Bachelor thesis

Title of Thesis

Name of author

Date: 9. Oktober 2017

Supervisors: Prof. Dr. Peter Sanders
Dipl. Inform. Zweiter Betreuer

Institute of Theoretical Informatics, Algorithmics
Department of Informatics
Karlsruhe Institute of Technology

Abstract

In this thesis we augment the existing Hypergraph Partitioner KaHyPar with an evolutionary framework with the goal to improve the solution quality.

Acknowledgments

I'd like to thank Timo for the supply of Club-Mate

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Ort, den Datum

Contents

1 Introduction

1.1 Motivation

Hypergraph Partitioning is a highly complex field of study. The Motivation behind this work is to add an evolutionary framework to the existing Hypergraph partitioner KaHyPar in order to improve the cuts of the Hypergraph.

1.2 Contribution

1.3 Structure of Thesis

2 Fundamentals

2.1 General Definitions

A Hypergraph $H = (V, E, c, w)$ is defined as a set of verticies V a set of hyperedges E where each edge may contain an arbitrarily large subset of V. $c : V \to \mathbb{R}_{\geq 0}$ is a function applying a weight to each Vertex and $w : E \to \mathbb{R}_{\geq 0}$ applying a weight to each hyperedge. Two verticies u, v are adjacent if $\exists e \in E | u, v \in e$ and a vertex u is incident to a hyperedge e if $u \in e$. The size $|e|$ of an Hyperedge e is the number of verticies contained in e. A k-way partition of a Hypergraph H is a partition of V into k disjoint blocks $V_1, ..V_k$. $part : V \to [0, k-1]$ is a function referencing the corresponding block of a k-way partition to a vertex u. A k-way partition is balanced when the weight of each block $V_i | 1 \leq i \leq k, \sum_{v_i \in V_i} c(v_i) \leq (1+\epsilon)\lceil \frac{\sum_{v \in V} c(v)}{k} \rceil$ for a balance constraint ϵ. A valid solution is a balanced k-way partition. An invalid solution is a partition where either the balance criterion is not met, or H has been partitioned for a different value for k. A Hyperedge e is a cut edge $cut(e)$ if $\exists u, v \in e | part(u) \neq part(v)$. The connectivity of a Hyperedge e is $\lambda(e) = \sum_{i=0}^{k-1} \delta(e, i) | \delta(e, i) = \begin{cases} 1 & \exists v \in e \ part(v) = i \\ 0 & \text{else} \end{cases}$

The set cut edges in H is defined as $cut(E) := \{e \in E | cut(e)\}$. The multiset connectivity edges in H is defined as $conn(E) := \{a(e) \in E | cut(e)\} | a(e) := \lambda(e)$ The cut metric $cut(H) := \sum_{e \in E} \begin{cases} \text{w(e)} & \text{e is cut edge} \\ 0 & \text{else} \end{cases}$ and gives the value of cuts. The connectivity metric $(\lambda - 1)(H) := \begin{cases} \lambda(e) * w(e) & \text{e is cut edge} \\ 0 & \text{else} \end{cases}$ Both metrics can be used to measure the quality of a solution. Throughout this thesis the solution quality is referenced. The metrics are interchangeable in this regard. An Individual I is a valid solution for the k-way partition problem of H. An individual eligible for further operations is considered *alive*. The difference of two individuals I_1, I_2 is $diff(I_1, I_2) := cut(I_1) \ominus cut(I_2)$ The connectivity difference of two individuals I_1, I_2 is $strongdiff(I_1, I_2) := conn(I_1) \ominus conn(I_2)$ A population P is a collection of Individuals.

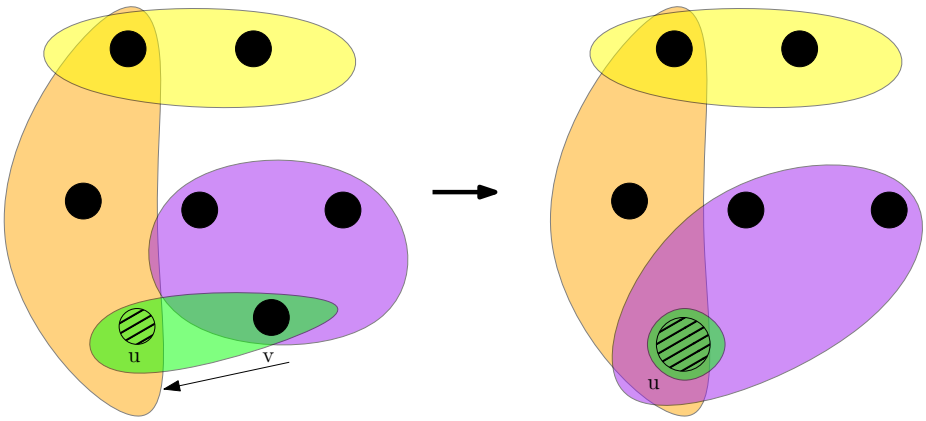


Figure 2.1: An example of a contraction. Note that the Hyperedges of the reduced node v are rearranged to contain u.

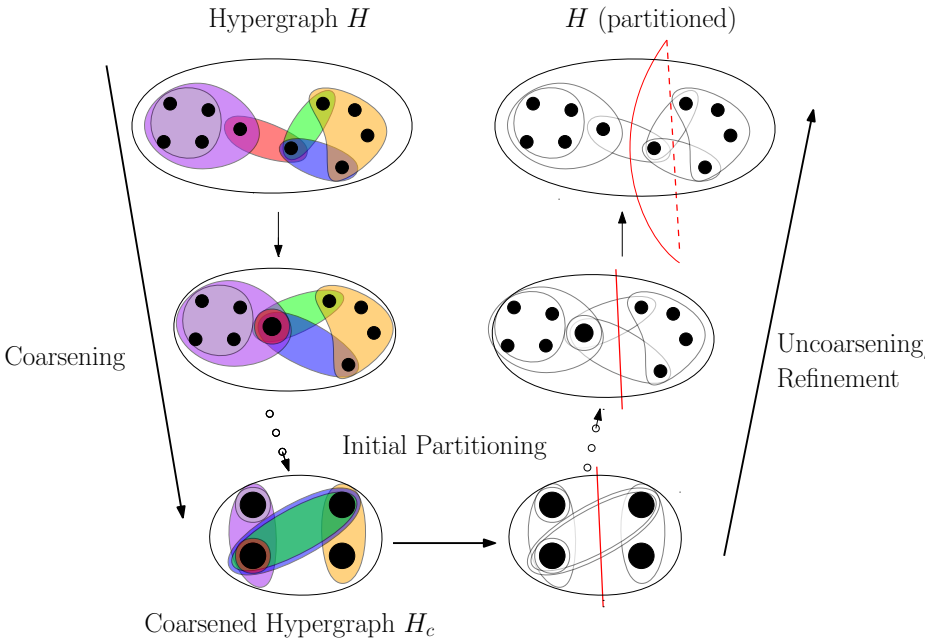


Figure 3.1: An example of an iteration. Coarsening, Initial Partitioning and Refinement

3 Related Work

The Hypergraph Partitioner KaHyPar uses a multilevel approach for partition. The original Hypergraph H is coarsened by repeadeatly contracting nodes u, v until either no more contractions are possible due to the size of the contracted nodes or that the minimum amount of nodes required to be in the coarsened Hypergraph H_c has been reached. During each step of the coarsening only one pair of nodes u, v is contracted. Nodes for contraction are chosen to contract multiple high weight edges to emphasize that u, v should most likely share the same partition. On H_c partitioning algorithm is chosen to generate an initial partitioning. Contracted nodes acquire the partition of their corresponding representing node. Afterwards to contraction operations will be reverted and during each step of the uncoarsening phase local search algorithms are used to improve the current connectivity of H. The local search is based on the principle of Fiduccia-Mattheysis algorithm where operations of decreasing quality are considered, but only executed if the sum of operations results in a net gain.

The memetic Graph Partitioner KaHiP uses evolutionary actions to increase solution quality and is the main inspiration for the operations implemented in KaHyParE.

4 KaHyParE

4.1 Overview

The objective of the algorithm is to optimize solution quality of a k-way partition of H within bounded time. During this time the algorithm will operate on already existing solutions in order to generate improved solutions. Most operators introduced in this thesis are using the multilevel approach of KaHyPar as black box, with some alterations towards the general procedure.

4.2 Population

The algorithm will produce multiple individuals, which are inserted and removed from the population. At any given time only a finite amount of individuals are *alive* which is the maximum population size. Furhter individuals have to compete for a place in the population. The population size is an important parameter, as a small value limits the solution scope and a high value limits convergence. We use KaHyPar to fill the initial population. in order to select a proper population size for the runtime, we dynamically allocate 15% of the runtime to create and fill the initial population.

4.3 Diversity

By measuring the difference between two individuals we can gain some knowledge over their internal structure and similarities. using the edges rather than the verticies for difference ensures that only elements relevant for the solution quality are considered for difference and perturbations of the partition blocks are not influencing the difference. As such we can consider the diversity of the population as the difference of the current individuals. Therefore a low diversity is a population where the individuals are in close proximity considering the solution space or convergent. A high diversity means that the individuals are spread throughout a larger portion of the solution space.

4.4 Selection Strategies

For an evolutionary algorithm we attempt to generate new, improved solutions by using existing solutions. A logical conclusion is that good individuals probably will generate solutions close to the original individual and as such good. We select our individuals using tournament selection. By first selecting two random individuals and using the one with the better solution quality we can extract one individual while ensuring that better solutions are more likely to be used by the algorithm. For operators requiring two seperate Individuals we can simply redo this step. In the unlikely case that the two selected individuals are the same we instead use the worse individual from the second tournament selection.

4.5 Combine operations

4.5.1 Basic Combine

The basic combine uses two parent partitions P_1 P_2 in order to create a child individual C. This is achieved by only allowing contractions of nodes u, v when these nodes are not placed in different partitions in either parent. Afterwards we do not perform an initial partitioning, instead we consider the coarsened Hypergraph and see which of the parents gives the better objective on said graph. Since the contraction condition is rather strong and we do not need to do an initial partitioning, we can remove the coarsening limit s used by KaHyPar to allow for more contractions. The uncoarsening and application of local search algorithms which guarantee to atleast maintain solution quality in combination with using the better partition of the two parents ensures that the child solution is at least as good as the best parent solution. It is imporant to note that the combine operation is more powerful the more diverse the parent individuals are, since similar parents are essentially just a vcycle. This will be important in the replacement strategy which will insert the child into the population and is the reason why the selection strategy is avoiding using the same individual.

4.5.2 Cross Combine

4.5.3 Edge Frequency Multicombine

We also introduce a multi-combine operator, capable of combining multiple individuals $I_1..I_n, n \leq |P|$ into a new child individual. By analyzing whether an edge e is a cut edge in $I_1..I_n$ we can calculate the edge frequency of e by $f(e) := \sum_{i=1}^{n} cut(e, I_i)$. We use the best $n = \sqrt{|P|}$ individuals from P for determining edge frequency. As such, edges with a high frequency are more likely to be cut edges in equally good solutions. Therefore we penalize contractions of nodes incident to a high frequency edge

during the multilevel partition approach by using this formula $\frac{1}{(w(u)w(v))^{1.2}} * e^{-\gamma * f(e)}$ to disincentivize early contractions of nodes in edges of high probability. Since these contractions are most likely not improving the quality and will make other, perhaps more favorable contractions invalid. Unlike the other combine operators edge frequency does not set any of the input partitions I_i as current partition of H during the iteration. The only step that differentiates this operator from a regular iteration using KaHyPar is that the rating function during the coarsening is adapted by the penality formula above. It is necessary that an initial partitioning is generated for the coarsened Hypergraph.

4.6 Mutation operations

Mutations are performed on randomly chosen individuals instead of tournament selection.

4.6.1 VCycle

A Vcycle is an iteration in the regular KaHyPar procedure with the difference that H is already partitioned. During the coarsening nodes u, v may only be contracted if $part(u) = part(v)$. Since the Graph is already partitioned there is no need for initial partitioning. The main benefit comes from the refinement during the uncoarsening. This operation takes one individual I and the result is a similar individual I_c. Due to the fact that neither refinement nor coarsening worsen the quality of the solution the quality of I_c will also not be worse than I. This is a weak mutation and the difference of I and I_c are small. This operation will also cause convergence, as multiple applications of a vcycle will eventually no longer be able to generate improvement and are unable to escape the local optima due to the fact that worse solutions can not be generated.

4.6.2 VCycle + New Initial Partitioning

Similar to a vcycle we can coarsen H with a set partition, but if the partition is dropped after the coarsening, and a new initial partitioning is performed we have an operation generating a more pertubed solution, while applying the information of the original partition during the coarsening. This operation also takes one individual I and produces an offspring I_c. Since the partition is dropped the algorithm used to generate a new partition might produce a worse solution as before. Therefore this operator can create worse solutions. This operation explores more solution space than regular vcycles.

4.6.3 Stable Nets

Opposed to edge frequeny where edges of high probability of being in the cut are penalized because these edges are most likely being in the cut regardless, stable net removal attempts to broaden the solution scope by forcefully removing these edges out of the cut. Again the $\sqrt{|P|}$ best individuals are analyzed, regarding edges most frequent in these solutions. These edges are then attempted to be forced into the block with the smallest amount of nodes in order to keep the balance criterion met. Forcefully moved nodes may not be reassigned through another stable net. These solutions have most likely siginficantly worse quality. Due to the nature of our selection strategy these solutions are very unlikely to be used in any combine operator. In order to keep these solutions competitive we also perform a vcycle after removing the stable nets.

4.7 Replacement Strategies

Regardless of the operator, the generated individual I_c has to be inserted into the population in order to be used in upcoming iterations. The naive approach is to remove the worst element from the population and insert I_c in its place, with the intention to ensure a vast majority of the best generated solutions. The consequence is that the population is rapidly converging towards a local optima and only covering a small amount of the solution space. Another approach is to replace the element used in the operator. This strategy was originally used for mutations, as the theory behind evolutionary algorithms is to pertubate an existing solution. Instead trying to merge two objectives for the replacement strategy, quality and diversity, there is a compromise strategy which replaces the most similar element to I_c that is also worse than I_c in terms of quality. The most natural approach towards this problem is to use the difference metric to the corresponding quality metric. As a result incompetitive solutions are removed entirely and the replaced element is in relatively close neighborhood towards the newly generated solution which ensures more granular steps through the solution space.

5 Experimental Evaluation

We evaluate our algorithm on two Hypergraph sets. One time for different $k = \{2, 4, 8, 16, 32, 64, 128\}$ and 174 Hypergraph Instances, repeating each run 5 times with different seeds and a algorithm runtime of 8 hours. This is referenced as benchmark subset. The other evaluation is a selection of 25 Hypergraphs using only one $k = 32$ and a runtime of 2 hours. This is referenced as tuning subset. As such our datasets contain multiple tuples of (H, k, s, t, λ) where H is the instance, s the seed, t the required time and λ the solution quality. An Instance I is the subset of (H, k, s, t, λ) where H and k are fixed. The main interest is the best solution over time compared to other partitioning algorithms. Unfortunately most graphs vary drastically in solution quality and time required to perform one iteration. We solve the time differences by not using the measured time, but rather the normalized time. In order to do so we choose one of the partitioning algorithms p as baseline and determine the average duration t_I of an iteration for each I. Afterwards for each I the normalized time t_n is calculated by $t_n = \frac{t}{t_I}$. We then generate a list for each instance containing (s, t_n, λ) sorted by t_n. Now we generate the averaged improvements for I. For each seed s a value a_s is reserved and the list is read in. When (s, t_n, λ) is better than the reserved value for s the average over all a_s is the new value appended to the averaged improvements with$((avg(a), t_n)$. Similarly we now reserve a value a_I for each Instance of averaged improvements and calculate a new list of general improvements by calculating the geometric mean over all a_I.

5.1 Implementation

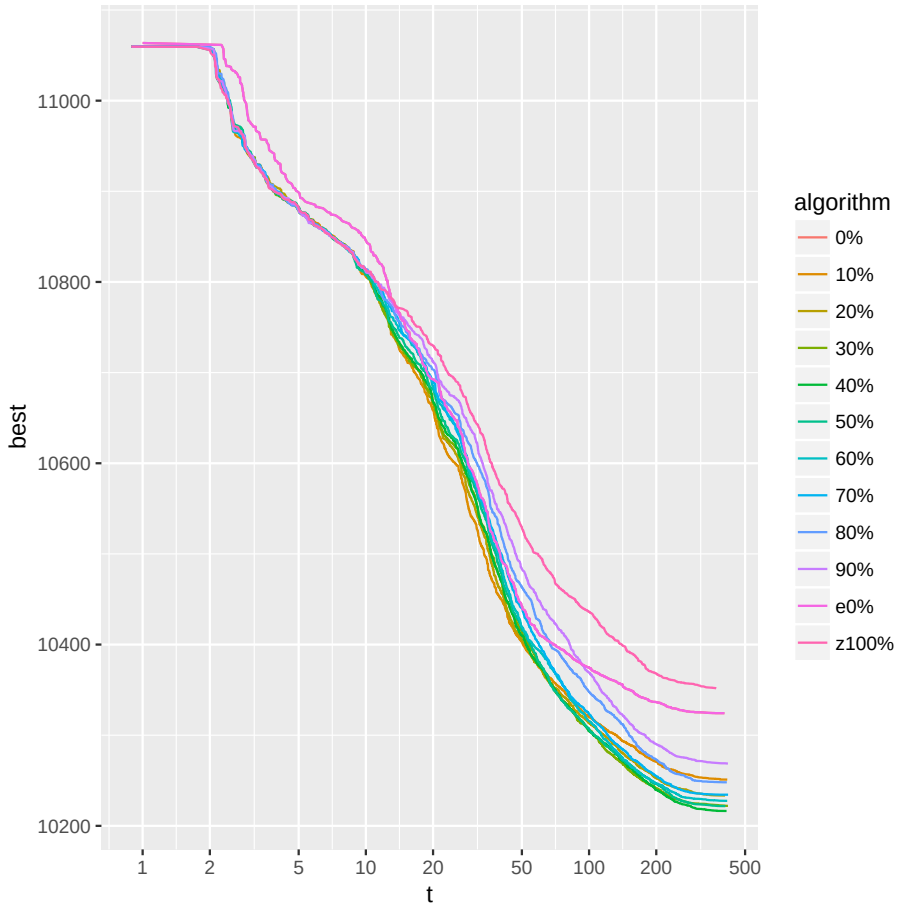
5.2 Experimental Setup

5.2.1 Environment

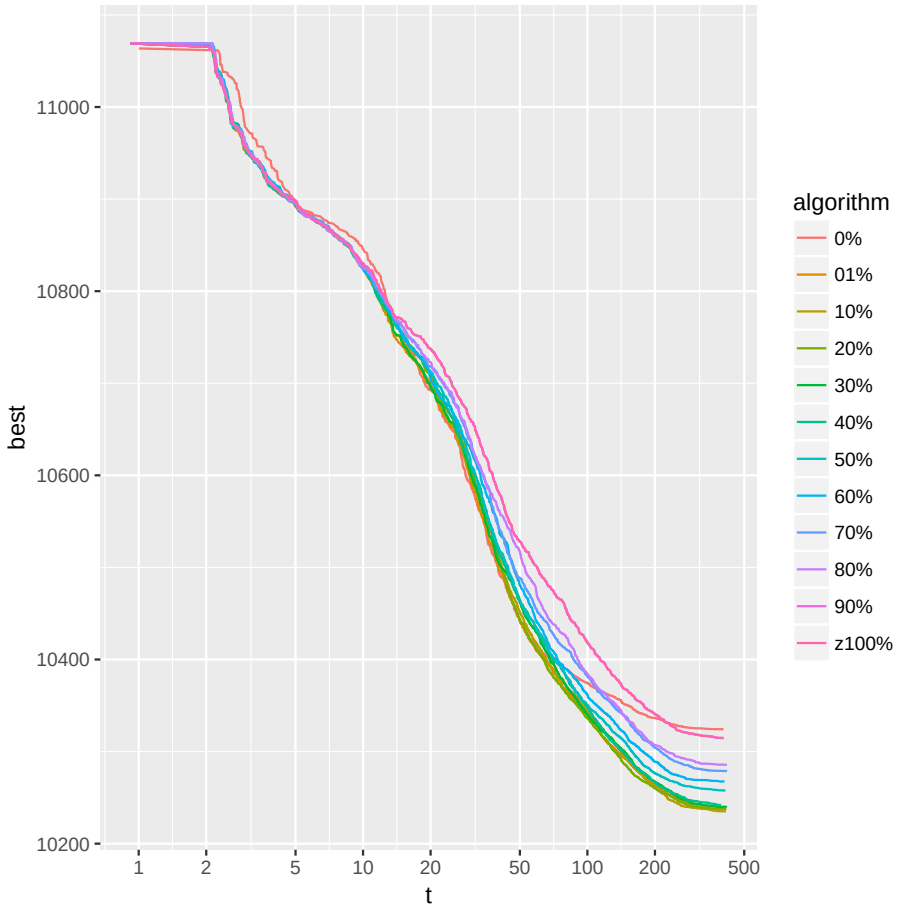
5.2.2 Tuning Parameters

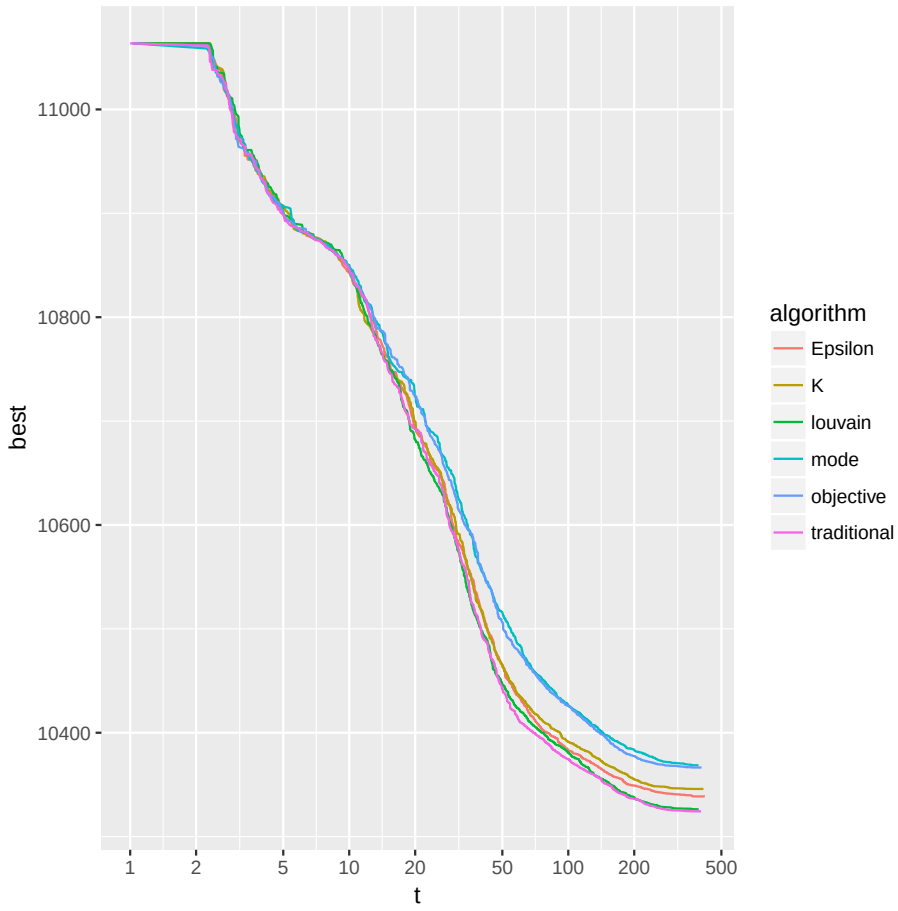
5.2.3 Instances

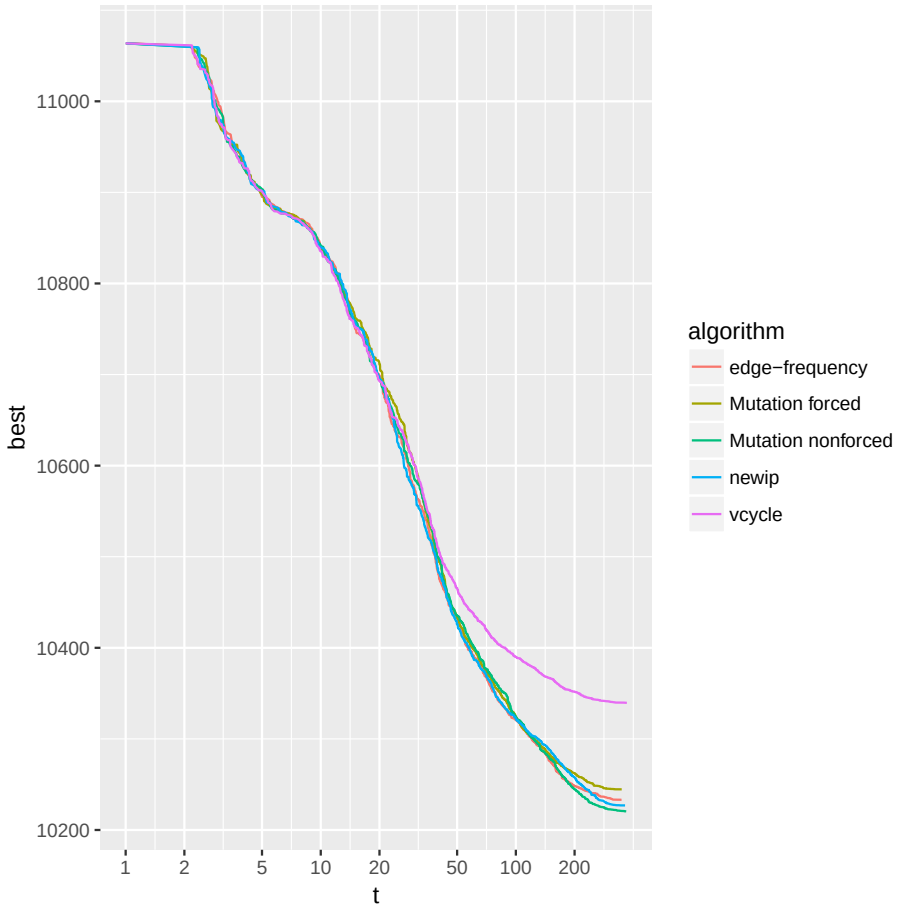
5.3 Your Experiment Headline

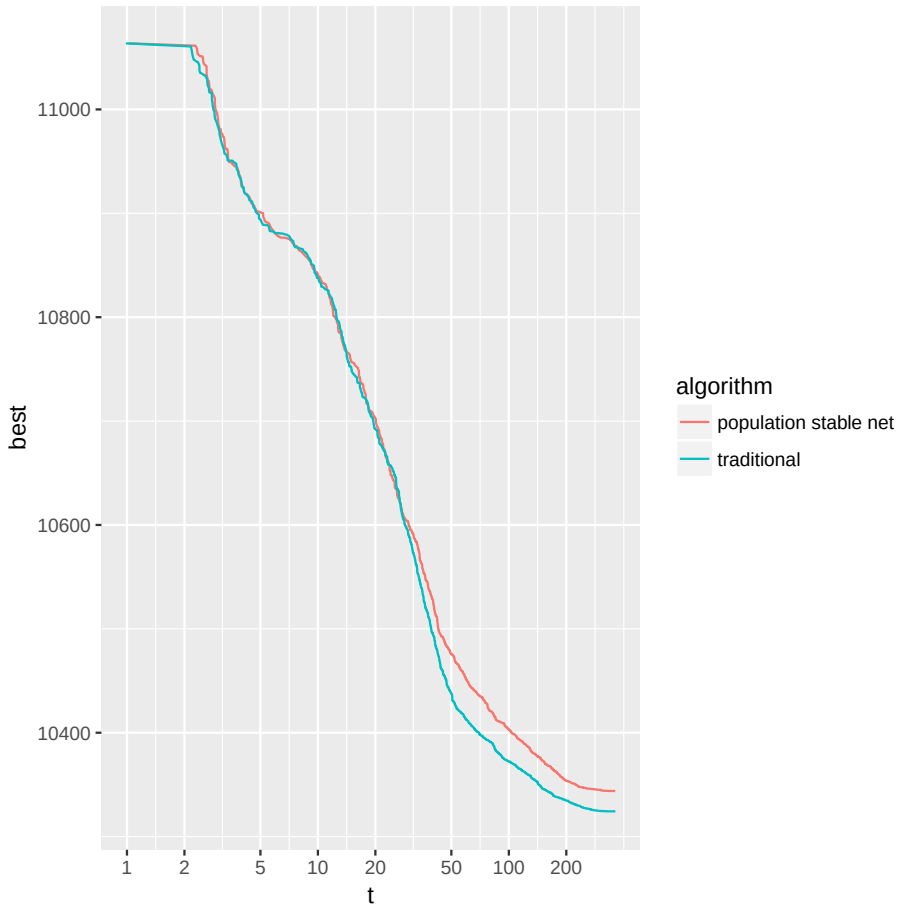


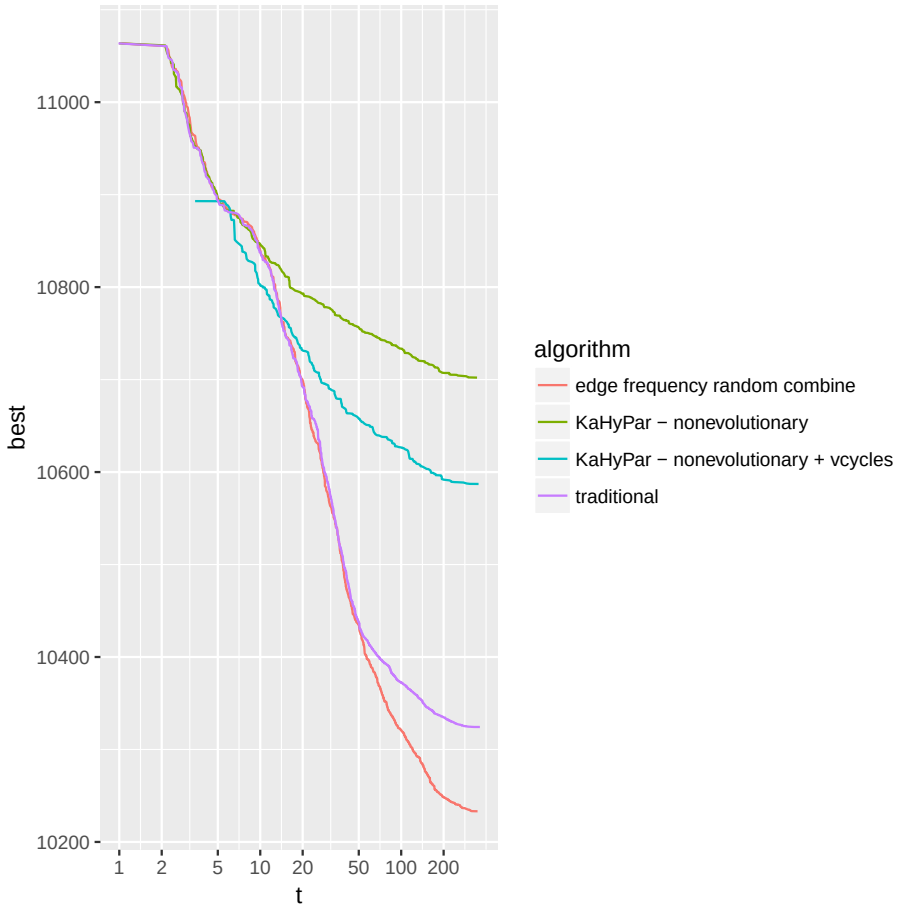
In this graph you can see clearly that optimizing stuff does things.


11000
10800
10600
10400
10200
best
1
2
5
10
20
50
100
200
500
t
algorithm
0%
01%
10%
20%
30%
40%
50%
60%
70%
80%
90%
z100%


11000
10800
10600
10400
best
1
2
5
10
20
50
100
200
500
t
algorithm
Epsilon
K
louvain
mode
objective
traditional


11000
10800
10600
10400
10200
best
1
2
5
10
20
50
100
200
t
algorithm
edge−frequency
Mutation forced
Mutation nonforced
newip
vcycle


11000
10800
10600
10400
best
1
2
5
10
20
50
100
200
t
algorithm
population stable net
traditional


11000
10800
10600
10400
10200
best
1
2
5
10
20
50
100
200
t
algorithm
edge frequency random combine
KaHyPar − nonevolutionary
KaHyPar − nonevolutionary + vcycles
traditional

6 Discussion

6.1 Conclusion

6.2 Future Work

Vcycles härter Kreativere auswahlstrategie

A Implementation Details

A.1 Software

A.2 Hardware