Northeast Parallel Architectures Center
at Syracuse University

A Collection of Graph Partitioning Algorithms:

Simulated Annealing, Simulated Tempering,
Kernighan Lin, Two Optimal,
Graph Reduction, Bisection

Gregor von Laszewski
gregor@nova.npac.syr.edu

Technical Report
unpublished

Science and Technology Center
111 College Place
Syracuse, NY 13244-4100
Tel.: (315) 443-1722, 1723; Fax: (315) 443-1973

A Collection of Graph Partitioning Algorithms:

Simulated Annealing, Simulated Tempering,
Kernighan Lin, Two Optimal,

Graph Reduction, Bisection

Gregor von Laszewski
gregor@nova.npac.syr.edu

May 17, 1993

Contents

A Collection of Graph Partitioning Algorithms:

Simulated Annealing, Simulated Tempering,

Kernighan Lin, Two Optimal,

Graph Reduction, Bisection

Gregor von Laszewski

gregor@nova.npac.syr.edu

Abstract

The NP complete problem of the graph bisection is a mayor problem occurring in the design of VLSI chips. A simulated annealing algorithm is used to obtain solutions to the graph partitioning problem.
As stated in may publications one of the major problems with the simulated annealing approach is the huge amount of time one has to spend in the annealing process. To increase the speed the structure of the graph is used before and while the annealing process is performed. This is done by reducing the graph and applying the annealing process after the reduction step. Nodes which have neighbors to the other partition are preferred for a possible interchange.
The project has the following purpose:

- Investigation of simulated annealing for the uniform graph partitioning problem. Different annealing schedules are compared.
- Investigation of the influence of the reduction algorithm on the speed and the quality of the solutions obtained.
- Investigation of the use of the Cauchy rule instead of the Boltzmann rule which is proposed in the VSFR algorithm.

Keywords: Simulated annealing, VLSI placement, graph partitioning.

1 Graph Partitioning

The uniform graph partitioning problem (GPP) is a fundamental combinatorial optimization problem which has applications in many areas of computer science (e.g., design of electrical circuits, mapping) [7, 12]. The term *graph partitioning problem* is used in literature for different problems. Following the paper [6] and the notation in [12] the graph partitioning problem can be formulated mathematically as follows:

Let $G = (V, E)$ be an undirected graph, where $V = \{v_1, v_2, ..., v_n\}$ is the set of n nodes, $E \subseteq V \times V$ is the set of edges between the nodes. The graph partitioning problem is to divide the graph into two disjoint subsets of nodes V_1 and V_2, such that the number of edges between the nodes in the different subsets is minimal, and the sizes of the subsets are nearly equal. The subsets are called *partitions*, and the set of edges between the partitions is called a *cut*.

Figure 1 shows a simple graph and a possible partition of this graph. The cost of this solution is 3 assuming that each edge has the weight 1.

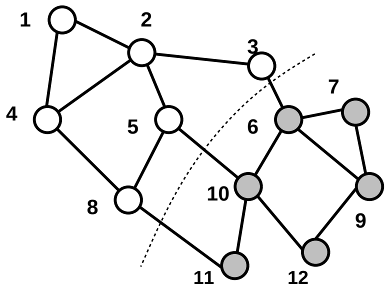


Figure 1: Example of a partition

For some algorithms it is from advantage not to maintain the strict constrained of the equal partition size. This can be done by extending the cost function with an imbalance term. Than the *cost* of a partition is defined to be

$$c(V_1, V_2) = |\{\{u, v\} \in E : u \in V_1 \text{ and } v \in V_2\}| + \alpha(|V_1| - |V_2|)^2$$

where $|A|$ is the number of elements in the set A and α controls the importance of the imbalance of the solution. The higher α the more important is the equal balance of the partitions in the cost function.

2 The Testbed and Classification

2.1 Classification

This study will present many algorithms for finding solutions to instances of the graph partitioning problem. One criterion for a classification of these algorithms is to determine how a solution is created.
Having this in mind *improvement* and *construction* algorithms are distinguished. A construction algorithm takes generally a node and decides to which partition it belongs while an improvement algorithm tries to improve a given solution with the help of a heuristic.
The algorithms used in this paper can be grouped in the following way:

1. Construction Algorithms
   (a) Random Solution
   generates a random solution
   (b) Simple Graph Reduction
   reduces the graph
   (c) Line Bisection
   divides the nodes of the graph with the help of a line
2. Improvement Algorithms
   (a) Two Optimal Heuristic
   is described in detail in [7] and [12]
   (b) Kernighan Lin Heuristic
   is described in detail in [7] and [12]
   (c) Simulated Annealing
   models a cooling process
   (d) Simulated Tempering and Adaptive Simulated Tempering
   models a tempering process

The algorithms above will be introduced in the next sections. For the Kernighan Lin and Two Optimal heuristic we refer the reader to the references specified above. At the end of the paper it will be clear that the combination of construction and improvement algorithms leads to very good solution strategies.
Nevertheless, it is obvious that for a given graph the strategy used is dependent on the problem instance. For example, it is useful to include geometrical information in the graph partitioning algorithm if such information is available. It does not make sense to waste time in lengthly runs while neglecting major information about the graph.

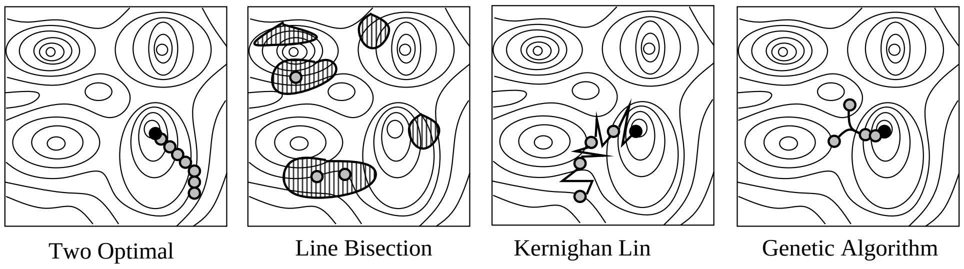


Figure 2: Landscape

For the improvement algorithms introduced here we try to classify them as shown in Figure 2. While the algorithm *Two Optimal* just looks in its immediate neighborhood for an improvement, *Simulated Annealing* and the *Kernighan Lin* algorithm are able to explore the solution space further, indicated with long jumps. Genetic Algorithms [14, 13] have the property of combining two solutions in order to construct a new solution in the hope to combine the good parts of both parent solutions. The *Line Bisection* algorithm restricts the solution space to small areas dependent on a parameter α. From this regions the best ones are taken.

2.2 Problem Instances

The graphs used in this study are shown in the Figures 3 to 6. Some properties of the graphs are displayed in the Table 1, where n specifies the number of nodes and $\overline{\Gamma}$ is the average degree of a node in the graph. The density of a graph is specified by the number of edges in the graph divided by n^2. Each edge in a graph has the weight 1. The properties of the geometric graph are described in detail in [6]. For most of the study we will concentrate on the geometric graph with 500 nodes.

	Graph	n	Number of Edges	$\overline{\Gamma}$	density in %	Source
1	Graph 78	78	270	3.5	4.44	[5]
2	Grid 100	100	360	3.6	3.60	
3	Grid 400	400	1520	3.8	0.95	
4	Geometric 500	500	4710	9.42	1.88	[6]

Table 1: Problem Instances

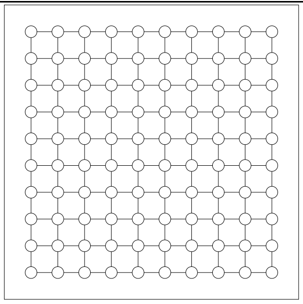


Figure 3: Grid graph with 100 nodes

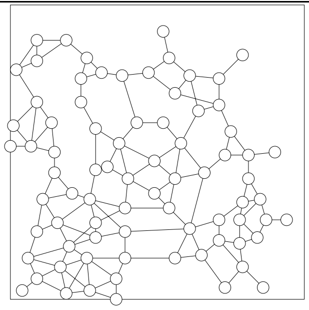


Figure 4: The graph with 78 nodes

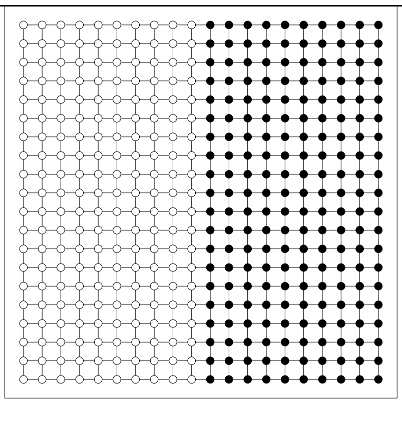


Figure 5: Grid with 400 Nodes. An optimal solution with cost of 20 is shown

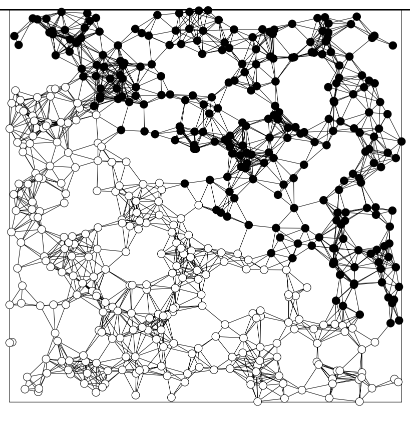


Figure 6: Johnsons geometric graph with average degree 5. The solution with cost 50 is obtained with the help of the line bisection algorithm

3 The Simulated Annealing Approach

Simulated Annealing (SA) is an optimization technique simulating a stochastically controlled cooling process[6, 2, 8, 9]. The principles of simulated annealing are based on statistical mechanics where interactions between a large number of elements are studied. One fundamental question in statistical mechanics is to study the behavior of such systems at low temperatures. To generate such states the system is cooled down slowly. As the name indicates *Simulated Annealing* simulates such an annealing process. The process is started at a random state s. Now the aim is to find a configuration t with lower energy obtained by a random disturbance of the current state s. The probability to get from one system configuration into the next can be described by a probability matrix (a_{st}) where a_{st} specifies the probability to get from configuration s to a *neighbor* configuration t. This probability matrix is chosen symmetrically. Is

$$E(t) - E(s) < 0,$$

the new configuration is accepted, since it has a lower energy. Is the energy of the new configuration t higher, than it is accepted with the probability proportional to

$$e^{\frac{E(t)-E(s)}{T}}.$$

The acceptance of the configuration with the higher energy is necessary to allow jumps out of local minimal configurations. This process is repeated as long as an improvement is likely. The choice of the temperature parameter is given by a cooling plan such that

$$T_1 \geq T_2 \geq ..., \text{ such that } \lim_{k\to\infty} T_k = 0$$

This leads to the generic simulated annealing algorithm shown in Figure 7.

```
PROC Generic Simulated Annealing
  BEGIN SEQUENTIAL
    chose the cooling strategy $T_1$, $T_2$, ...
    generate a starting solution $s$
    $k \leftarrow 1$
    REPEAT
      generate a new solution $t$ in the
        neighborhood of $s$
      $\Delta E \leftarrow E(t) - E(s)$
      IF $\Delta E < 0$ THEN
        $s \leftarrow t$
      ELSEIF $e^{-\frac{\Delta E}{T_k}}$ < random $[0,1]$ THEN
        $s \leftarrow t$
      END IF
      $k \leftarrow k+1$
    UNTIL $T_k \leq T_{min}$ or time limit reached
  END SEQUENTIAL
END PROC
```

Figure 7: The generic simulated annealing algorithm

3.1 SA and GPP

For the graph partitioning problem one can find the following analogies between the physical system:

physical system	SA-GPP
state	feasible partition
energy	cost of the solution
ground state	optimal solution

Using the cost function as given in equation (1) one can obtain solutions to problem instances by an annealing process. The definition of the probability matrix (a_{st}) can be chosen as follows:

- A partition t is neighbor to a partition s iff s can be obtained by moving one node of a subset to the other.

- The probability of the neighbor states to a partition s are the same.

3.2 Parameter scheduling for the Simulated annealing algorithm

Instead of using the generic simulated annealing algorithm we use the algorithm displayed in Figure ??. This is motivated by the following reasons:

1. It is easier to define a cooling scheme.
2. Many studies published use this scheme which makes comparison more easy.

```
PROC Simulated Annealing
   BEGIN SEQUENTIAL
     $T \leftarrow T_0$
     generate a starting solution $s$
     WHILE (NOT frozen) DO
        DO $i \leftarrow 1, L*$ Number of Nodes
            generate a new solution $t$ in the
               neighborhood of $s$
            (swap one node to the other part)
            $\Delta E \leftarrow E(t) - E(s)$
            IF $\Delta E < 0$ THEN
               $s \leftarrow t$
            ELSEIF $e^{-\frac{\Delta E}{T_k}} <$ random $[0,1]$ THEN
               $s \leftarrow t$
            END IF
            $time \leftarrow time + 1$
          END DO
          $T \leftarrow k * T$
      END WHILE
   END SEQUENTIAL
END PROC
```

Figure 8: Simulated Annealing Algorithm used for the Experiments

The simulated annealing algorithm 8 is controled by the parameters $L, k, alpha$. The variation of this parameter has a large influence on the solution quality as shown in the paper by Johnson. We were able to reproduce the results and show some of them in the figures 9 to 14. In these figures we show the temperature, the acceptance frequency, and the cost of some solutions generated during an Annealing process.

Acceptance Frequency The acceptance frequency specifies how frequent a change in the solution during a timeinterval of the annealing process occurred. The frequency is calculated over all trials in the inner loop of the interval length L.

This acceptance frequency is important since it helps to estimate when an improvement of the solution will be unlikely while continuing the annealing process. We decided to terminate the algorithm (the solution is frozen) when the acceptance probability drops under 0.00025 percent. With this value we achieved very good results while using an interval length L of size 16.

Imbalance Factor The imbalance factor has also an huge influence in the acceptance rate. Is the value to large the process stops to early in a bad solution is it to small we fall in a solution which is very imbalanced and often unwanted. In many studies the value of 0.05 is chosen and we could reproduce with this value very good results (Figure 11).

Temperature The initial temperature is very critical for an efficient simulated annealing run. While choosing the temperature to high no improvement of the solution occurs during the first annealing steps (Figure 14). Is the temperature chosen to low the cooling process terminates to quickly and does not spend time in intermediate solutions. Therefore, the solution space is not explored in depth. For our results we obtained very good results with a temperature of 0.5. The acceptance frequency at this temperature for the given problem instance is about 0.80. Johnson et. al. proposed to reduce the temperature such that the acceptance frequency drops to 0.4. In this case we reduced the running time about 1/3, while keeping the quality of the solutions found.

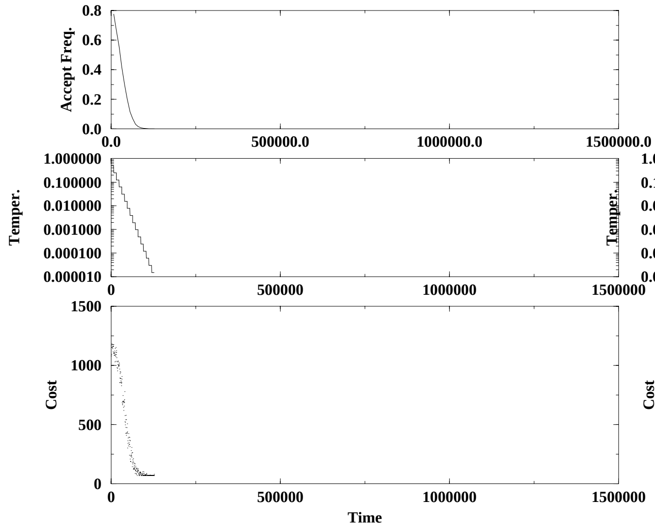


Figure 9: Using a to rapid cooling schedule (here $r = 0.5$) leads to fast convergence in solutions with high costs

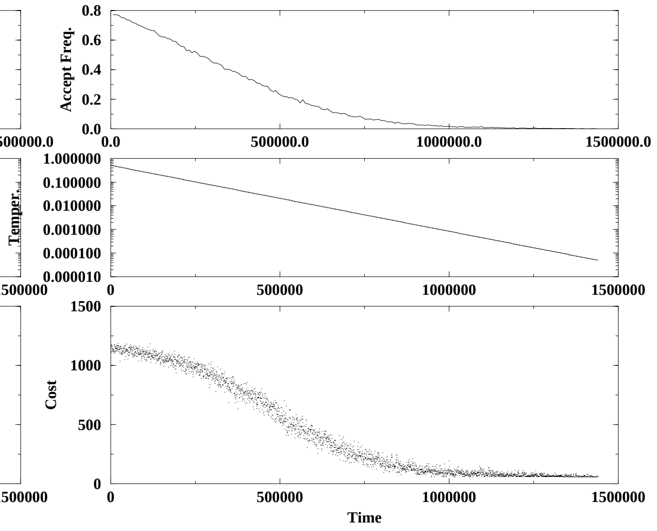


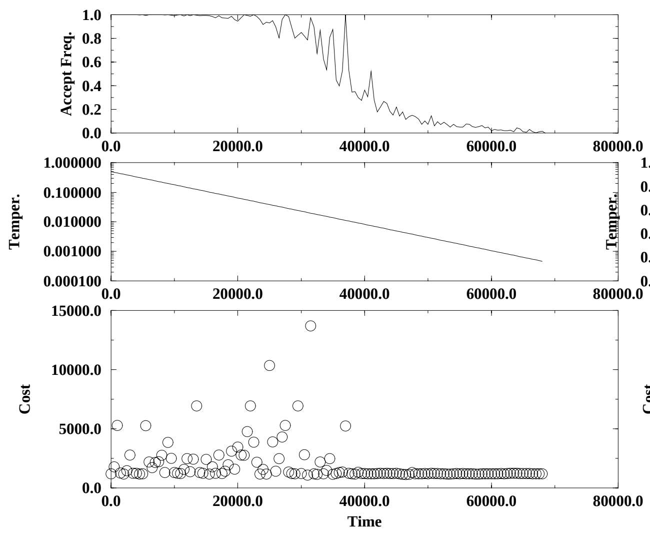
Figure 10: Using am initial temperature of 0.5 results in an acceptance rate of 0.8 at the beginning of the simulation and leads to good results

In case of high temperatures most of the transpositions are accepted. The smaller the temperature get the fewer transpositions are accepted. At temperature 0 only transpositions with positive ΔE are accepted. This effect is shown in Figure 13. A common way to specify the cooling strategy to chose a cooling rate such that

$$T_{k+1} = rT_k \ .$$

With an $r = 0.95$ we obtained very good results. Figure 9 shows what happens if the cooling scheme drops to quickly.

The following parameters leed to very good results: $T_0 = 0.06, \alpha = 0.05, k = 0.95, L = 16$. Overall the



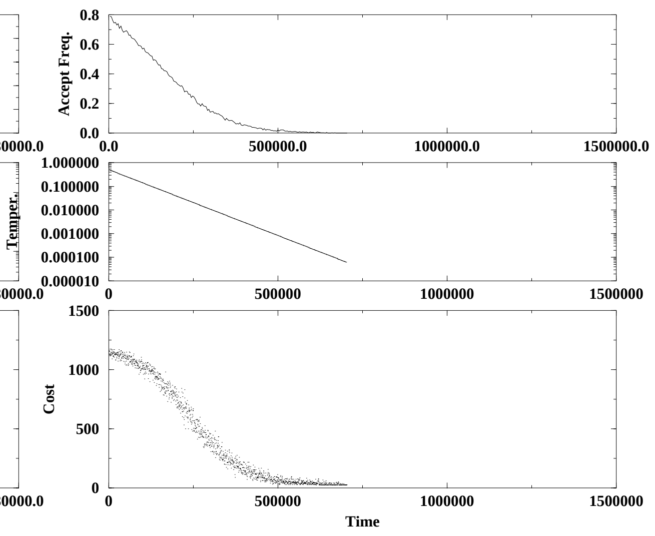


Figure 11: Using an high imbalance term (here 1) results in very quick termination and a solution with high cost

Figure 12: Reducing the parameter L increases the speed of the algorithm but with longer L better results are found

following results for the GPP and Simulated annealing are valid (see above and [6]).

- Long annealing must be used to get the best results.
- It is not necessary to spend a long time at high temperatures.
- A geometric cooling strategy is sufficient
- The variation of the solutions can be large even with long runs.
- The parameter setting is dependent on the problem instance.
- Small neighborhood sizes improve the running time

A comparison between the solutions found by the different algorithms is shown in section 6.2.

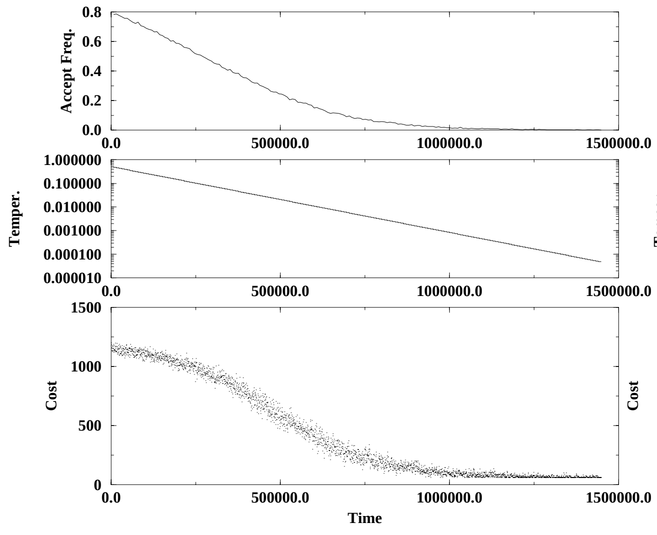


Figure 13: With the parameter $L = 16$ very good results have been found

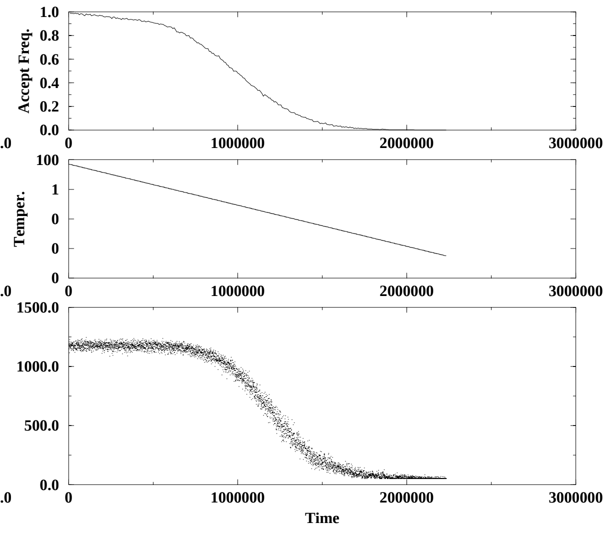


Figure 14: Choosing the Temperature to high (here 50) a lot of time is wasted at the beginning of the cooling strategy

4 Simulated Tempering

Simulated tempering is a global optimization method for simulating a system with a rough free energy landscape at finite non zero temperatures[10]. In contrast to the simulated annealing algorithm the state is kept at equilibrium. Instead of modifying the temperature with the function $T_{k+1} = rT_k$ the temperature is included in the annealing process. Its value is chosen stochastically from a set of predefined values. The method has been applied successfully to the Random Field Ising Model. We tried to apply the tempering algorithm to the Graph Partitioning Problem.

Inspite of many experiments we could not find a parameter setting which could come close to the results produced by the simulated annealing algorithm. The best value we could find with the tempering algorithm has the cost of 59. Unfortunately, this result could not be reproduced in a control experiment. For the most trials we obtained results in between the cost values of 80 and 120.

The Figures 15 to 19 show sample trials for different temperature schemes displayed in the Figure 20. The median of the temperature values is chosen as initial temperature. Having this in mind it is clear that the choice of a temperature scale from a high temperature to a low temperature does not lead to a good performance as shown in Figure 18. Based on the original implementation the frequency of the temperatures occurring during the experiment are Gaussian distributed around the median. Therefore, it is best to chose a symmetric function which has its median at the minimum value. Doing this we could gain a drastical improvement of the solution quality. Furthermore, we tried a linear distribution of the temperatures. This did not lead to good results, since too little time was spent in the regions around local minimal solutions. We could not introduce a direction in the search process.

Nevertheless, by choosing a logarithmic distribution also motivated by the experimental results found with the normal simulated annealing algorithm we could overcome this problem to a certain extend. It appears that the choice of the temperature values is even more difficult than with the simulated annealing algorithm. Our experiments show it is worth to consider for future experiments to adapt the temperature dynamically. This has two advantages

1. The choice of the temperature scheme is avoided.
2. The annealing process would be self adapting to the problem

The question arises why the simple simulated tempering algorithm is so difficult to adjust for the Graph Partitioning Problem but gives very good results for the Random Field Ising Model.
In the later problem there exists a phase transition and the temperature values in the region of interest are distinct from 0. For the problem instances used in this study we could not distinguish such phases. Which we will show in section 6.2. Another difference is that in order to find good solutions the temperature should be close to 0 in case of the graph partitioning problem.
The algorithmic search is much more smoothly. We conclude that the tempering algorithm in its original form is not suited for the problem instances considered here.

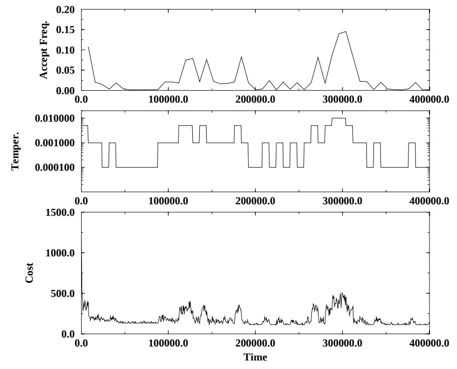


Figure 15: Temperatures : 0.01 0.001 0.005 0.0001

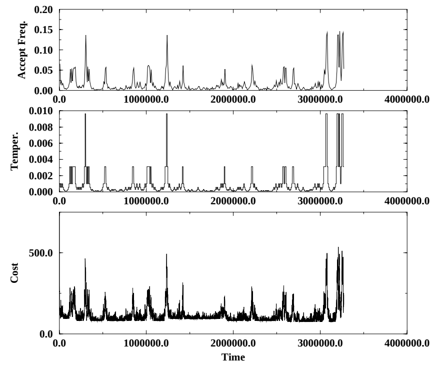


Figure 16: Experiment 4

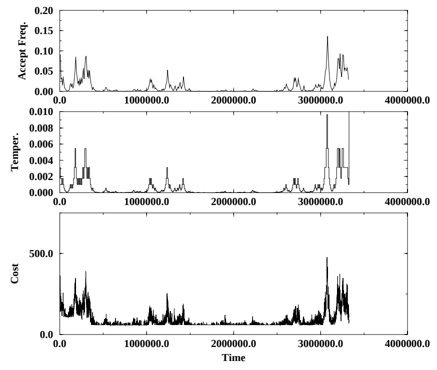


Figure 17: Experiment 2

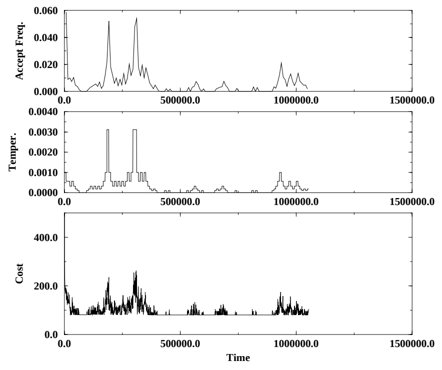


Figure 18: Experiment 3

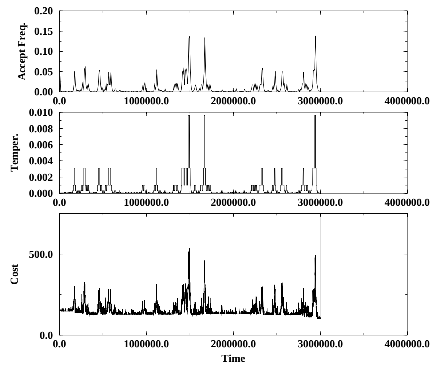


Figure 19: Experiment 6

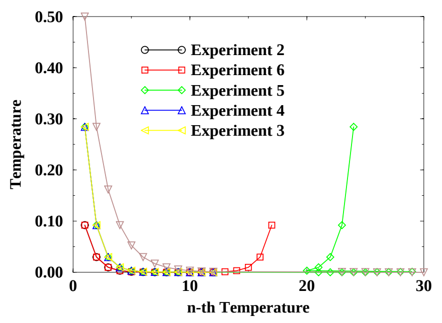


Figure 20: Temperatures

4.1 Adaptive Simulated Tempering

Our experiments show it is worth to consider for future experiments to adapt the temperature dynamically. First, we sampled a typical good annealing run and noted for each cost value at which temperature it occurred first and last. Figure 21 shows the result and Figure 22 shows the difference between the two Temperature extremes for each cost value.

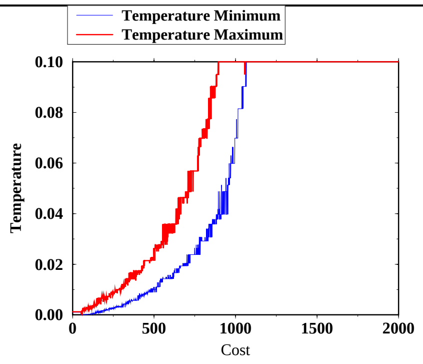


Figure 21: Minimal and maximal temperature value where a solution with given cost occurred

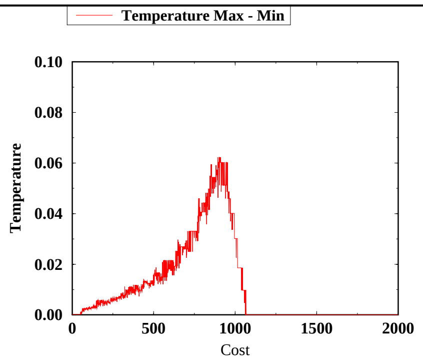


Figure 22: Difference between the minimal and maximal temperature value

Interesting is that the difference does not exceed the temperature value of 0.06. This can be explained by the fact that at temperatures greater than 0.06 only solutions occur which are more or less random and have high cost values.

Now it is clear that we the initial temperature of 0.1 is too high in our normal simulated annealing run. With the value of 0.06 we could obtain the same good results.
For the adaptive Tempering scheme we designed algorithm 23. The update of the temperature is done as described in [10]. The temperatures have been chosen

1. symmetrically with different schemes (logarithmic and linear) around in the interval $[s*T_{min}, s*T_{max}]$.

2. simply in the interval $[s*T_{min}, s*T_{max}]$.

As before we obtained better results for the symmetrical temperature distributions. The parameter s introduces the possibility to reduce the actual temperature interval to focus the search. The best result of cut 40 with imbalance 14 was obtained with the parameters $L = 16, s = 0.5, \alpha = 0.05$ and 10 temperature values for each tempering. The program has been terminated when the time exceeded 3000000.

Cost	Cut
Min	40
Median	85
Max	137
Mean	86

```
PROC Adaptive Simulated Tempering
  BEGIN SEQUENTIAL
    $T \leftarrow T_0$
    generate a starting solution $s$
    WHILE (NOT frozen) DO
      Generate temperatures for tempering
      $s \leftarrow$ minimal solution found so far
      DO $k \leftarrow 1, K$
        Update the temperature
        DO $i \leftarrow 1, L*$ Number of Nodes
          generate a new solution $t$ in the
            neighborhood of $s$
          (swap one node to the other part)
          $\Delta E \leftarrow E(t) - E(s)$
          IF $\Delta E < 0$ THEN
            $s \leftarrow t$
          ELSEIF $e^{-\frac{\Delta E}{T_k}} <$ random $[0, 1]$ THEN
            $s \leftarrow t$
          END IF
          $time \leftarrow time + 1$
        END DO
      END DO
    END WHILE
  END SEQUENTIAL
END PROC
```

Figure 23: Adaptive Simulated Tempering Algorithm used for the Experiments

5 Line Bisection

Recursive bisection is a very fast partitioning strategy. It is often used for dividing points in a two dimensional plane into a number of partitions containing an approximately equal number of points. The membership of a point to a partition is determined by its location in the two dimensional plane. By specifying the number of planes and their linear order for each plane a variety of real problems can be solved with minimum effort.

One realistic application is the VLSI circuit simulation [3] where the computational core is dominated by the simulation of the transistors. To increase the speed of the simulation the goal is to distribute the calculations onto different processors of a parallel machine. To achieve load balancing the recursive bisection technique is used. On each processor should be mapped an equal number of transistors for the simulation to achieve a minimal execution time for the simulation.
A parallel bisection algorithm can be found in [15] and [16].

5.1 The Bisection Algorithm

The Figure 24 explains best the functionality of the bisection algorithm. First, an angle is chosen and than the points of the plain are divided in such a way that half of them are placed on one side of the line and the other points are placed on the other side.

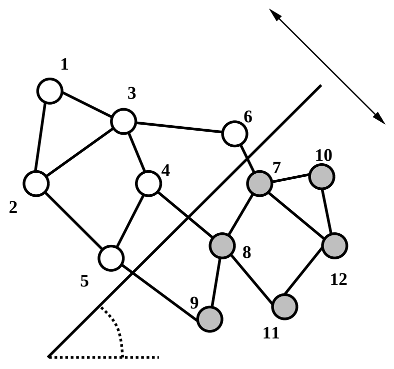


Figure 24: Example of a Line Bisection

A value is attached to each point in the plain specified by the following function dependent on the angle α. Let (x_i, y_i) denote the position of point i in the plane than the point

$$x'_i = \cos(\alpha)x_i - \sin(\alpha)y_i$$

specifies a rotation of the x coordinate with the angle α. The points are now sorted in respect to the new coordinate value x'_i. This algorithm has the complexity $O(n \log n)$ while using the quicksort algorithm. Instead of using a sorting algorithm one could use a median finding algorithm, reducing the complexity to $O(n)$ [1].

5.2 Quality and Performance of the Bisection Algorithm

The bisection algorithm introduces a way to visualize its solution space by varying the angle α in the interval $[0, \pi]$. We show here the results for the geometric graph with 500 nodes and the grid with 400 nodes (Figures 25 and 26). The cost values for two hundred different angles are displayed.

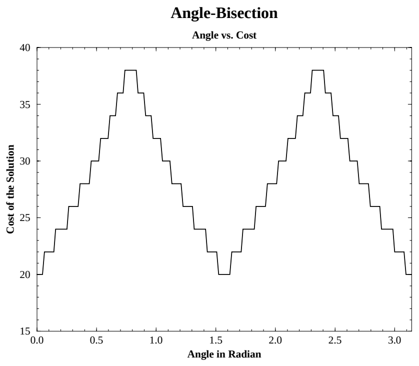


Figure 25: (Grid 400) Cost of the solutions found by the bisection algorithm using different angles

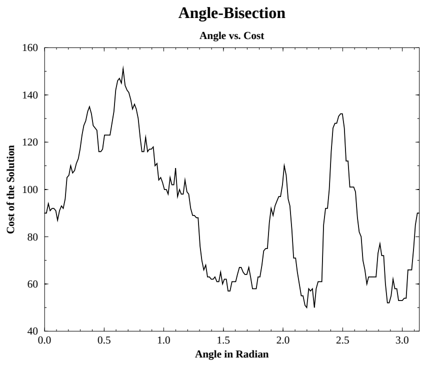


Figure 26: (Geometric 500) Cost of the solutions found by the bisection algorithm using different angles

Because a slight change in the angle for the grid graph does not lead to a new solution a step function occurs (Figure 25). The best solution found has the cost of 20 and is also the global minimal solution for this problem instance. The symmetry inherented in the solution (the solution is 90 degree rotation invariant) can be examined from the Figure 25.

More interesting is the graph with 500 nodes (Figure 26). Small plateaus at about the angles 0.1, 1.5, 2.1, and 3.0 are obvious. Nevertheless, the structural nature of the graph is not anymore so obvious from the function values. The best solution found with varying angle is 50.

To improve the algorithm in an adaptive way regions with nearly the same cost value can be examined at higher resolution of the angle α in the hope of finding a better solution. Better is to use other algorithms, like the Two Optimal or the Kernighan-Lin algorithm[7] to improve the solution.

The Figure 27 shows the best, average and worst cost values obtained while combining the Line Bisection and the Two Optimal improvement algorithm. For this problem instance the result is very stable and in each case a significant improvement could be achieved.
The question arises if one can improve the algorithm while using a better improvement algorithm. The Figure 28 shows in addition the result found by combining the line section algorithm and the Kernighan-Lin improvement algorithm.
In spite of an average improvement the quality of the solution for specific angles the overall cost could not be improved. Furthermore, the running times of of the combined two-opt algorithm is about 1/3 of the running time of the combined KL algorithm (Table 2).
The performance and quality of the line bisection algorithm is dependent on the problem instance. Often other algorithms have to be used because the geometrical information is not available or lead to bad results if the geometrical structure inherented in the graph is not useful for the line bisection.

	Time/angle	Best Solution found
Bisection	0.13	50
Bisection+2opt	0.72	26
Bisection+KL	2.45	26

Table 2: (Geometric 500) Comparison of the running times for the Bisection algorithms

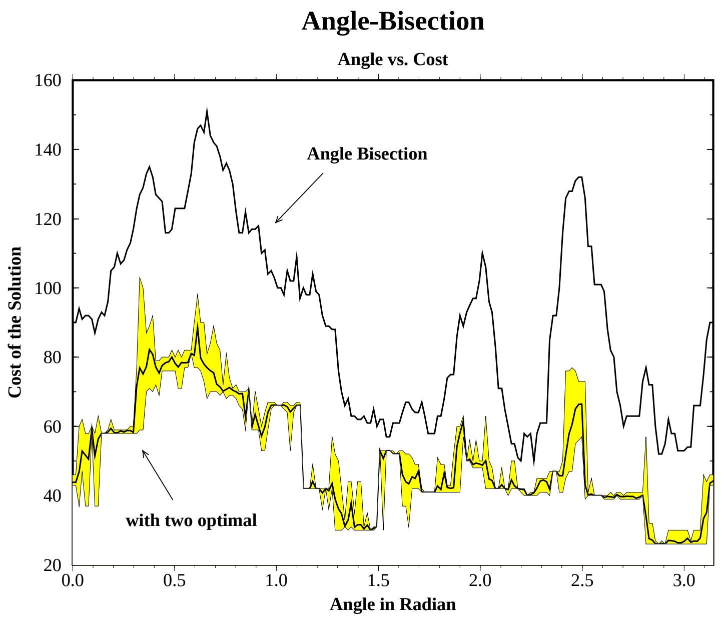


Figure 27: (Geometric 500) Comparison of Line Bisection with Two Optimal improvement. The shaded area specifies the worst,average, and best cost value found for a number of experiments

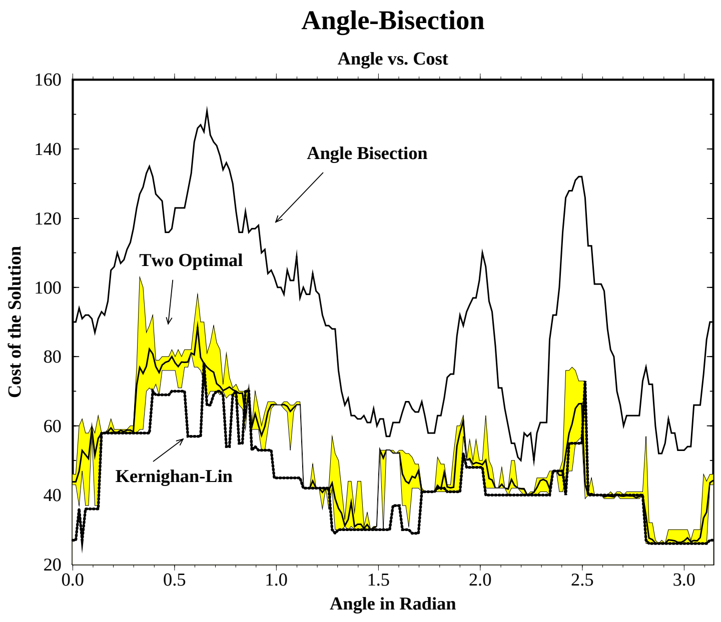


Figure 28: (Geometric 500) Line Bisection with Kernighan-Lin improvement

6 Graph Reduction

This algorithm makes use of the geometrical information included in the graph. In the line section algorithm we made use of the geographic information of the nodes. The reduction algorithm makes use of the inherented neighborhood information given by the edges of the graph. The purpose of a graph reduction algorithm is to reduce the number of nodes of the graph participating in the application of an improvement algorithm leading to reduced running times (compare also the size of the solution space as included in the appendix). Two strategies are possible:

- The first strategies uses a subgraph of the original graph to start an improvement algorithm like SA or the Kernighan Lin Algorithm on this set of nodes. An example of such a strategies is the parallel k-way partitioning proposed by Moore[11].
- The second strategy reduces simply the number of nodes in the graph in such a way that some nodes are joined in hypernodes. After reducing the graph a standard heuristic can be applied to find a partition to the reduced graph.

To reduce the nodes of a given graph a mapping function

$$\Re_R : (V, E, w) \longrightarrow (V', E', w')$$

is constructed with the properties $|V'| < |V|$ and $|E|' < |E|$. To define such a mapping groups of nodes are determined which will be viewed in the reduced graph as a single node called *hypernode*. The degree of reduction R specifies the maximal amount of nodes placed in one hypernode.
Since it is possible that the degree of a node v is smaller than R the set of neighbors should be extended. Let $N(v)$ be the set of neighbor nodes to node v. The *neighborhood set* $N_r(v)$ contains all nodes such that there is a path from v which is not longer than r
One simple way to construct the neighborhood set is given by the recursive construction formula:

$$N_r(v) \overset{\text{def}}{=} \begin{cases} N(v) & \text{falls } r = 1 \\ N_{r-1}(v) \bigcup\limits_{v' \in N_{r-1}(v)} N(v') & \text{otherwise} \end{cases} .$$

Randomly a node $v \in V$ is selected. For this node the neighborhood set $N_r(v)$ with an arbitrary but fixed r is determined such that $|N_r(v)| \geq R - 1$. Next $R - 1$ nodes form $N_r(v)$ are selected. These nodes together with v are building a hypernode in the new graph. In case that there exists no r with $|N_r(v)| \geq R - 1$ only the node v together with the $|N_r(v)|$ nodes will be joined to a hypernode. The nodes joined in the hypernode and all their corresponding edges are removed from the original Graph (V, E, w). A new node is selected from the remaining subset and the above described algorithm is repeated. This is done as long as all nodes are removed from the set of nodes V.
After determining the nodes of the reduced graph the edges have to be updated. Using the original graph the nodes are now joined to a hypernode. While building the hypernodes multiple edges and self-loops occur. Self-loops are simply deleted and multiple edges from one node to the other will be joined by adding their weight. The result is a *simple* graph.

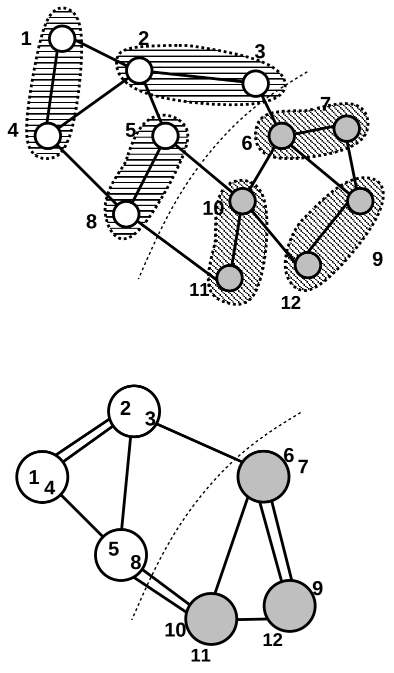


Figure 29: Example of a Graph Reduction

The Figure 29 explains how to find a reduction mapping function $\Re_R$. The Graph with 12 nodes is reduced with $R = 2$. First the hypernodes are determined as shown in the top part of the figure and than the edges are updated.
Now one can apply standard heuristics to find a partition for the reduced graph.
One disadvantage is immediately obvious:

> It can not be guaranteed that the hypernodes contain the same number of nodes resulting in problems to maintain the condition of the load balance between the partitions.

This can be overcome e.g. by the following strategies:

1. A simple greedy heuristic is applied on the resulting partitions as used in this paper.
2. A random remapping algorithm is applied as introduced on [14]. This algorithm uses the following two phases:
   A. After the graph reduction certain nodes (e.g. from the node cut set) will be remapped such that the partitions are equally large.
   B. Now the solution is not anymore optimal, but applying the standard heuristic e.g. Two Optimal improves the solution.

Very important is to notice that in the reduced graph many nodes have been joined to a hypernode and belong to the same partition. Therfore, it is unnecessary to apply an improvement heuristic on the whole set of nodes. Often it is sufficient to apply it only on nodes in the *cut* set $V_{cut}(P)$ (see appendix for definition).

```
PROC Reduction-2-Opt:
  BEGIN SEQUENTIAL
    reduce the graph $G \xrightarrow{\mathcal{R}} G'$
    generate randomly a partition for the reduced graph
    apply 2-Opt $(G' = (V', E', w'), \mathcal{P}'_m)$
    expand $\mathcal{P}'_m$ onto $\mathcal{P}_m$
    improve the balance between the parts $P_{a \in [1,m]}$
    apply Two Optimal $(G_{cut} = (V_{cut}, E, w), \mathcal{P}_m)$
  END SEQUENTIAL
END PROC
```

Algorithm 1: Combination of graph reduction with the two optimal algorithm

```
PROC Reduction-2-Opt:
  BEGIN SEQUENTIAL
    reduce the graph $G \xrightarrow{\mathcal{R}} G'$
    generate randomly a partition for the reduced graph
    apply greedy $(G' = (V', E', w'), \mathcal{P}'_m)$
    expand $\mathcal{P}'_m$ onto $\mathcal{P}_m$
    apply greedy $(G_{cut} = (V_{cut}, E, w), \mathcal{P}_m)$
  END SEQUENTIAL
END PROC
```

Algorithm 2: Combination of graph reduction with the greedy algorithm

Naturally many combinations of these algorithms are possible two of them are outlined in Algorithm 1 and 2.
This combination of reduction and improvement algorithms finds fast very good solutions.

Complexity To determine for one node the set $N_r(v)$ maximal $O(|E|)$ steps are necessary resulting in $O(n|E|)$ for all nodes. Updating the edges and weights is done in $O(|E|)$ steps resulting in $O(n|E|)$ steps for the reduction. For the overall algorithm the complexity of the improvement algorithm and the reduction algorithm must be added.

6.1 Performance of the Reduction Algorithm

In the Figures 30 to 33 the performance of the reduction algorithm is shown using the geometric graph with 500 nodes.
The Figure 31 shows Box-Whisker diagrams for the solutions found with different degree of reduction. In the Figure 30 we show a closeup by ignoring the maximal cost value. First we observe that with increasing degree of reduction the median decreases. Nevertheless, the range of the solution found increases, because of the problem of the imbalance inherented in the simple reduction algorithm. This is also the reason why the mean increases with a higher degree of reduction.
The big advantage of the reduction algorithm is the time spend to find a solution. In Figure 32 we show the range of the time spend to find a solution for different degrees of reduction. The Figure 33 shows the

average time spend in the graph reduction and the partitioning of the reduced graph. We can see clearly that the time spend for the reduction is small and for larger degrees of reduction almost the same, based on the linear complexity of the reduction strategy. The time spend to find a partition for the reduced graph is very small.

The overall time spend for finding a partition is almost the same for degrees higher than 9.

To outcast the effect of the big ranges for larger degrees of reduction the algorithm should be repeated a couple of times in order to find a very good result. In the appendix we included results found for more than 2 partitions.

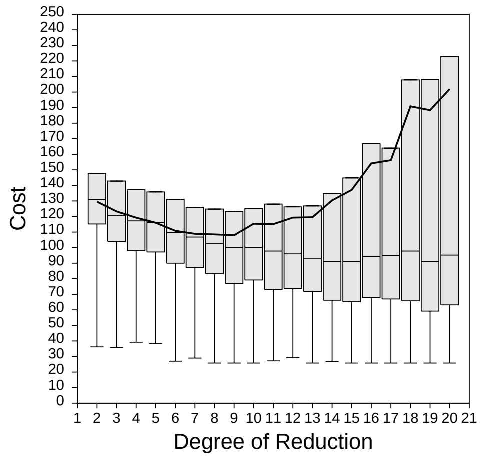


Figure 30: Reduction Algorithm applied to the graph with 500 nodes and different degree of reduction

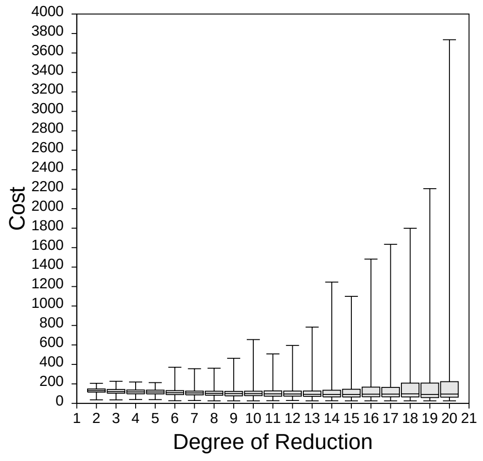


Figure 31: Reduction Algorithm applied to the graph with 500 nodes and different degree of reduction

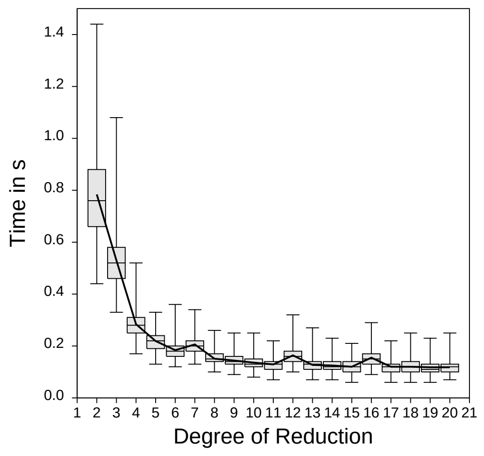


Figure 32: CPU Time used by the Reduction Algorithm using an IBM RS/6000

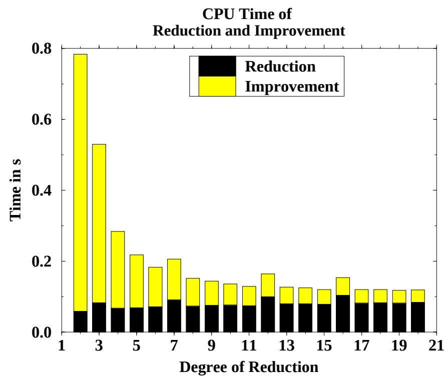


Figure 33: CPU Time used by the Reduction Algorithm using an IBM RS/6000

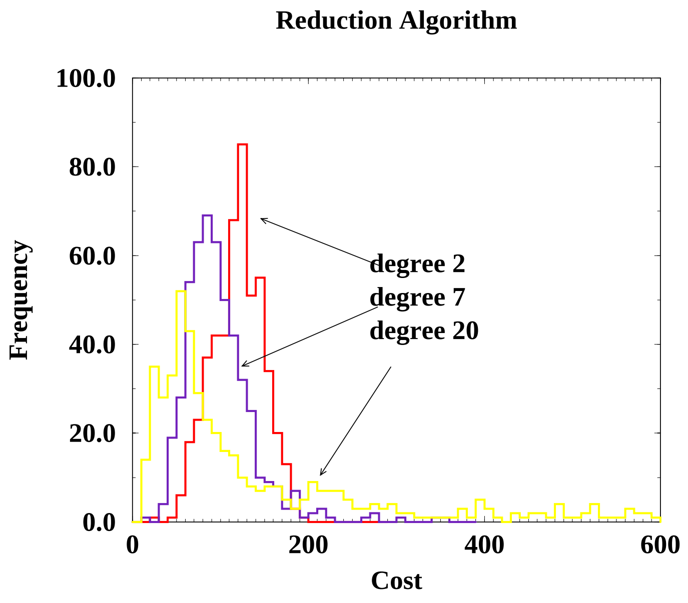


Figure 34: The frequency distribution with different degree of reduction

6.2 Stepping down in the Solution Space

The Figure 35 displays the frequency distribution of the costs found with the Two Optimal, Kernighan Lin and Simulated Annealing Algorithm. We see clearly that the Simulated Annealing algorithm performs the best.

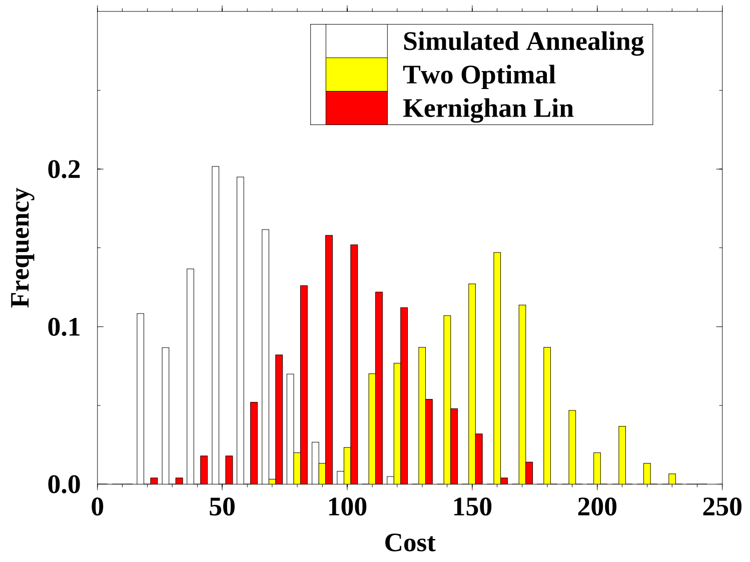


Figure 35: The frequency distribution of different heuristics

An interesting aspect is if it is possible to eliminate possible solutions out of the solution space in order to increase the speed to find a very good solution. The heuristics described so far make only use of the assignment of nodes to particular partitions. The question arises:

- Do we gain information about good solutions if we observe the edges in the cutset?

To answer this questions all solutions of a particular heuristic have been stored in a file. Than a frequency diagram on the number of edges are generated in the following way:

1. Only solutions within the interval $[border_{min}, border_{max}]$ are considered.
2. Set the weight of all edges to zero.
3. Whenever an edge is element of the cut set the weight of the edge is increased by one.

The idea behind this edge analysis is to see if there are specific edges favored in good solutions. These edges should be used in a self learning graph partitioning algorithm making use of knowledge gained from the past. Furthermore, it could be possible to animate the step through the solution space to find the regions of change from bad to good solutions.
The Figure 36 and F:edge-f-2opt-100 show such frequency graphs for the heuristic *Two Optimal*. We see clearly that in better solutions certain edges do not occur anymore. Interesting is to observe that several nodes do not have edges in the cut set indicating that this node does should be reduced with its neighbor

nodes in order to reduce the number of nodes of the graph. Since its property is inherented in the graph it is clear why the graph reduction algorithm works so well.
For the better heuristics Kernighan Lin and Simulated Annealing the effect is even better visible. Already eliminations edges and nodes not occurring in a number of bad solutions would help to reduce the search space.

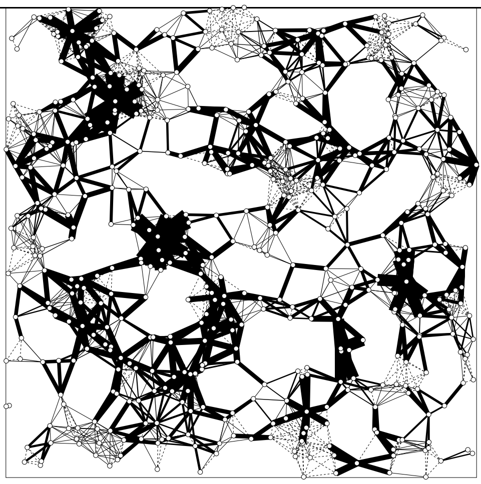


Figure 36: Frequency analysis of the edges in all generated solutions by the heuristic Two Optimal

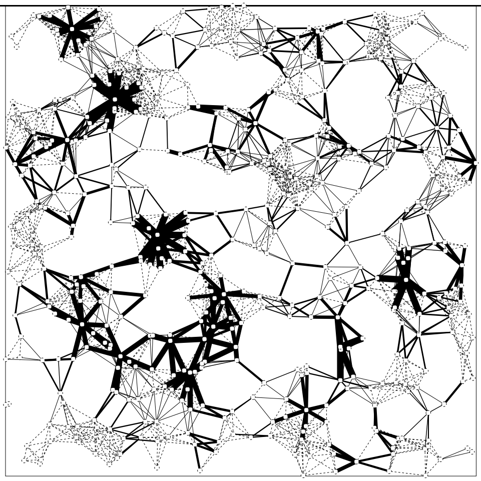


Figure 37: Frequency analysis of the edges in all generated solutions in the interval [0−100]

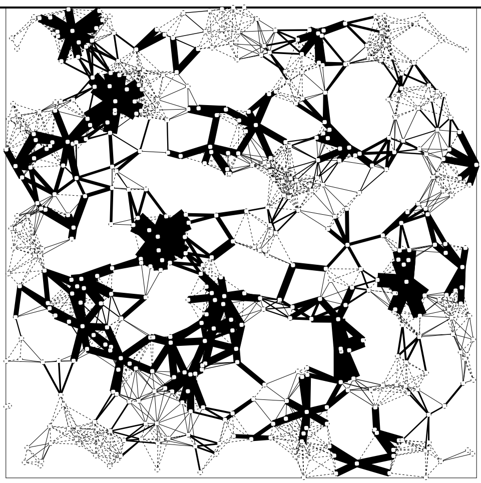


Figure 38: Frequency analysis of the edges in all generated solutions by the heuristic Simulated Annealing

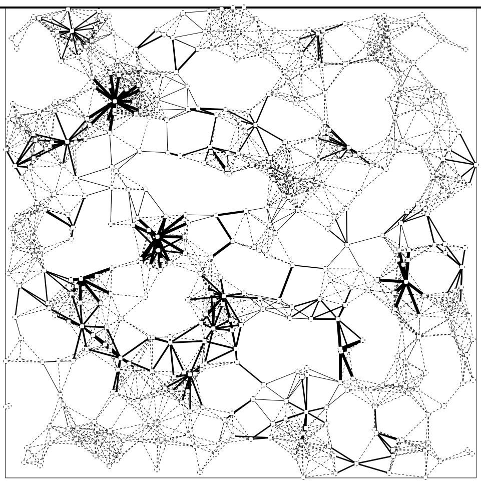


Figure 39: Frequency analysis of the edges in all generated solutions in the interval [0−30]

The Figure ?? shows the frequency with which particular edges occur in all solutions generated with the heuristic Two Optimal and Simulated Annealing. It is obvious that in case of the Simulated Annealing algorithm certain edges occur more frequently but others less frequently. Almost 2700 edges never occur in the solutions which motivates that the graph is structured and the search in the solution landscape is

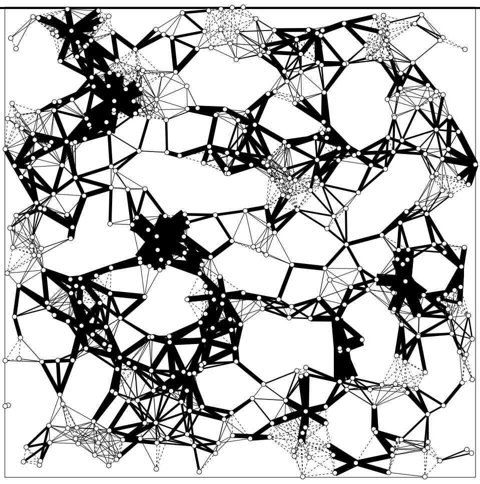


Figure 40: Frequency analysis of the edges in all generated solutions by the heuristic Kernighan Lin

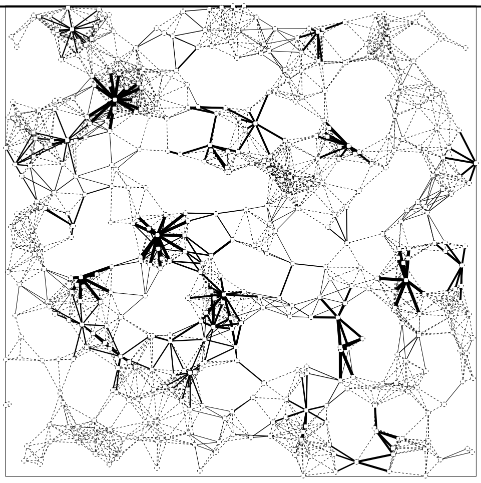


Figure 41: Frequency analysis of the edges in all generated solutions in the interval [0−40]

directed.

The line with SA-resorted displays the sorted frequency of all edges in the simulated annealing. As expected less edges occur in the possible solution favoring a specific direction. One interesting aspect of this function is the step occurring at a frequency value of 0.5. So far we did not look into this aspect.

Now we can answer our questions from the beginning:

- The edge analyses of good solutions shows that information can be obtained about the edges which are likely to be in the cutset. Removing nodes not participating as a neighbor of the edge set and introducing hypernodes with variable sizes could leeds to a heuristic finding very good results.

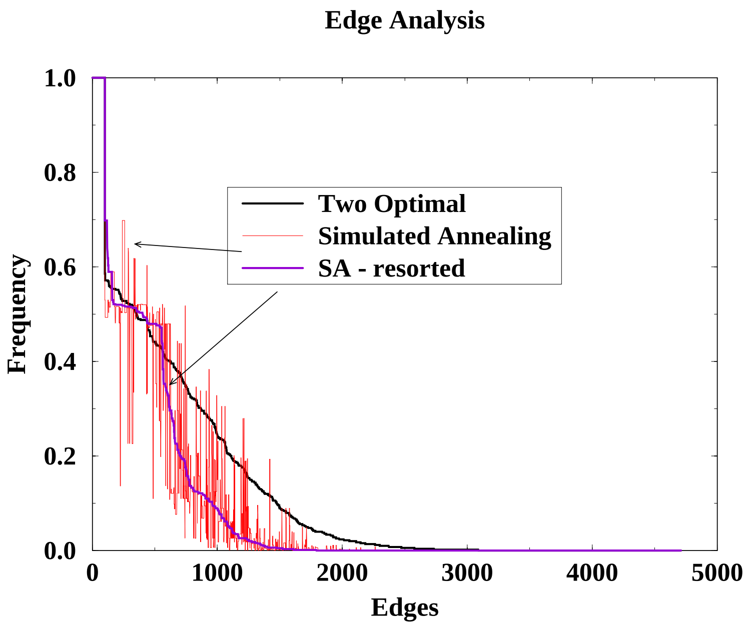


Figure 42: The frequency of edges occurring in all solutions

7 Parallelization of the Algorithms

Some of the algorithms introduced here are from sequential nature. This is in especially valid for the Simulated Annealing, the Kernighan Lin and the Two Optimal algorithm. The Line Bisection algorithm could be easily parllelized by computing different ranges of angles on different processors.
Since we wanted to have enough statistical data we repeated the same experiment a couple of times. Therefore, we used a network of 6 IBM RS6000 and a network of 8 SUN SPARC workstations to obtain quickly our results. We did not use the CM-5 with 32 processors available at NPAC to do this evaluations since this machine has been heavily used by other users at the time of the experiments.

References

[1] Cormen, T., Leiserson, C., and Rivest, R. *Introduction to Algorithms.* McGraw Hill, 1990.

[2] Faigle, U., and Schrade, R. Simulated Annealing – Eine Fallstudie. *Angewandte Informatik*, 6 (June 1988), pp. 259–263. (in german).

[3] Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D. *Solving Problems on Concurrent Processors.* Prentice Hall, New Jersey, 1988.

[4] Garey, M. R., and Johnson, D. S. *Computers and Intractability.* W.H. Freeman and Company, San Francisco, CA, 1979.

[5] Iwainsky, A., Canuto, E., Taraszow, O., and Villa, A. Network Decomposition for the Optimization of Connected Structures. *Networks 16* (1986), 205–235.

[6] Johnson, D. S., Aragon, C. R., and McGeoch, L. A. Optimization by Stimulated Annealing: An Experimental EvaluationPart I, Graph Partitioning. *Operations research Vol. 37*, No. 6 (Nov. 1989).

[7] Kernighan, B., and Lin, S. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal Vol. 49*, No. 2 (1970), pp. 291–308.

[8] Kirkpatrick, S. Optimization by Simulated Annealing: Quantitative Studies. *Journal of Statistical Physics 34* (1984), pp. 975–986.

[9] Kirkpatrick, S., Gellatt, C. D., and Vecchi, M. P. Optimization by Simulated Annealing. *Science 220* (1983), pp. 671–680.

[10] Marinari, E., and Parisi, G. Simulated Tempering: A New Montecarlo Scheme. *Europhysics Letters*.

[11] Moore, D. A Round-Robin Parallel Partitioning Algorithm. Tech. Rep. 88–916, Cornell University, 1988.

[12] Papadimitriou, C. H., and Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall Inc., 82.

[13] von Laszewski, G. A Genetic Algorithm for the Graph Partitioning Problem. Master's thesis, University of Bonn, Nov. 1990. (in German).

[14] von Laszewski, G. Intelligent Structural Operators for the k-way Graph Partitioning Problem. In *Proc. of the 4th intern. Conf. on Genetic Algorithms* (July 1991), Morgan Kaufman. plenary presentation.

[15] von Laszewski, G. Object Oriented Recursive Bisection on the CM-5. Tech. Rep. SCCS 476, Northeast Parallel Architectures Center at Syracuse University, April 1993.

[16] von Laszewski, G. Object Oriented Recursive Bisection on the iPSC860. Tech. Rep. SCCS 477, Northeast Parallel Architectures Center at Syracuse University, April 1993.

A Sequential Heuristics for more than 2 partitions

The following sequential heuristics can be applied also on graphs which should be partitioned into more than 2 parts.

2-Opt: The heuristic Two Optimal takes two nodes randomly and exchanges them if it leads to an improvement in the cost. A solution generated by this algorithm is called 2-optimal.

KL: The heuristic from Kernighan and Lin takes a sequence of exchanges and does only the one which leads to a maximal cost improvement in this sequence. The algorithm is in detail described in [7, 12].

Reduction: The reduction algorithm is followed by the 2-Opt algorithm in order to improve the result. The description of this algorithm can be found in in an earlier section. The degree of reduction R specifies how many nodes should be joint to one in the original graph.

Because of the low running times for the smaller problems many experiments are conducted. Unfortunately, this has not been completed for the bigger problems

Graph with 78 Nodes

First, we show the effect of the degree of reduction on the runtime and the quality of the solution. To generate a sufficiently large testbed the reduction algorithm is repeated 1000 times for the various parameters.

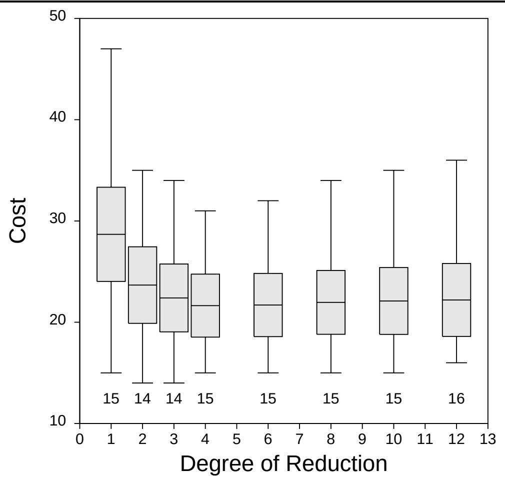


Figure 43: (Graph 78) Quality of the solution found with different reduction degrees

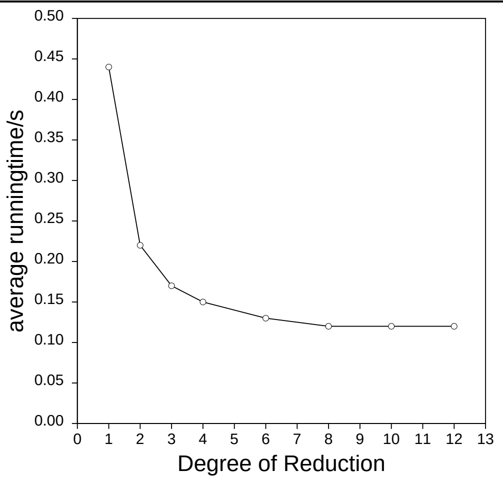
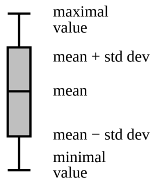


Figure 44: (Graph 78) Comparison of the average runtime on a SUN3

The Figure 43 shows the dependency between the degree of reduction and the quality of the solution found and the Figure 44 shows the average runtime necessary to obtain a solution. In Figure 43 a five values are displayed for each degree of reduction:



The top and bottom value represent the maximal values found in the experiments. The block in the middle is determined by the standard deviation and the line in the middle represents the mean for the number of experiments.
This way of displaying is used to show the stability of the method while using different parameter settings. The closer the values are the more stable the method is.
The best result is obtained at a degree of reduction of 2 and 3 Choosing the reduction degree to high does not lead to better results, since now to many nodes are combined to a hypernode introducing a big imbalance in the solution. This imbalance is repaired by th following strategy:

1. distribute some nodes randomly to new partitions so that the imbalance is minimized.

2. execute the 2 optimal algorithm on this solution

It is clear that this disturbance of the solution results also in high running times for large reduction degrees caused by the repair of the solution. This repair needs more time for more disturbed solutions.
The Figures 45 and 46 compare the distribution of solutions obtained with the algorithms 2-Opt, KL and Reduction of degree 2 and 3.
It is obvious that the reduction algorithm can produce a distribution similar to the KL algorithm. Nevertheless for this problem instance the frequency for obtaining the minimal solution is smaller than for the KL algorithm. But the running time for the algorithm is up to 12-17 faster (Table 3).

Algorithm	minimal Cost	maximal Cost	$\overline{\text{Cost}}$	σ(Cost)	$\overline{\text{runtime}}$ in s *
Random	83	117	100.97	5.07	0.02
2-Opt	15	47	29.01	4.63	0.65
Reduction (R=2)	14	35	23.67	3.78	0.35
Reduction (R=3)	14	34	22.40	3.35	0.25
KL	14	31	22.54	3.62	4.43

* CPU-time on a SUN/3

Table 3: Timings for sequential heuristics for the problem with 78 nodes

A Grid With 100 Nodes

The Figure 47 shows the range of solutions obtained with different sequential heuristics. For the reduction algorithm the value R represents the degree of reduction. Also for this problem instance the reduction algorithm was able to generate the optimal solution in very short time.

A Grid With 400 Nodes

The reduction algorithm could obtain very good results for smaller graphs. Nevertheless, for bigger graphs the quality of the solution obtained is not anymore so good.
For example, the Figure ?? displays the range of solutions found for a grid with 400 nodes. The algorithm was not able to detect the global optimum.

Result

For small graphs the reduction algorithm finds very good results in short time. This holds also for larger graphs, inspite of the fact that this solutions are not anymore so good as compared to the small problems. Is the running time of partitioning algorithm a mayor factor the reduction algorithm is a very feasible heuristic for the problem instances used in this paper.

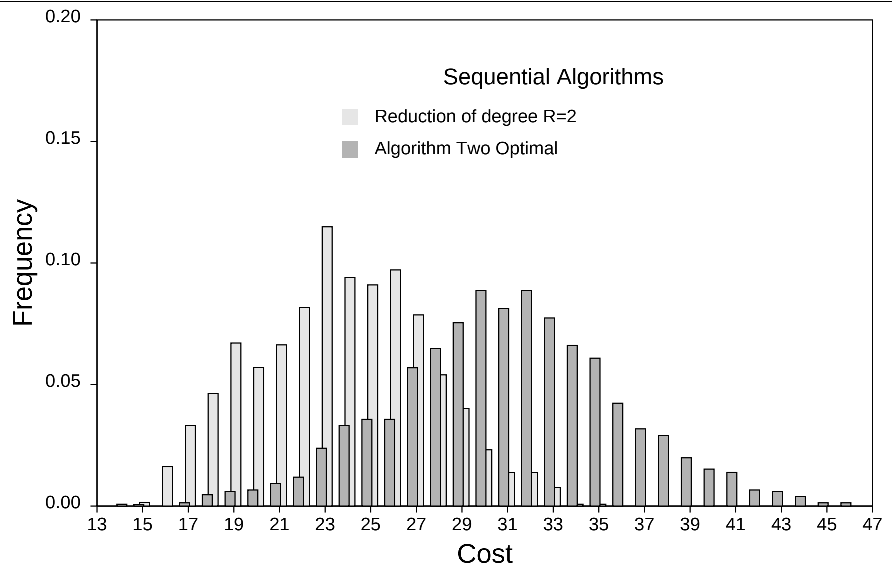


Figure 45: (Graph 78) Distribution of the cost of solutions generated with the heuristics Two Optimal and reduction of degree 2

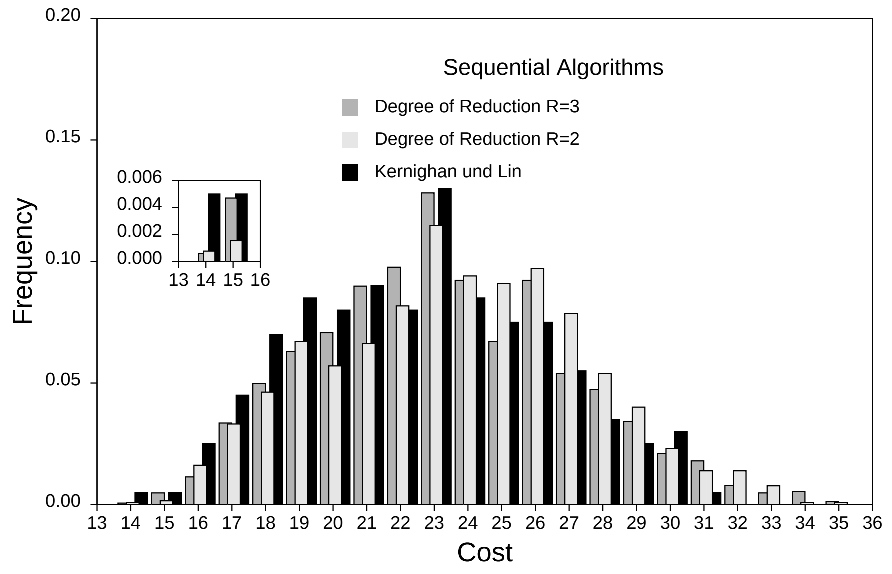
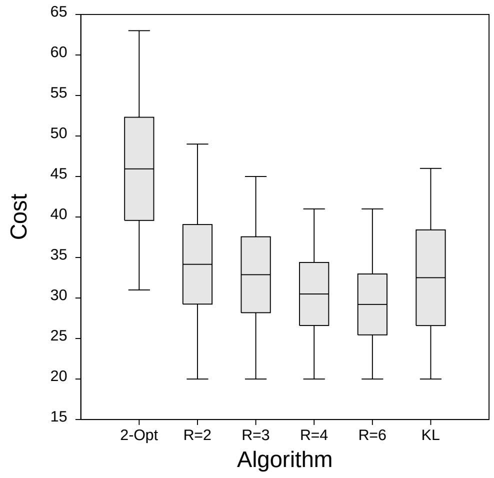


Figure 46: (Graph 78) Distribution of the cost of solutions generated with the heuristics Kernighan Lin and reduction of degree 2 and 3



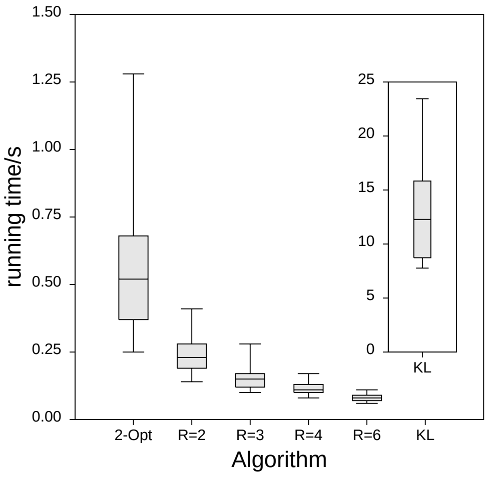


Figure 47: (Grid 100) Quality of the solution found with different reduction degrees

Figure 48: (Grid 100) Comparison of the average runtime on a SUN3

Figure 49: (Grid 400) Comparison of the running times

Figure 50: (Grid 400) Comparsion of the solution quality

B Definitions and Properties

Graph 1 (Graph)

A Graph G *is a triple (V, E, w), where V and E are disjunct ordered finite sets. The elements from V are called* nodes *and from E are called* edges. *An element from E is a tuple from elements of V and describes a connection between two nodes.*

$$\begin{aligned} G &= (V, E, w) \\ V &= \{v_1, v_2, ..., v_n\} \\ |V| &= n \\ E &\subseteq V \times V \end{aligned}$$

The function w describes the weights of the edges:

$$w : E \mapsto \mathbb{N} \quad .$$

The set $N(v)$ are the immediate neighbors of a node v.

$$N(v) \quad = \quad \{u | (u, v) \in E \vee (v, w) \in E\} \quad .$$

The *Degree* $\Gamma(v)$ of a node specifies the number of edges pointing to this node:

$$\Gamma(v) \quad = \quad |\{u | (u, v) \in E\}| \quad .$$

Partition 1 (Partition)

Let V be a set with n elements, $m \in [1, n]$ and the family P_a with $a \in [1, m]$ non empty subsets of V. The set $P = \{P_1, ..., P_m\}$ is a m-partition of V, iff:

1. *The partition P contains all elements of V:*

$$\bigcup_{a \in [1,m]} P_a = V \quad .$$

2. *All partitions in P are disjunct:*

$$P_a \cap P_{a'} = \{\} \qquad \forall a, a' \in [1, m] \quad \wedge \quad a \neq a' \quad .$$

A P_a is called Partition. *There are exactly m partitions. For the uniform graph partition problem $m = 2$.*

Mapping 1 (Mapping)

The Mapping function M_P describes which node is member of which partition.

$$M_P \;:\; V \;\mapsto\; \{P_a | a \in [1, m]\}$$

with

$$M_P(v) = P_a \Longleftrightarrow v \in P_a \quad .$$

Cut 1 (Cut)

$$E_{cut}(P) \;=\; \{(v_i, v_j) | (v_i, v_j) \in E \quad und \quad M_P(v_i) \neq M_P(v_j)\}.$$

$$V_{cut}(P) \;=\; \{v_i | \exists v_j \in V \quad mit \quad (v_i, v_j) \in E_{cut}(P)\}$$

Properties

The Graph partitioning problem is NP complete [4]. The size of the search space for an arbitrary problem instance is defined by the number of nodes and partitions.
Let $|P_i| = \frac{n}{m}$ be the size of a partition than there are $\binom{n}{|P_i|}$ possible choices for the first partition, $\binom{n-|P_i|}{|P_i|}$ for the second and so forth. Since the order of the partitions is not considered the total number of solutions is

$$\frac{1}{m!}\binom{n}{|P_i|}\binom{n-|P_i|}{|P_i|}\ldots\binom{2|P_i|}{|P_i|}\binom{|P_i|}{|P_i|} \quad .$$

For a graph with 120 nodes and 4 partitions which we consider as a *small* problem more than $1.126 * 10^{68}$ solutions are available not considering the imbalance term. If one includes a possible imbalance in the solution we have a total of m^n solutions.