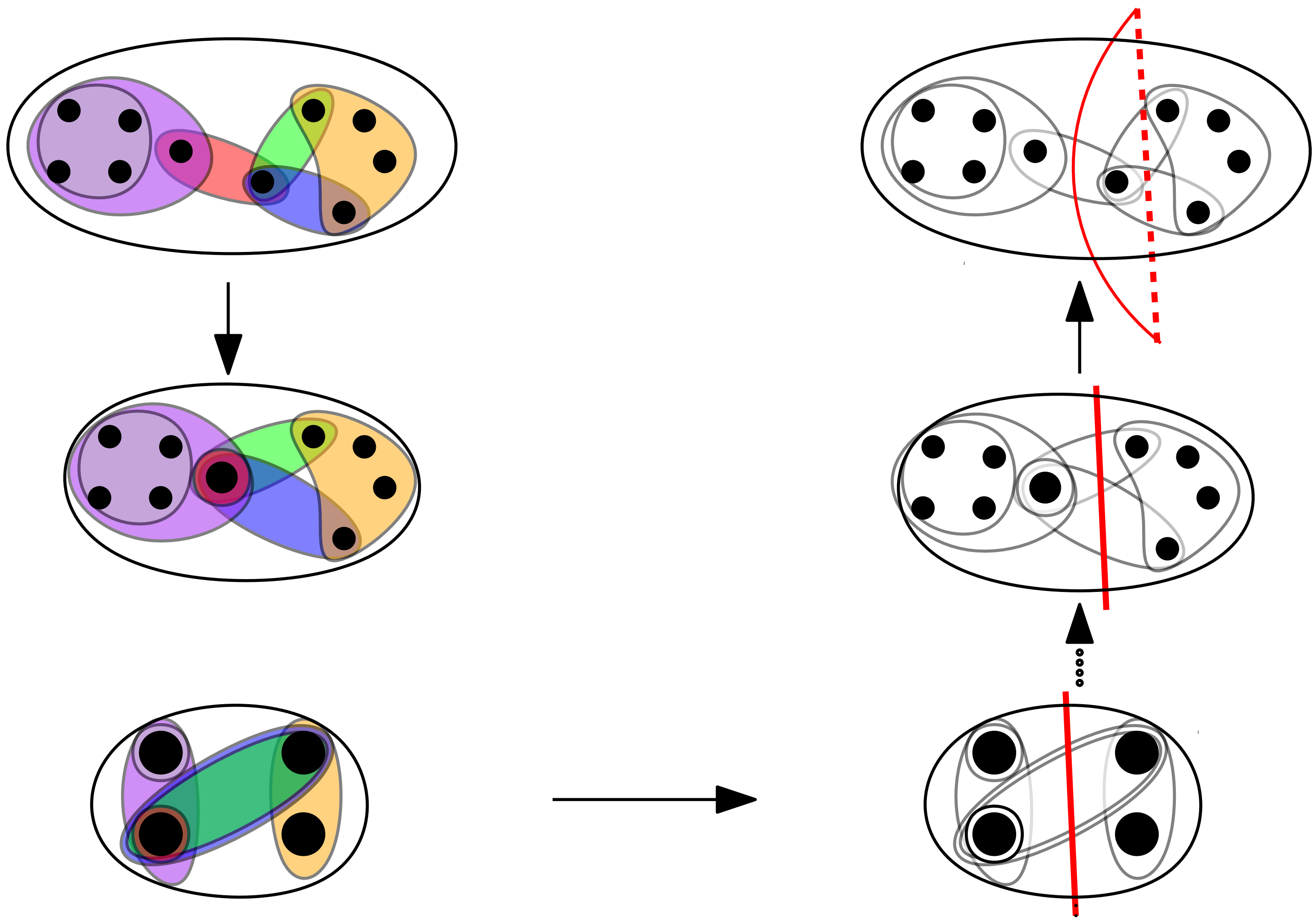


Initial Partitioning:

- An algorithm generates an Initial $k$ partition for $H_C$

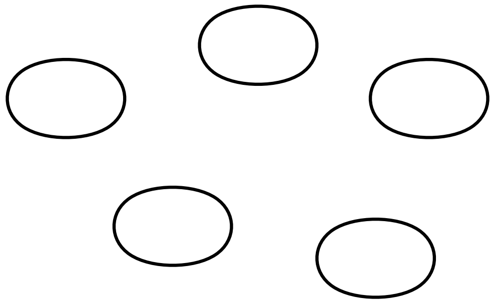


# KaHyPar - Multilevel Partitioner



# Evolutionary Process

- Choose individuals for recombination

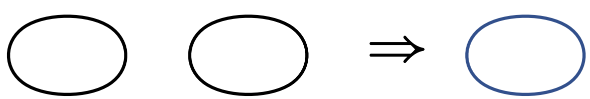


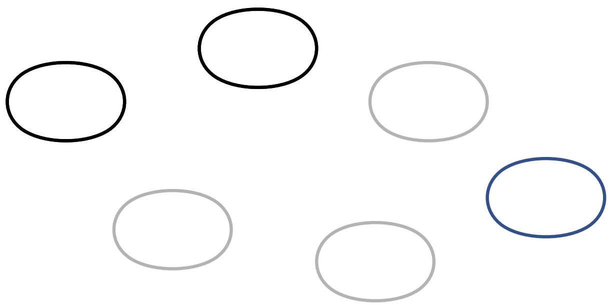
# Evolutionary Process

- Choose individuals for recombination

# Evolutionary Process

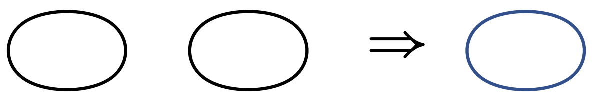
- Choose individuals for recombination
- Generate offspring $O$

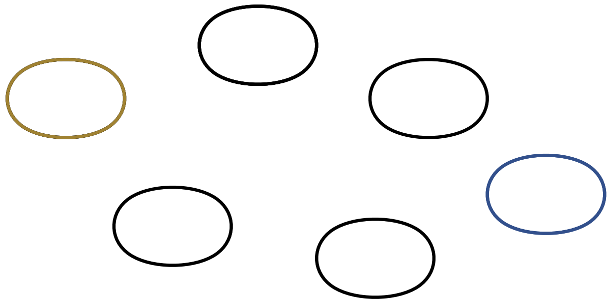




# Evolutionary Process

- Choose individuals for recombination
- Generate offspring $O$
- Perform mutations $M$

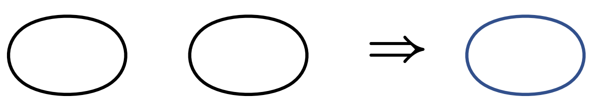


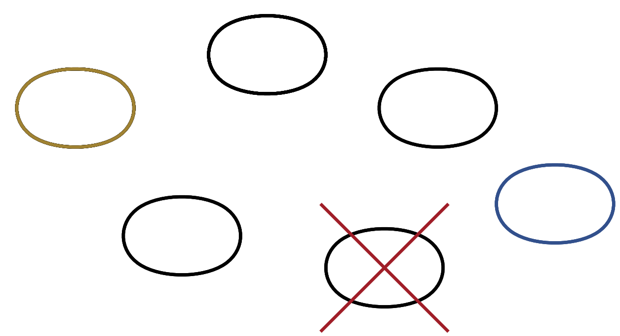


# Evolutionary Process



- Choose individuals for recombination
- Generate offspring *O*
- Perform mutations *M*
- Select survivors

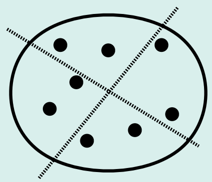




# Evolutionary Algorithm


- *KaHyPar* generates multiple partitions
- dynamic allocation $\delta = 15\%$
- balances time/hypergraph size

# Evolutionary Algorithm

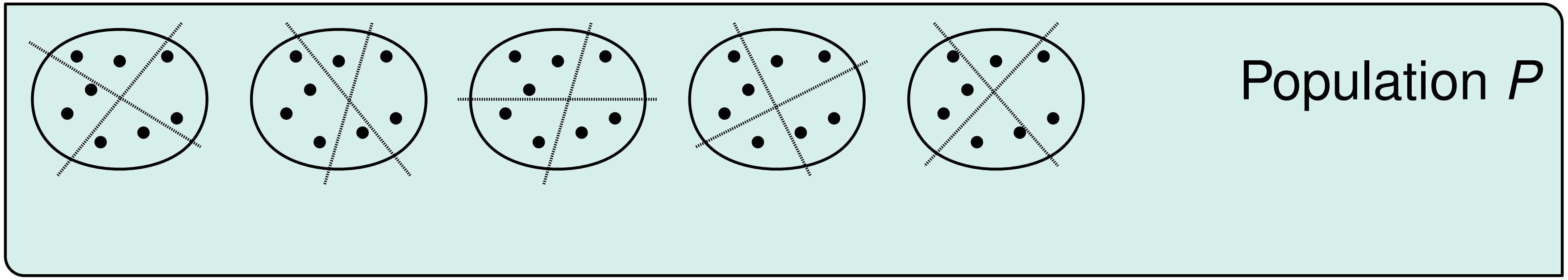


Population $P$

3.1s $time = 100s$ $\longrightarrow$ $\sim$ 5 iterations

- *KaHyPar* generates multiple partitions
- dynamic allocation $\delta = 15\%$
- balances time/hypergraph size

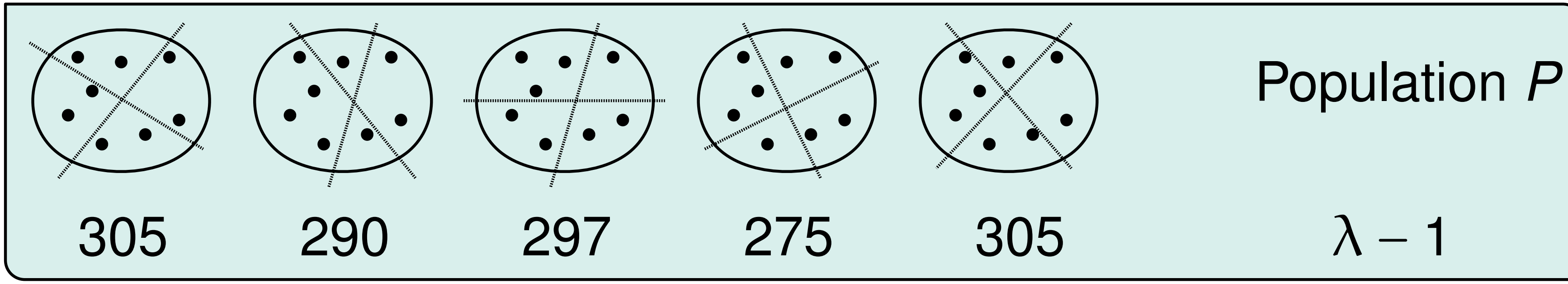
# Evolutionary Algorithm




3.1s $time = 100s$ → $\sim$ 5 iterations

- *KaHyPar* generates multiple partitions
- dynamic allocation $\delta = 15\%$
- balances time/hypergraph size

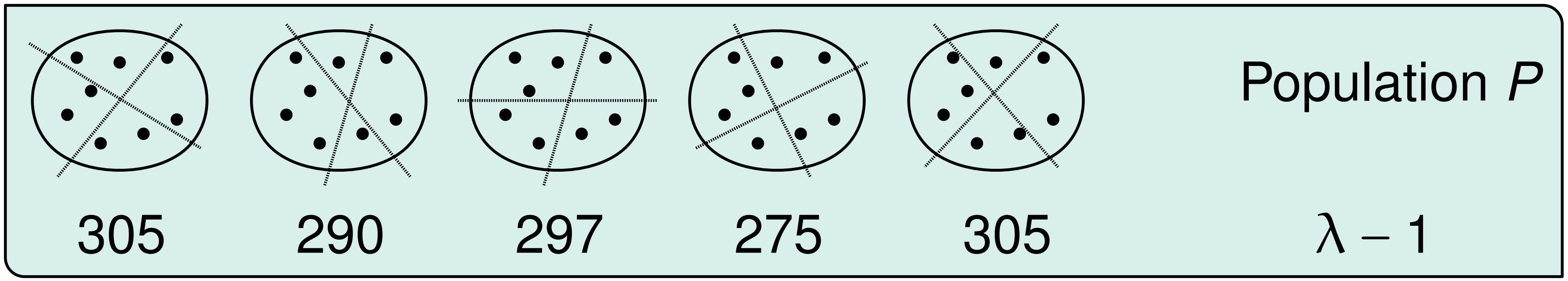
# Evolutionary Algorithm




3.1s $time = 100s$ ⟶ ~ 5 iterations

- *KaHyPar* generates multiple partitions
- dynamic allocation $\delta = 15\%$
- balances time/hypergraph size

# Evolutionary Algorithm

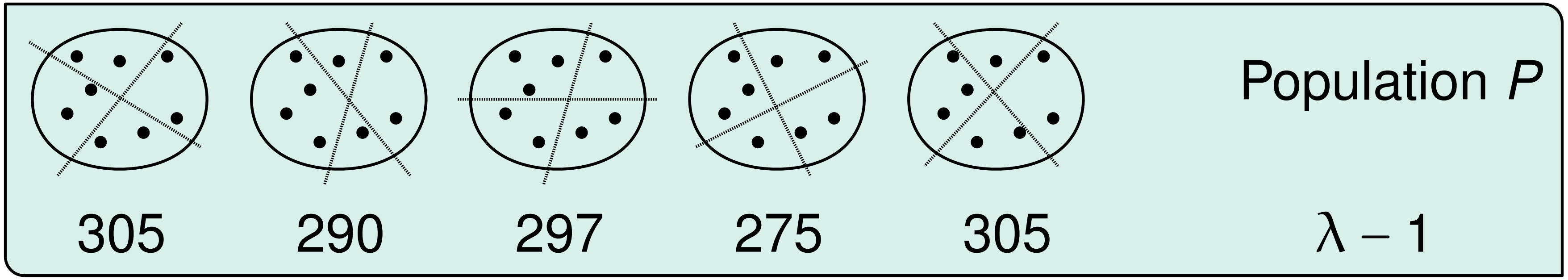



3.1s $time = 100s$ $\longrightarrow$ $\sim$ 5 iterations

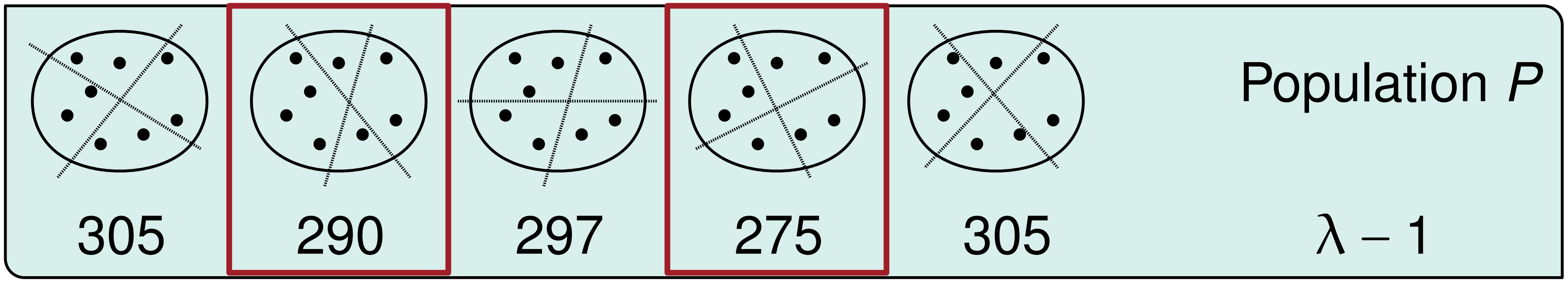
- *KaHyPar* generates multiple partitions
- dynamic allocation $\delta = 15\%$
- balances time/hypergraph size

high quality solutions

# Tournament Selection

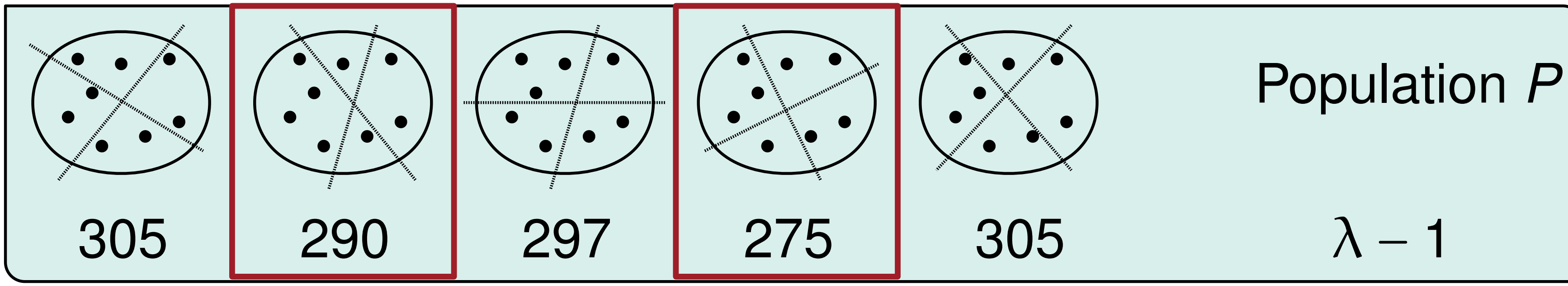


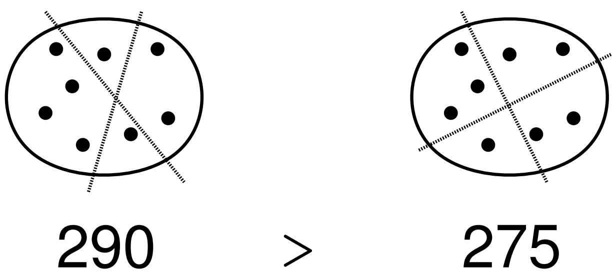
# Tournament Selection




- Select 2 random individuals

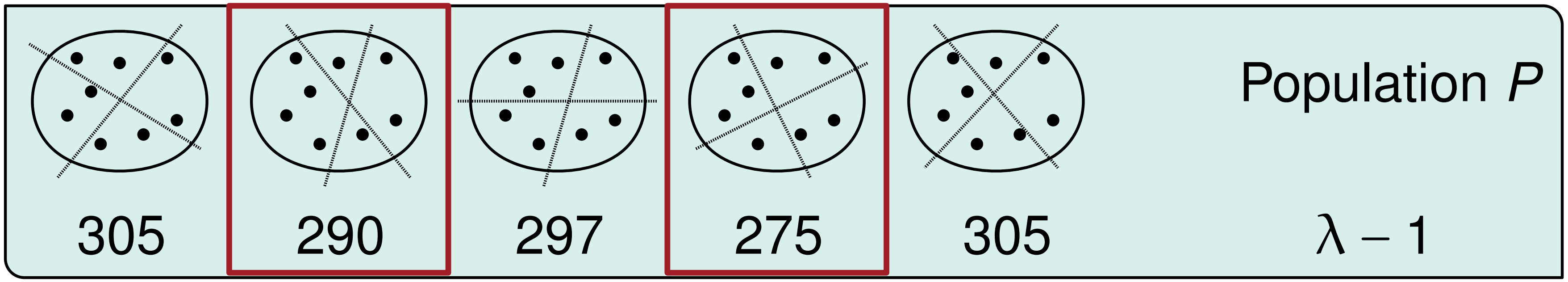
# Tournament Selection

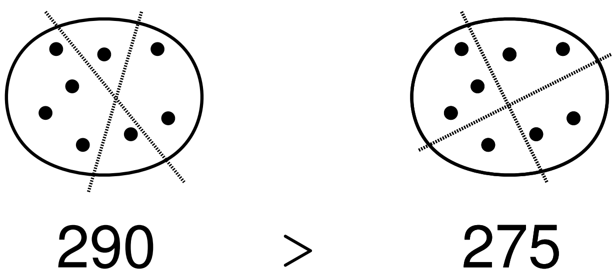



- Select 2 random individuals
- Compare their fitness

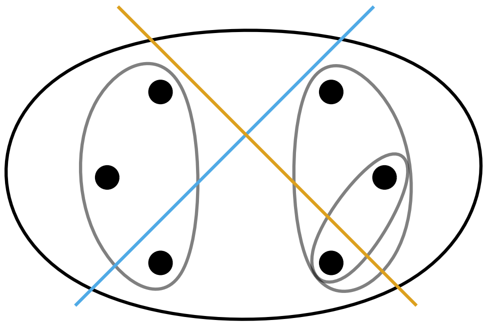
# Tournament Selection

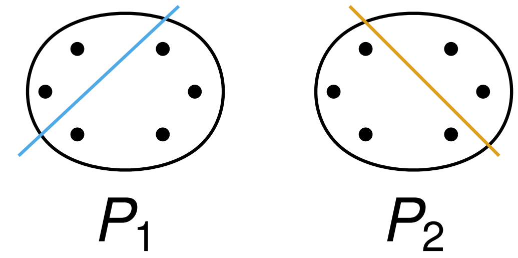



- Select 2 random individuals
- Compare their fitness
- Choose the better individual

# Combine Operator

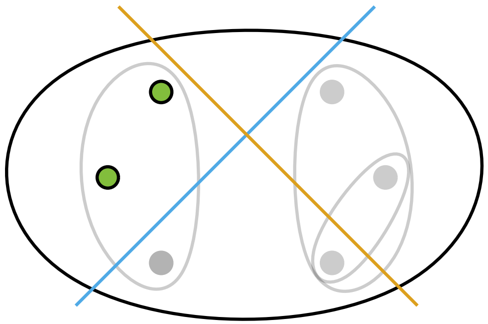


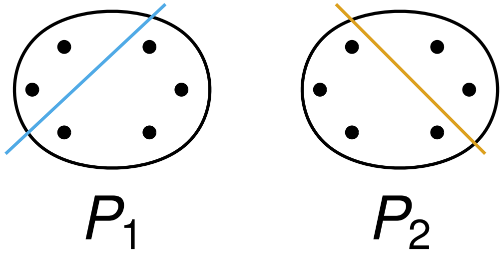



Coarsening:

- contractions must respect $P_1$ & $P_2$
- does not change solution quality

# Combine Operator



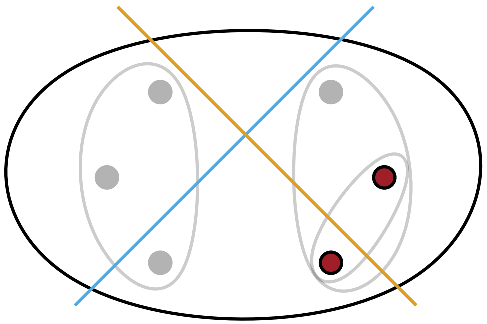


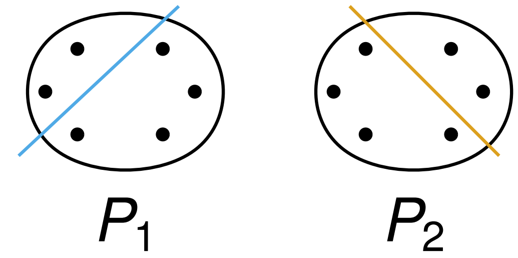

Valid Contraction

Coarsening:

- contractions must respect $P_1$ & $P_2$
- does not change solution quality

# Combine Operator



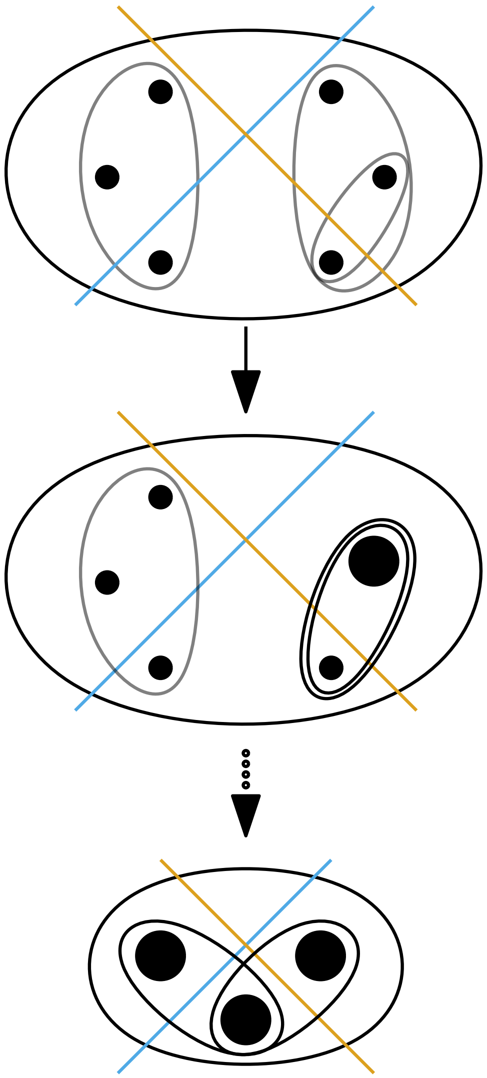


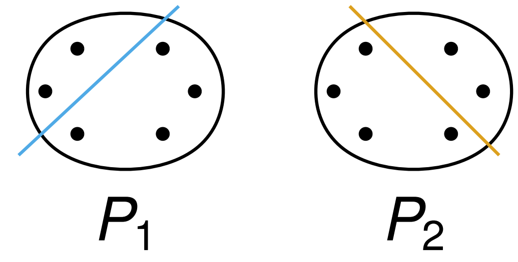

Invalid Contraction

Coarsening:

- contractions must respect $P_1$ & $P_2$
- does not change solution quality

# Combine Operator

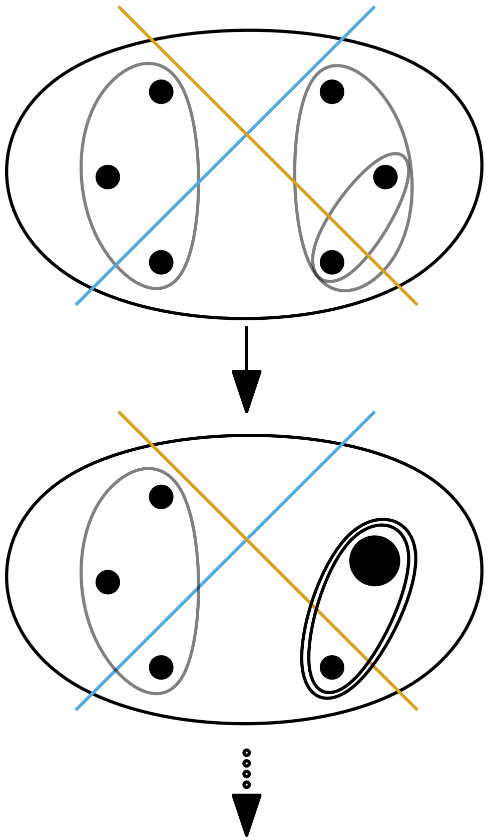


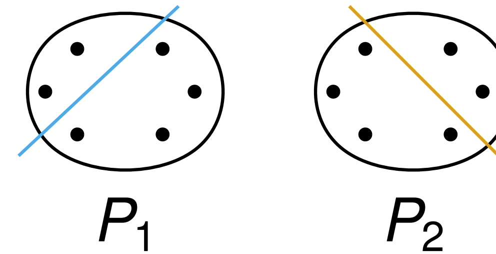



Coarsening:

- contractions must respect $P_1$ & $P_2$
- does not change solution quality

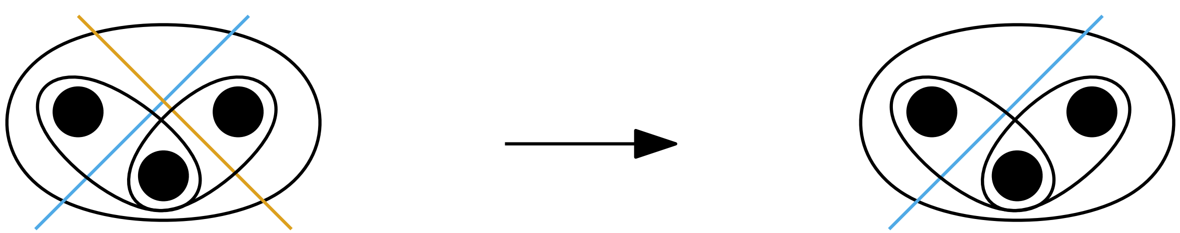
# Combine Operator



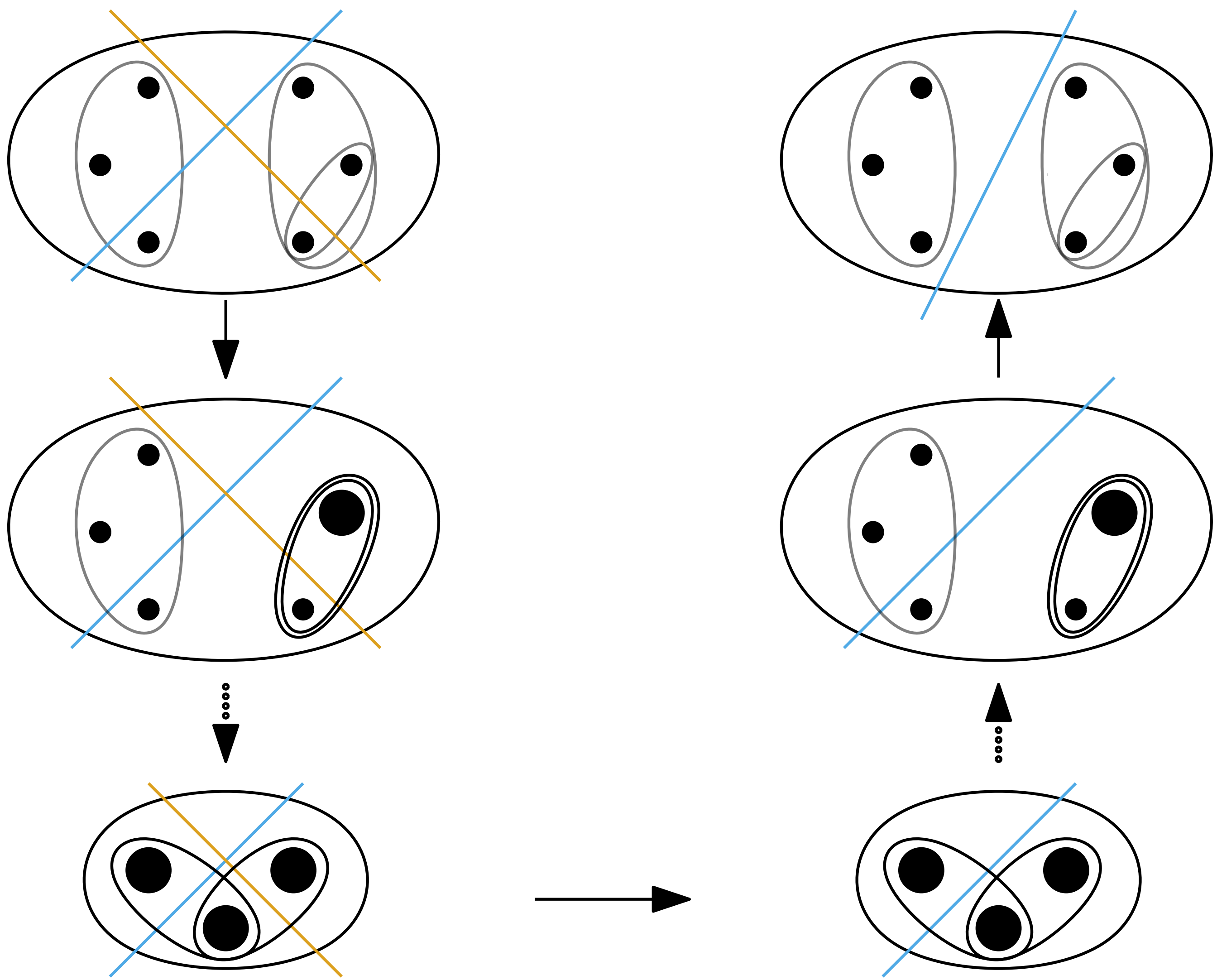



Initial Partitioning:

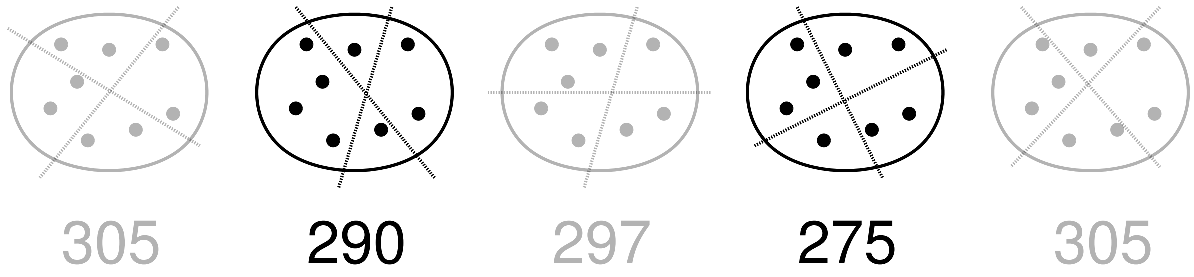
- Use the better parent partition ($P_1$)
- Maintains solution quality



# Combine Operator

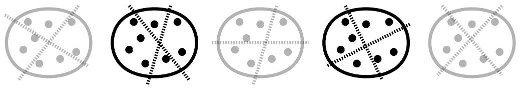


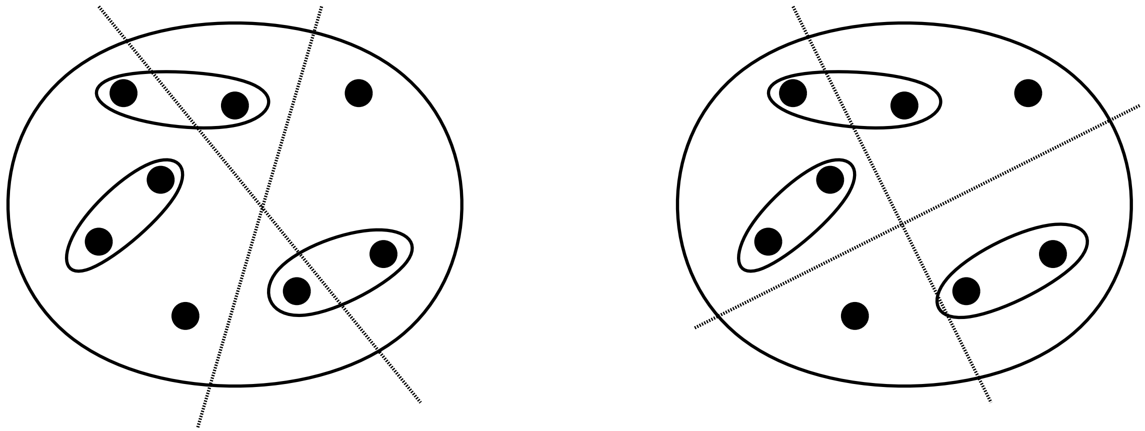
# Edge Frequency Multicombine




- We inspect the $\sqrt{|P|}$ best individuals of $P$
- Each hyperedge $e$ has a counter based on how often $e$ is in a cut

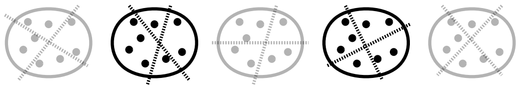
# Edge Frequency Multicombine

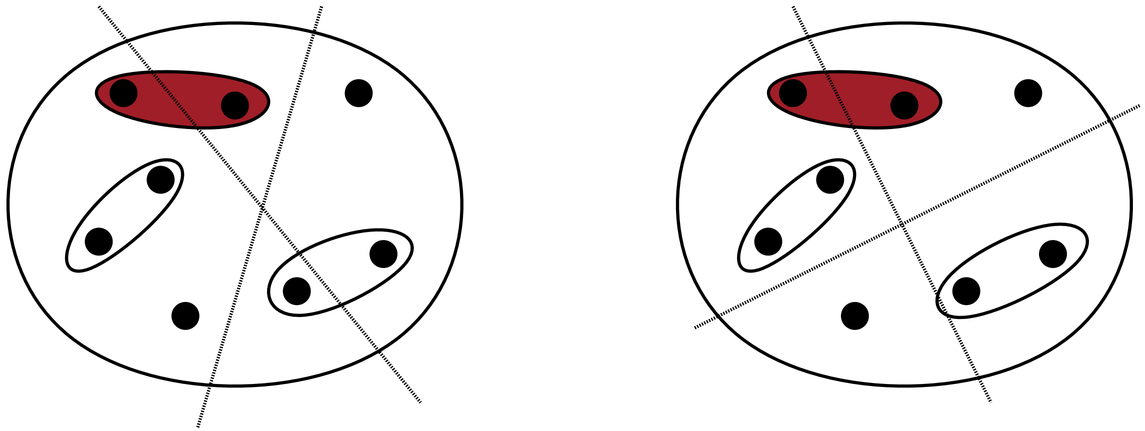




- Frequent edges are most likely cut edges in good solutions
- Contracting frequent edges may be detrimental to solution quality
- Additionally it may limit other contractions

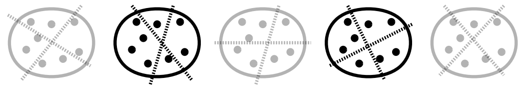
# Edge Frequency Multicombine

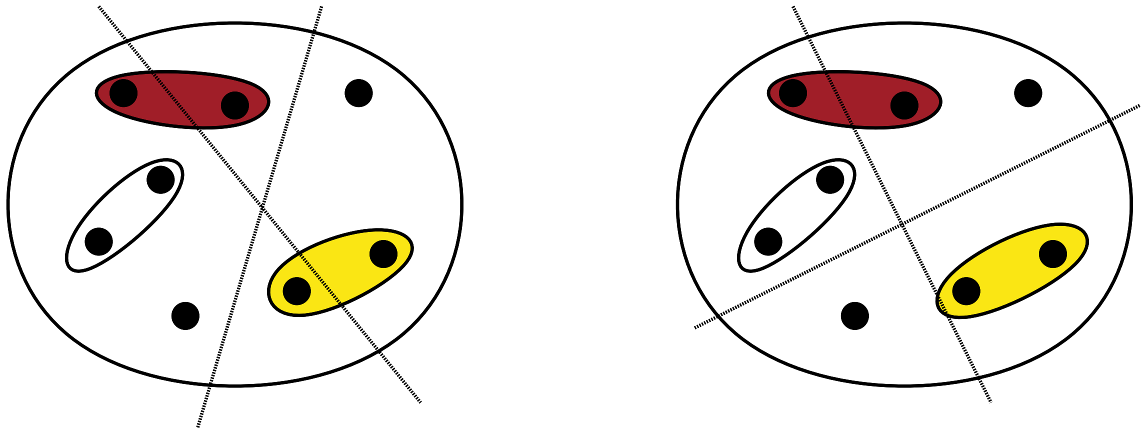




- Frequent edges are most likely cut edges in good solutions
- Contracting frequent edges may be detrimental to solution quality
- Additionally it may limit other contractions

# Edge Frequency Multicombine



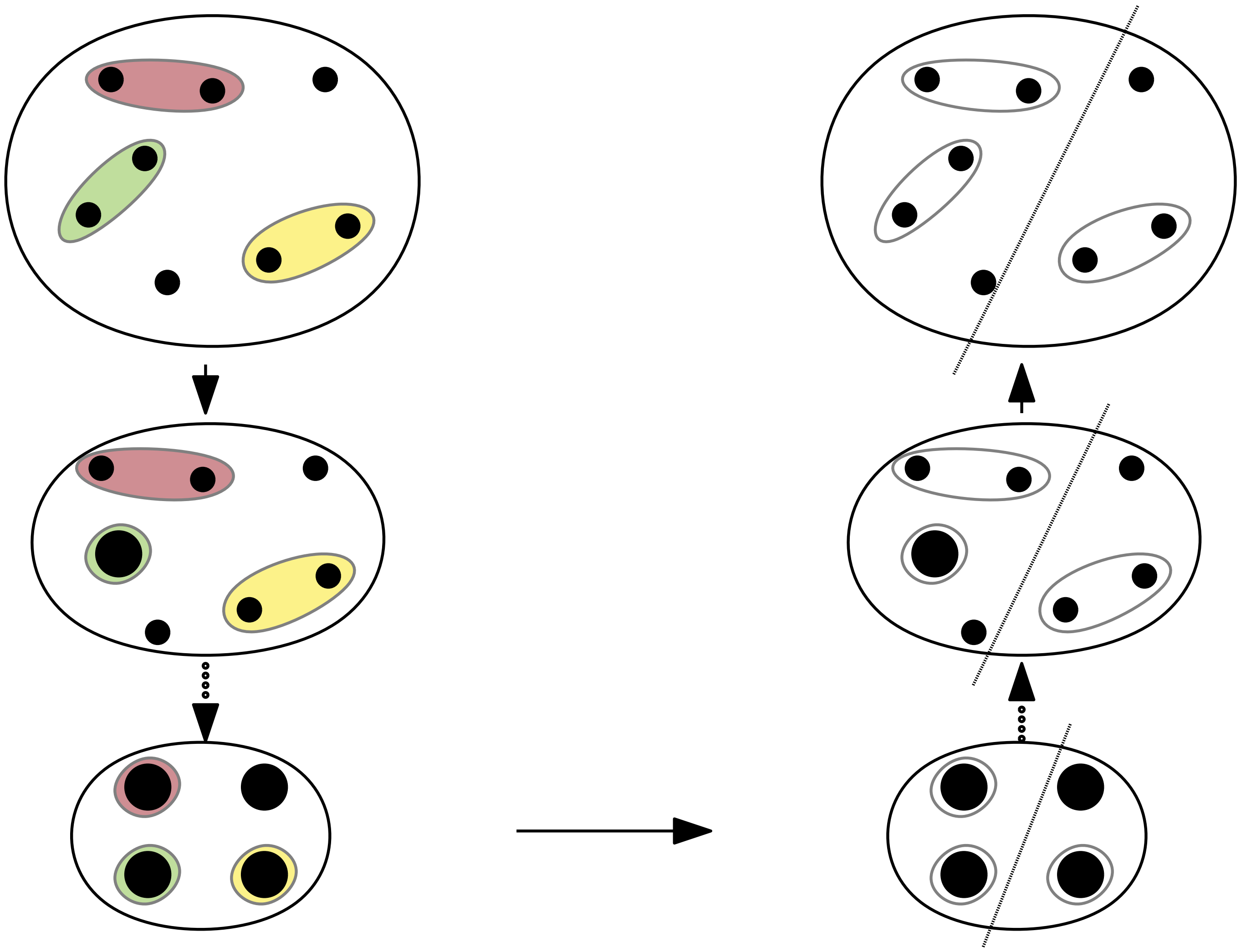


- Frequent edges are most likely cut edges in good solutions
- Contracting frequent edges may be detrimental to solution quality
- Additionally it may limit other contractions

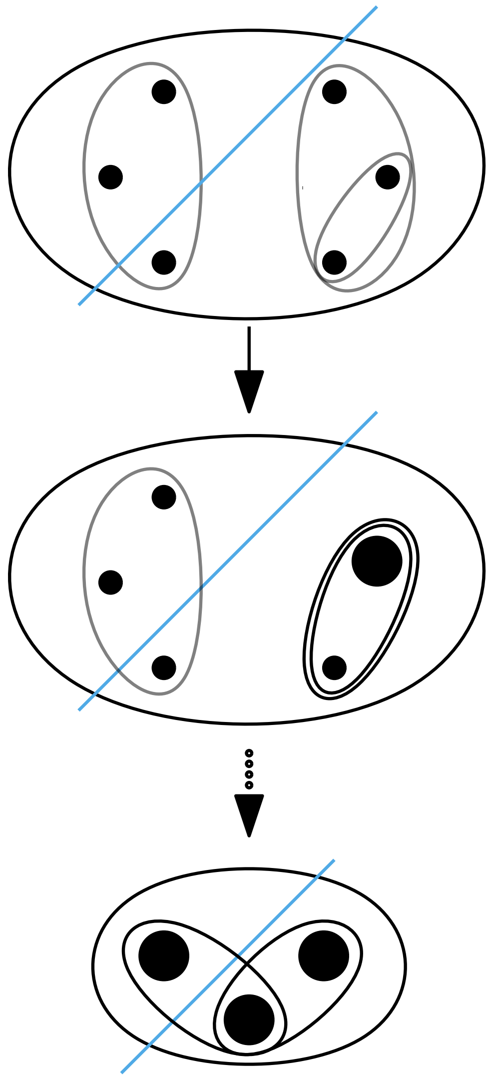
# Edge Frequency Multicombine

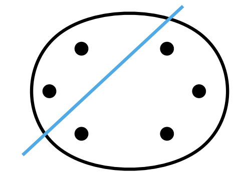
- Frequent edges are most likely cut edges in good solutions
- Contracting frequent edges may be detrimental to solution quality
- Additionally it may limit other contractions

# Edge Frequency Multicombine



# V-Cycle (+ New Initial Partitioning)



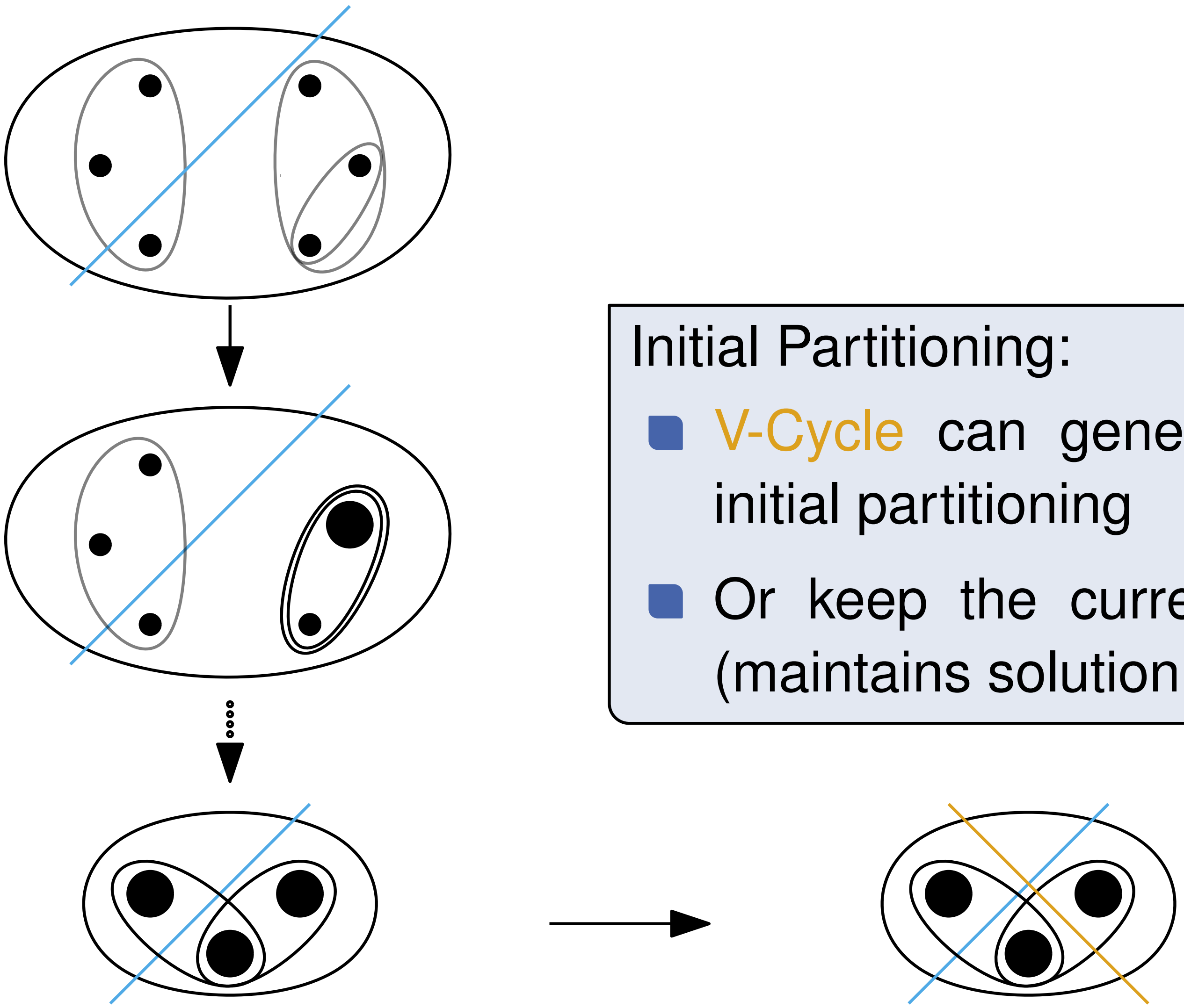



Coarsening:

- Contractions must respect $P$
- Does not change solution quality

# V-Cycle (+ New Initial Partitioning)

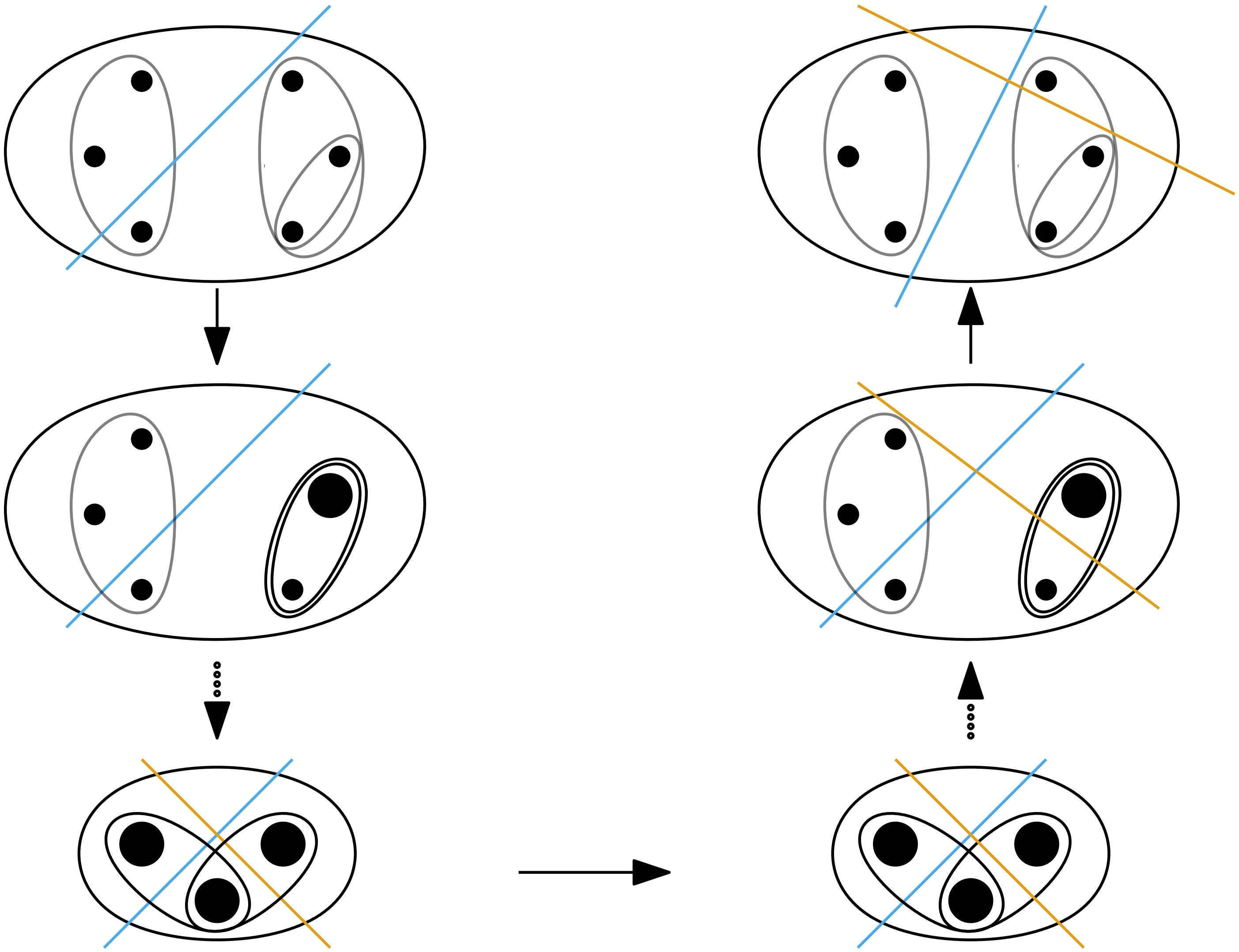
Karlsruhe Institute of Technology



Initial Partitioning:

- V-Cycle can generate a new initial partitioning
- Or keep the current partition (maintains solution quality)

# V-Cycle (+ New Initial Partitioning)



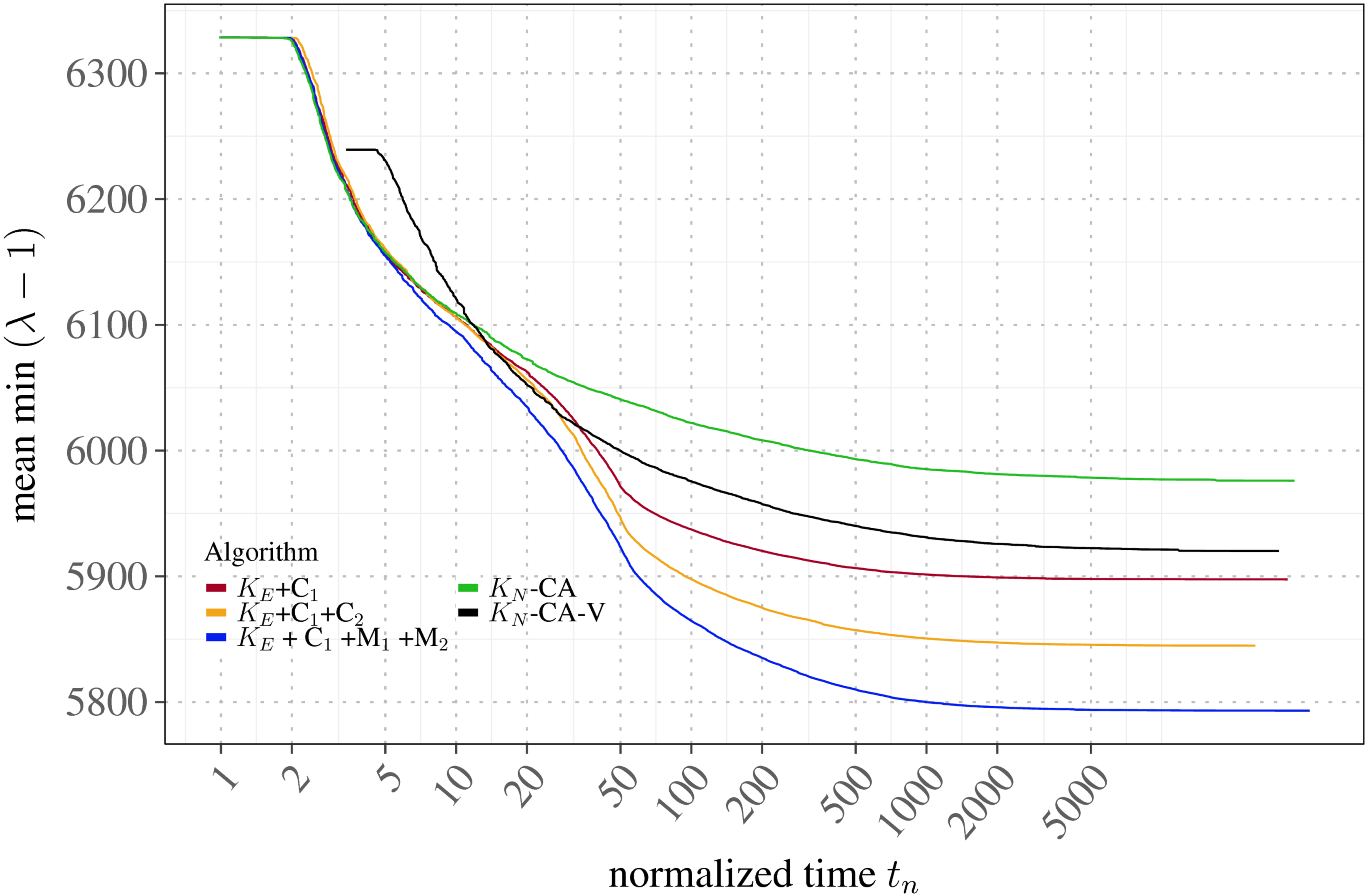
# Experimental Setup

Karlsruhe Institute of Technology

- $k = \{2, 4, 8, 16, 32, 64, 128\}; \epsilon = 0.03$
- 90 hypergraphs (Sparse Matrices, SAT instances, Routability & Circuits)
- Comparison against repeated $KaHyPar$-CA & $KaHyPar$-CA-V

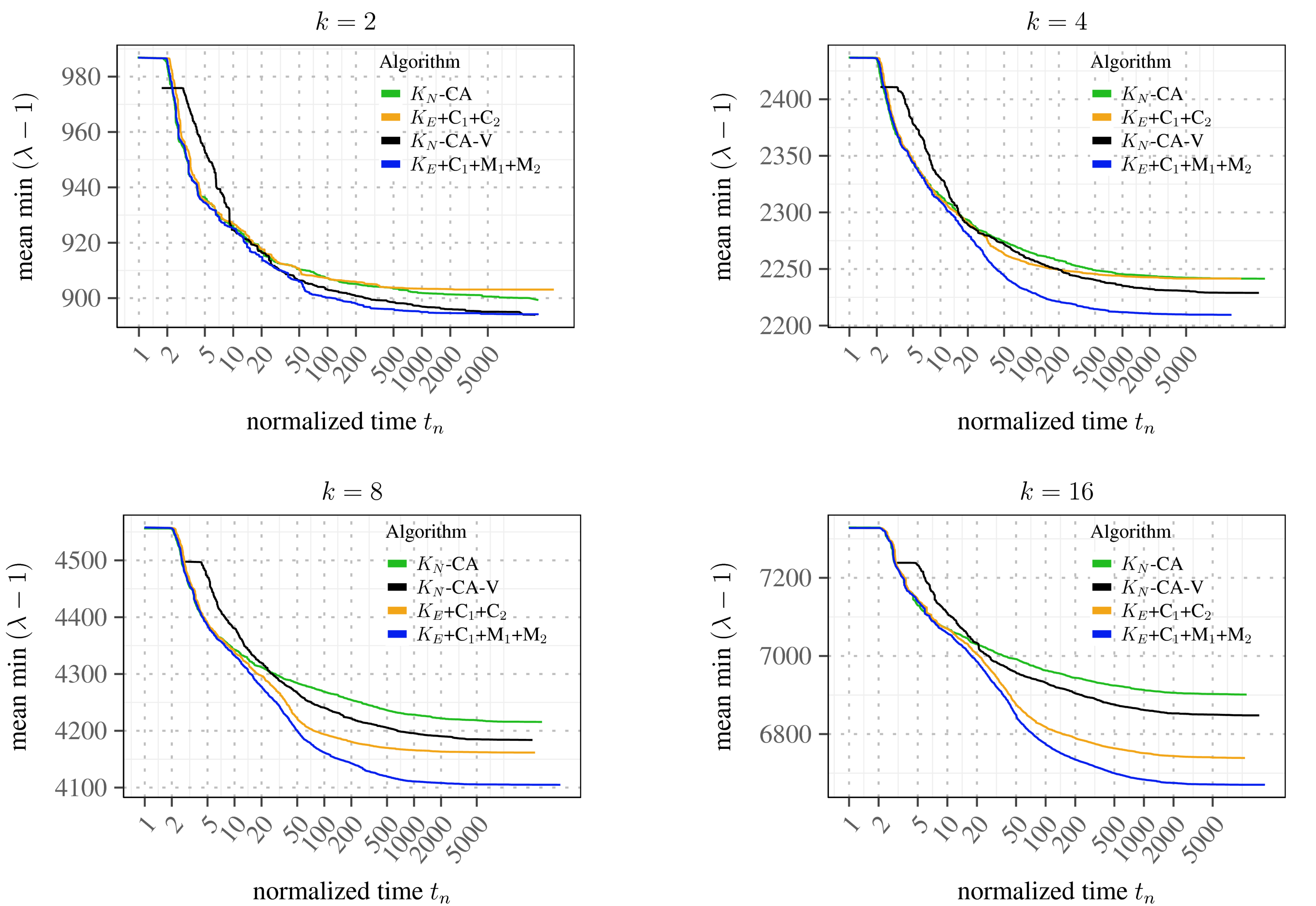
- The run time is normalized $t_n = \frac{time}{t_1}$; $t_1$ := duration of first iteration.
  - Allows comparing differently sized hypergraphs
  - Algorithmic components can be analyzed on run time

# Results



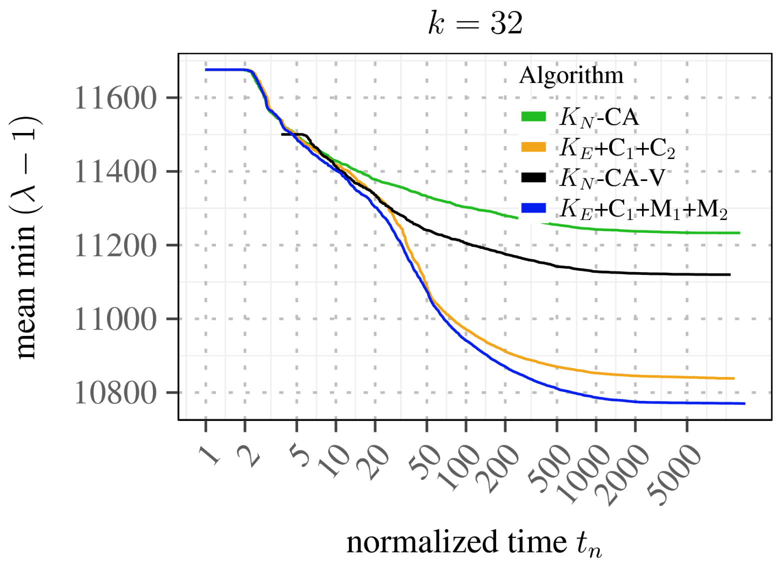
mean min $(\lambda - 1)$
6300
6200
6100
6000
5900
5800
1
2
5
10
20
50
100
200
500
1000
2000
5000
normalized time $t_n$
Algorithm
$K_E$+$C_1$
$K_E$+$C_1$+$C_2$
$K_E$ + $C_1$ +$M_1$ +$M_2$
$K_N$-CA
$K_N$-CA-V

# Results

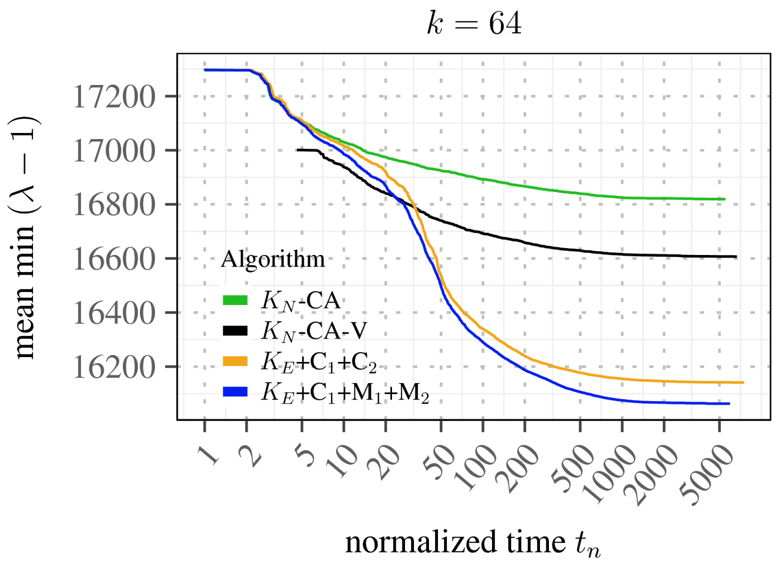


$k = 2$
Algorithm
$K_N$-CA
$K_E$+C$_1$+C$_2$
$K_N$-CA-V
$K_E$+C$_1$+M$_1$+M$_2$
mean min $(\lambda - 1)$
980
960
940
920
900
1 2 5 10 20 50 100 200 500 1000 2000 5000
normalized time $t_n$
$k = 4$
Algorithm
$K_N$-CA
$K_E$+C$_1$+C$_2$
$K_N$-CA-V
$K_E$+C$_1$+M$_1$+M$_2$
mean min $(\lambda - 1)$
2400
2350
2300
2250
2200
1 2 5 10 20 50 100 200 500 1000 2000 5000
normalized time $t_n$
$k = 8$
Algorithm
$K_N$-CA
$K_N$-CA-V
$K_E$+C$_1$+C$_2$
$K_E$+C$_1$+M$_1$+M$_2$
mean min $(\lambda - 1)$
4500
4400
4300
4200
4100
1 2 5 10 20 50 100 200 500 1000 2000 5000
normalized time $t_n$
$k = 16$
Algorithm
$K_N$-CA
$K_N$-CA-V
$K_E$+C$_1$+C$_2$
$K_E$+C$_1$+M$_1$+M$_2$
mean min $(\lambda - 1)$
7200
7000
6800
1 2 5 10 20 50 100 200 500 1000 2000 5000
normalized time $t_n$

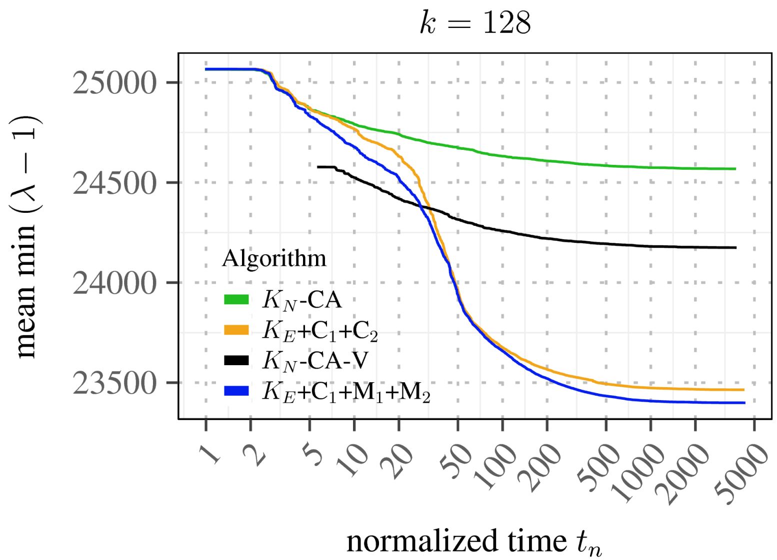
# Results



$k = 32$
Algorithm
$K_N$-CA
$K_E$+C$_1$+C$_2$
$K_N$-CA-V
$K_E$+C$_1$+M$_1$+M$_2$
mean min $(\lambda - 1)$
11600
11400
11200
11000
10800
1
2
5
10
20
50
100
200
500
1000
2000
5000
normalized time $t_n$

$k = 64$
Algorithm
$K_N$-CA
$K_N$-CA-V
$K_E$+C$_1$+C$_2$
$K_E$+C$_1$+M$_1$+M$_2$
mean min $(\lambda - 1)$
17200
17000
16800
16600
16400
16200
1
2
5
10
20
50
100
200
500
1000
2000
5000
normalized time $t_n$

$k = 128$
Algorithm
$K_N$-CA
$K_E$+C$_1$+C$_2$
$K_N$-CA-V
$K_E$+C$_1$+M$_1$+M$_2$
mean min $(\lambda - 1)$
25000
24500
24000
23500
1
2
5
10
20
50
100
200
500
1000
2000
5000
normalized time $t_n$

# Table of Improvements

Karlsruhe Institute of Technology

| | $K_E$ + $C_1$ + $C_2$ | | $K_E$ + $C_1$ + $M_1$ + $M_2$ | |
|---|---|---|---|---|
| $k$ | $K_N$-CA-V | $K_N$-CA | $K_N$-CA-V | $K_N$-CA |
| all $k$ | 1.7% | 2.7% | 2.2% | 3.2% |
| 2 | -0.2% | 0.4% | 0.2% | 0.8% |
| 4 | -0.2% | 0.3% | 0.9% | 1.3% |
| 8 | 0.7% | 1.6% | 1.9% | 2.7% |
| 16 | 1.9% | 2.8% | 2.6% | 3.5% |
| 32 | 2.9% | 3.9% | 3.2% | 4.2% |
| 64 | 3.2% | 4.7% | 3.4% | 4.8% |
| 128 | 3.3% | 5.0% | 3.3% | 5.0% |

# Conclusion

## Conclusion

- $(\lambda - 1)$ improvement of up to 5%
- High integration of evolutionary aspects in the multilevel approach

## Future Work

- Added parallelization for faster partitioning
- Different approach for generating the initial population
- Time cost analysis for evolutionary operators