ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics
Bachelor Thesis [Econometrics & Operations Research]

Adaptive Kernel Search: A General-Purpose MIP Heuristic

Name student: Karel Bleichrodt
Student ID number: 485093

Supervisor: Rowan Hoogervorst
Second assessor: Prof.dr. Dennis Huisman

Date final version: July 5, 2020

Abstract

I investigate the *Adaptive Kernel Search* (AKS) heuristic introduced by Guastaroba et al. (2017). AKS is designed to provide solutions to general Mixed Integer Programs (MIPs). The heuristic works with a set of promising variables, named the *kernel*. The promising variables are likely to be non-zero in an optimal solution. According to the kernel, sub-problems of the MIP are constantly solved. AKS offers options to manage the trade-off between a high solution quality and a low computational time. The CPLEX solver with a time limit of 1 hour and 5 hours was utilized as a benchmark method. The results indicate that the solution quality of AKS is slightly inferior to the solution quality of CPLEX, but AKS can be associated with a reduced computation time.
I introduce an extension to AKS. The extension attempts to establish an improvement, by resolving a weakness of the current AKS framework. The extension variant appears to compete well with the current framework of AKS. To obtain a representative image of the performance of the extension variant, further research is required.

Contents

1 Introduction

Linear Programming (LP) optimizes a maximization or minimization problem subject to numerous linear constraints. LP problems only contain continuous variables, variables that can take any possible value between the lower bound and the upper bound. Mixed Integer Programs (MIPs) extend Linear Programs, as they also contain variables that are exclusively allowed to take an integer value. A large variety of logistics problems can be formulated as MIPs. Examples of such logistics problems are Vehicle Routing Problems, one-dimensional or multi-dimensional Knapsack Problems, Facility Location Problems and Travelling Salesman Problems. MIP problems have many applications such as the delivery of goods, resource allocation, telecommunication networks, electricity regulation, agricultural or industrial production planning etc.

In this thesis, I aim to replicate the results of Guastaroba et al. (2017). They developed a heuristic called *Adaptive Kernel Search* (AKS), which is a heuristic for MIPs. AKS is an extension of *Kernel Search* (KS), which was introduced by Angelelli et al. (2010). KS maintains a set of promising variables, named the *kernel*. The promising variables are likely to be non-zero in an optimal solution. KS consists of two main stages. In the first stage, the kernel is initialized and a feasible solution is obtained. The second stage concerns improving the solution. According to the kernel, sub-problems of the actual MIP are constantly solved.

A disadvantageous point of KS is that the heuristic requires problem-specific tuning. For separate problems, alternative parameter settings need to be examined in order to perform well. AKS resolves this issue. Additional to the two stages of KS, AKS appends an extra stage. The extra stage assesses the specific instance and adapts the heuristic, according to certain characteristics of the instance. This makes AKS well applicable for any arbitrary MIP, therefore having an exceptionally wide reach of practical applications.

Guastaroba et al. showed that their heuristic has numerous advantages over solving MIPs using the CPLEX solver. AKS can be used to achieve, on average, better solutions than CPLEX, given that both methods use a time limit of 5 hours. By changing a few parameters, several variants of AKS can be implemented easily. These variants provide great flexibility in the well-known trade-off between solution quality and computational time. CPLEX with a time limit of 5 hours was compared to a variant of AKS that has a strong focus on a relatively short computational time. This variant of AKS was able to drastically reduce the computational time at the expense of a small deterioration in the quality of the solutions.

Guastaroba et al. also compared AKS to *Variable Decomposed Neighbourhood Search* (VNDS), a state-of-the-art heuristic by Lazic et al. (2010). AKS outperformed VNDS on the large set of instances tested. Analyzing the different variants of VNDS and the different variants of AKS, the AKS variants proved to be superior in solution quality as well as in computational time. The limited time of seven weeks that is reserved for this thesis excluded the possibility of generating the VNDS results. Therefore, AKS will solely be compared to solving MIPs using the CPLEX solver.

Besides aiming to replicate the results, I will propose an extension of the current AKS framework. This extension modifies the stage wherein the search for an improving solution is executed.

The main research questions of this thesis are the following:

(1) *How does AKS perform on solution quality and computation time in comparison with the CPLEX solver?*

(2) *To what extent do my results match the results of Guastaroba et al. (2017)*

(3) *How does the extension perform on solution quality and computation time in comparison with the current AKS framework?*

Sub-questions that arise for the second main research question are:

(a) *What are possible explanations for any differences?*

(b) *Can these explanations fully account for the differences?*

To answer the research questions, I will run a subset of the instances that are tested in the replicated paper. These instances are predominantly obtained from the MIPLIB 2010 library (http://miplib2010.zib.de/miplib2010.php).

2 Literature Review

This section consists of two parts. First, the literature review on the search for a feasible solution is conducted. In the second part, developments in the field of general-purpose MIPs are discussed.

For complicated (large-size) MIP problems, it is often very challenging to find a feasible solution. An efficient heuristic, that is able to reduce the time before discovering a feasible solution, is desired. Fischetti et al. (2005) were among the first to tackle this issue, with the Feasibility Pump (FP) heuristic. The FP heuristic is centered around two points (say $\bar{x}$ and x^*). Both points partly satisfy the MIP feasibility criteria. $\bar{x}$ is feasible for the Linear Programming relaxation, but is not necessarily integer. x^*, on the other hand, is integer and not necessarily LP feasible. The obtaining of x^* is done through a simple rounding procedure. The idea of the FP heuristic is to iteratively reduce the distance between $\bar{x}$ and x^*, eventually arriving at a feasible solution. Since the introduction of the FP heuristic, a lot of effort has been made in order to explore further improvements to the FP scheme. Achterberg and Berthold (2007) have slightly altered the original FP scheme. Because the solutions of the FP heuristic were often poor, they focused on constructing a superior LP feasible solution ($\bar{x}$). Other authors have concentrated on ameliorating the integer solution (x^*). In the original FP scheme, x^* was obtained using a simple rounding procedure. In numerous papers, the attempt was made to improve the FP heuristic by considering more advanced rounding procedures. For instance, Fischetti and Salvagnin (2009) investigate the possible advantages of inserting a diving-like rounding procedure in the FP heuristic. This procedure is based on rounding and constraint propagation. Baena and Castro (2011) also introduce an alternative rounding procedure. Their *Analytic Center Feasibility Pump*(AC-FP) first determines the analytic center of the MIP problem. Subsequently, using a line segment, points that are candidates to become feasible solutions are explored. An interesting recent development in the area of finding a feasible solution is the paper by Adamo et al. (2020). The authors use a framework that resembles the approach introduced by Ghiani et al. (2015), who start from a feasible MIP solution and attempt to improve it. Ghiani et al. use two main phases. The outline is that the variables of the MIP are divided amongst several clusters in the first phase. In the second phase, each cluster is examined, where all variables outside the cluster are fixed to their value in the feasible solution that was supplied to the heuristic. Adamo et al. (2020) have altered this framework, such that it is able to start with an infeasible solution. The goal is no longer to constantly improve a solution, but merely to find a feasible solution. Adamo et al. have named their framework a *learn-and-construct* framework. On three well-known problems, the learn-and-construct framework outperforms the FP heuristic and another state-of-the-art MIP solver in both success rate and average computational time.

Now, the literature review regarding the concept of general-purpose MIPs is conducted. As Balas et al. (2001) note, compared to the amount of research on problem-specific MIPs, research on general-purpose MIPs is scarce. An overview of the major developments in the field of heuristics for general-purpose MIPs is discussed in the paper by Fischetti and Lodi (2011). Regarding this thesis, a notable method is the *Relaxation Enforced Neighborhood Search* (RENS), introduced in Berthold (2007). This heuristic makes use of a rounding procedure to simplify the MIP. RENS is used in the extension of this thesis and will be explained in more detail in the corresponding section. Guastaroba et al. (2017) state that, to the best of their knowledge, the *Variable Neighbourhood Decomposition Search* by Lazic et al. (2010) is the state-of-the-art heuristic for solving general 0-1 MIPs. VNDS forms sub-problems of the original problem. In each iteration, several variables are fixed, leading to a smaller problem. More specifically, all variables are sorted first. This sorting occurs according to the criteria of non-decreasing order of the absolute value of the distance between the variable's value in the LP relaxation and the variable's value in the incumbent solution. The first k variables of the sorted list are then fixed and the remaining sub-problem is solved. Lazic et al. have investigated numerous instances to assess the performance of VNDS. They concluded that, on the investigated set of instances, VNDS outperformed the CPLEX solver, as well as three other successful MIP solution methods. On the domain of general-purpose MIPs, the article by Gamrath et al. (2019) is a noteworthy recent addition. Gamrath et al. have created a framework that provides feasible solutions easily. This can reduce the time that a solver needs to prove optimality. The authors developed their methods around three types of global structures (*Cliques*, *Variable bound relations* and *Variable locks*) within MIP problems. According to the global structures, certain variables are fixed, leading to a smaller problem. They create three different heuristics, each based on a different type of global structure. The proposed heuristics managed to find a feasible solution for approximately 60% of the instances that were tested. The supplying of a feasible solution was accompanied by a reduced average solving time to optimality.

3 Methodology

Let B be the set of all binary variables and I be the set of all general integer variables of a MIP. If one attempts to solve this MIP in an exact way, all the variables in B and I need to be considered simultaneously. This often results in a substantial computation time for large instances. The AKS heuristic avoids this scenario. For the sake of convenience in explaining, I will first show the framework of AKS with only binary variables and then extend to incorporate the general integer variables.

First, the AKS heuristic aims to execute preliminary steps that result in finding a feasible solution and judging the difficulty of the instance. After that, the main idea is to divide the actual MIP into many sub-problems, in order to efficiently improve the current solution. Consider the previously mentioned set B that contains all binary variables of a MIP. Then, in AKS each sub-problem considers a subset $U \subset B$, where U is meant to consist of promising variables. Let MIP(U) denote the sub-problem wherein all variables in $B \setminus U$ are fixed to zero.

A crucial component of AKS is the *kernel*, which is a set of promising variables, and will be abbreviated by K. The binary variables that are not in the set K are divided into buckets. The sequence of buckets is $\{A_i\}_{i=1}^{NB}$, where NB is the total amount of buckets. Each sub-problem consists of solving MIP($K \cup A_i$), thus solely including the variables in K plus the variables in a single bucket A_i. Note that all continuous variables are included in the sub-problems. It is hard to determine an appropriate fixed kernel up front. Therefore, the kernel K is constantly adjusted by adding variables, as will be explained later in this section.

The pseudocode of the AKS heuristic is given in Algorithm 1. Throughout the explanation of this algorithm, z^{UB} represents the value of the best solution discovered so far by the heuristic and $\widetilde{NB}$ represents the maximum amount of buckets that are considered. The algorithm can be divided into three phases, the *Initialization Phase*, the *Assessment Phase* and the *Improvement Phase*.

Algorithm 1 AKS framework

```
(Initialization Phase)
Step 1: Solve the root node LP relaxation, create initial kernel $K$. The binary variables that are not in $K$ are added to an ordered list $L$.
Step 2: Solve MIP(K)
if MIP(K) is infeasible then
    Step 3: Apply the method getFeasible (see Algorithm 2)
end if

(Assessment Phase)
Step 4: Assess the difficulty of the instance
if Instance is easy then
    Step 5a: Adjust the kernel using the "easy procedure"
end if
if Instance is hard then
    Step 5b: Adjust the kernel using the "hard procedure"
end if
Step 6: Create buckets $A_i$ to insert the non-kernel variables ($i$ = 1,2, ... , $NB$)

(Improvement Phase)
while $i \leq \min\{NB, \widetilde{NB}\}$ do
    Step 7: Solve MIP($K \cup A_i$)
    Step 8: Use the result of 7 to update the kernel
end while
```

3.1 Initialization Phase

The goal of the Initialization Phase is to obtain a (as-good-as-possible) feasible solution. The first step of the Initialization Phase concerns solving the root node of the MIP. Recall that z^{UB} stands for the objective value of the current best solution that the heuristic has discovered. If solving the root node resulted in a feasible solution to the MIP, then z^{UB} is set to the corresponding objective value. Otherwise, z^{UB} is initialized as ∞.

Next, to form an initial kernel, the root node LP relaxation of the MIP is considered. The solution to the final linear program solved at the root node is denoted as x^{LP}. All binary variables that have a value larger than zero in x^{LP} are added to the kernel. The size of the initial kernel is denoted by the parameter m. The binary variables with a value of zero in x^{LP} are added to an ordered list L, sorted by non-decreasing reduced costs. These reduced costs comprise a slight difference between my implementation and the implementation in the original paper. The authors of the original paper used the reduced costs of the root node LP relaxation. Unfortunately, CPLEX does not provide a straightforward way to extract these. Instead, the reduced costs of the LP relaxation are used. The consequence of this difference is that the ordered list will likely be slightly different. This means that the sub-problems are non-identical, which might lead to a deviation in the results.

In the final step of the Initialization Phase, MIP(K) is solved, with z^{UB} as objective cutoff value. If this doesn't result in a feasible solution, the method getFeasible (see Algorithm 2 below) is invoked. This method keeps adding variables to the kernel, until MIP(K) becomes feasible. Here, w is a pre-specified parameter. In getFeasible, the objective cutoff value z^{UB} is used as well. Once a feasible solution is found, z^{UB} is updated, after which the Initialization Phase is terminated.

Algorithm 2 getFeasible

```
while MIP($K$) is infeasible do
    - Add the first $w \times m$ variables of the ordered list $L$ to $K$.
    - Solve MIP($K$)
end while
```

3.2 Assessment Phase

In the Assessment Phase the difficulty of the instance is assessed. The difference in difficulty of solving between MIPs can be large. Because of this difference, it is necessary to make distinctions between instances, in order for AKS to perform well. The measure of difficulty that will be used is the time required to solve the last MIP of the Initialization Phase (t_{last}). This is either the MIP(K) with the initial kernel or the last restricted MIP solved in the method getFeasible. An instance can be judged to be *easy*, *normal* or *hard*, making use of the parameters t_{easy} and t_{hard}. If $t_{last} \leq t_{easy}$ the instance is considered easy and if $t_{last} \geq t_{hard}$ the instance is considered hard. If neither of these criteria is met, then the instance is judged to be normal.

Depending on the judged difficulty the kernel from the Initialization Phase is reconsidered. A larger kernel will generally lead to better solutions in the upcoming Improvement Phase, but these solutions are accompanied by longer running times. When AKS is implemented with a time limit, an excessive kernel size might also reduce the quality of the solutions in the Improvement Phase. A trade-off thus has to be made. For easy instances the kernel is enlarged, until the computation time of the sub-problem becomes higher than threshold t_{easy}. This likely leads to better solutions in the Improvement Phase and the fact that the running time is still smaller than t_{easy} is an indication that the sub-problems in the Improvement Phase won't need excessive computation time. For normal instances the kernel is not altered. For hard instances, certain variables in the kernel will be fixed. This is done to prevent excessive computation time of the sub-problems in the upcoming Improvement Phase. The details of updating the kernel when an instance is labeled as "easy" or "hard" are discussed in the following subsections.

3.2.1 Updating the kernel for easy instances

The pseudocode of the updating of the kernel for easy instances is given in Algorithm 3. Let q be a fixed parameter. In each iteration, the first $q \times m$ variables of the ordered list L are put in a set K^+ (and subsequently deleted from the ordered list). Then, MIP($K \cup K^+$) is solved with the current z^{UB} as objective cutoff value and $\sum_{i \in K^+} x_i \geq 1$ as additional constraint. The objective cutoff value specifies that we are only interested in improving solutions. The additional constraint imposes that at least one of the binary variables in K^+ should be equal to one. At the end of each iteration, all variables in K^+ are added to K. If MIP($K \cup K^+$) has a solution, the corresponding objective value becomes the new z^{UB} (note that the objective cutoff value ensures that this solution is at least as good as the solution associated with the previous z^{UB}). As long as the time required to solve MIP($K \cup K^+$) is smaller than t_{easy} and the kernel doesn't contain all binary variables, new iterations are performed.

Algorithm 3 Updating kernel for easy instances

```
while $t_{last} \leq t_{easy}$ AND $K \neq B$ do
  - Let $K^+$ consist of the first $q \times m$ variables of the ordered list $L$.
  - Solve MIP($K \cup K^+$)
  - Update $t_{last}$ to the time that was needed to solve MIP($K \cup K^+$). Add all
  variables of $K^+$ to $K$.
end while
```

3.2.2 Updating the kernel for hard instances

The pseudocode corresponding with the altering of the kernel for hard instances is given in Algorithm 4. The solution of the root node LP relaxation, x^{LP}, is investigated. The binary variables ($j \in B$) for which the value in x_j^{LP} is greater than $1 - \varepsilon$ are permanently fixed to 1. Here, ε is a tolerance parameter that is specified before running the AKS heuristic.

Algorithm 4 Updating kernel for hard instances

```
for $j \in B$ do
  if $x_j^{LP} > 1 - \varepsilon$ then
    Fix value $x_j = 1$
  end if
end for
```

The last step of the Assessment Phase, regardless of the judged difficulty of the instance, is the division of non-kernel variables into buckets. The bucket size is set equal to m, which was the initial kernel size after solving the root node LP relaxation. The first m variables of the ordered list L are added to the first bucket etc.

3.3 Improvement Phase

In the Improvement Phase, AKS will evaluate the sub-problems that each include one bucket A_i. To repeat, NB represents the total number of buckets and $\widetilde{NB}$ is a parameter that indicates the maximum amount of buckets to consider. For the first $\min\{NB, \widetilde{NB}\}$ buckets in $\{A_i\}_{i=1}^{NB}$, the problem MIP($K \cup A_i$) is solved. In a similar way as in the updating of the kernel for easy instances, z^{UB} is set as the objective cutoff value and the constraint $\sum_{j \in A_i} x_j \geq 1$ is added to MIP($K \cup A_i$). Then, if a solution to MIP($K \cup A_i$) is found, the corresponding objective value is the new value of z^{UB}. All binary variables of A_i that had a non-zero value in the best discovered solution to MIP($K \cup A_i$) are added to K.

Considering a situation where a time limit is used, it is possible that for a certain MIP($K \cup A_{i-1}$) a feasible solution is found, but was not proven to be optimal ($i = 2, 3, \ldots, NB$). If this scenario occurs, then a solution to the following MIP($K \cup A_i$) could be an improving solution in which all variables in A_i are zero. However, the constraint that was introduced in the previous paragraph would smother the possibility of finding this improving solution. A change is required. Let x^H denote the solution associated with z^{UB}. The constraint then becomes $\sum_{j \in A_i \cup \{j \in K : x_j^H = 0\}} x_j \geq 1$ instead of $\sum_{j \in A_i} x_j \geq 1$.

3.4 Accommodating general integer variables

For general integer variables, a separate ordered list $\bar{L}$ is kept, which is also created after determining the initial kernel. Likewise, separate buckets $\bar{A}_i$ are created. Contrary to the separate ordered list and the separate buckets, binary variables and integer variables are merged into a single kernel.

The procedure for fixing general integer variables ($j \in I$) when an instance is labeled as "hard" is the following. Let $R(x_j^{LP})$ be the value of x_j^{LP} rounded to the nearest integer. If $R(x_j^{LP}) - \varepsilon \leq x_j^{LP} \leq R(x_j^{LP}) + \varepsilon$, then variable j is permanently fixed to $R(x_j^{LP})$.

As explained in previous subsections, $w \times m$ variables from the ordered list L are constantly added to the kernel in getFeasible and $q \times m$ variables from L are constantly added to the kernel when an instance is easy. Appending general integer variables, these same amounts of variables are taken from the ordered list $\bar{L}$ and inserted in the kernel.

The Improvement Phase of AKS will focus on one bucket of binary variables and one bucket of integer variables simultaneously. Generally, the amount of buckets with binary variables is not equal to the amount of buckets containing general integer variables. If there are no more buckets containing one variable type, only the bucket of the other variable type is taken. For the ease of explanation, I will continue using the notation MIP($K \cup A_i$). From now on, MIP($K \cup A_i$) also contains a bucket of general integer variables ($\bar{A}_i$), when there is at least one bucket of general integer variables remaining.

There is a minor change in the constraint that is added to all MIPs in the Improvement Phase. The original constraint was: $\sum_{j \in A_i} x_j \geq 1$. To include the variables coming from the bucket of general integer variables, this constraint becomes: $\sum_{j \in A_i \cup \bar{A}_i} x_j \geq 1$.

3.5 Extension

The extension that is introduced will accommodate a major modification of the Improvement Phase of AKS. The extension variant of AKS is named *AKS-EXT*. In the extension variant, the parameter $\widetilde{NB}$ is set to five, meaning that only the first five buckets are considered. The variables from the sixth bucket onward are deleted and will not be used anywhere in the extension.

Currently, when solving MIP$(K \cup A_i)$, the variables in the remaining buckets are ignored. In AKS-EXT a variant wherein the variables in the remaining buckets are transformed into continuous variables and then added to MIP$(K \cup A_i)$ is explored. From now on, this modified MIP is indicated by MIP$^*(K \cup A_i)$. Note that a feasible solution to MIP$^*(K \cup A_i)$ is not necessarily feasible to the original problem. In attempt to accomplish a feasible solution to the original problem, the solution to MIP$^*(K \cup A_i)$ is combined with the Local Search heuristic *Relaxation Enforced Neighborhood Search* (RENS) of Berthold (2007). RENS is a relatively simple Local Search heuristic, which is desirable, as the AKS part of the extension is expected to consume considerable computation time. RENS starts from a certain solution and performs a fixation procedure on the variables, in order to decrease the size of the problem. RENS was designed to start from the solution of the LP relaxation, but in AKS-EXT RENS will start from solutions to MIP$^*(K \cup A_i)$. The λ solutions to MIP$^*(K \cup A_i)$ with the lowest objective value are explored. Logically, when the amount of solutions to MIP$^*(K \cup A_i)$ is smaller than λ, all solutions are explored.

An advantageous point of AKS-EXT is that, because of MIP$^*(K \cup A_i)$ partly taking into account the variables in the other buckets, the heuristic can "anticipate" towards the future wherein the variables in the other buckets are added as integer variables. In the current version of AKS, these variables are completely ignored, making anticipation towards the future impossible. The approach of AKS-EXT is described in more detail below.

3.5.1 AKS-EXT approach

In the beginning of each iteration of the Improvement Phase, the sub-problem MIP$^*(K \cup A_i)$ is solved. If no feasible solution is found the iteration is ended immediately. If solving the modified MIP did result in a feasible solution, the iteration proceeds as follows, consistent with the procedure of RENS.

Recall that a total of λ solutions can be supplied to RENS. Consider a specific solution to MIP$^*(K \cup A_i)$. First, all variables in the kernel and bucket A_i are fixed to their value in this solution. By construction, these variables are guaranteed to be integer. Then, the variables in the other buckets, which were added as continuous variables in MIP$^*(K \cup A_i)$, are considered. The binary variables in the other buckets that took an integer value in the solution to MIP$^*(K \cup A_i)$ are fixed to their value. The approach for general integer variables in the other buckets is slightly different. Similar to the binary variables, the general integer variables that had an integer value in the solution to the modified MIP are fixed to their value. For the general integer variables that don't satisfy this property, *rebounding* is applied. Rebounding assures that general integer variables are restricted to two possible values. The fractional value in the solution to MIP$^*(K \cup A_i)$ is rounded down to form a lower bound for the general integer variable. An upper bound for the general integer variable is obtained by rounding up the fractional value. By now, the problem size has likely been reduced. The resulting MIP, containing a combination of fixed variables and non-fixed variables, is solved with z^{UB} as objective cutoff value. This MIP is labeled as MIP_{RENS}, as this is the MIP that results from executing the fixation procedure of RENS. If a feasible solution to MIP_{RENS} is acquired, it is a feasible and improving solution.

The kernel regulation of AKS-EXT is as follows. If MIP$^*(K \cup A_i)$ was solved, the best discovered solution is analyzed. All variables in bucket A_i that had a non-zero value in this best discovered solution are added to the kernel. Furthermore, if a variable in the other buckets (added as a continuous variable) took a value greater or equal to 1-γ in the best discovered solution it is added to the kernel. In the case that one or multiple feasible solutions to MIP_{RENS} are obtained, the kernel regulation goes as follows. All variables that are not yet in the kernel and take a non-zero value in any of the examined feasible solutions to MIP_{RENS} are added to the kernel.

4 Results

The Results section will provide the AKS results and will compare them to the results from the CPLEX solver. Also, the extent to which the replicated AKS results are in line with the results of Guastaroba et al. (2017) is exhibited. Last, the performance of the extension is assessed.

The total running time of all solving methods or heuristics of Guastaroba et al. is very large. They presented a total of 137 instances, nearly all instances were run for at least ten hours. Considering the limited amount of weeks reserved for a thesis, a subset of instances used in the original paper was selected. The authors of the original paper defined three categories: *Lazic*, *MIPLIB-2010-OS* and *MIPLIB-2010-NOS*. The first category consists of instances investigated by Lazic et al. (2010). The second category contains instances from the MIPLIB 2010 library (http://miplib2010.zib.de/miplib2010.php) that have a proven optimal solution. The last category is a set of instances from the MIPLIB 2010 library with an unknown optimal solution. The subset of 25 instances that I composed is a mix of instances from these three categories. Four of the instances in the *Lazic* category were selected, along with eleven instances in the *MIPLIB-2010-OS* category and ten instances belonging to the *MIPLIB-2010-NOS* category. The subset contains instances on which AKS performed well (according to the results of Guastaroba et al. (2017)), as well as instances on which AKS performed poorly.

4.1 AKS - Variants and parameters

In this subsection, multiple variants of AKS are introduced and all parameters are set.

The AKS heuristic was implemented with a time limit of 5 hours. This variant is labeled as *AKS(5h)*. The time allocated to every part of AKS is organized as follows. The solving process of the root node and the LP relaxation are only limited by the total time of five hours. After that, the time allocated to solving MIP(K) has to be defined. One tries to achieve a reasonable allocation time by estimating the amount of sub-problems that will be solved after MIP(K). If the method getFeasible is not called and the instance is labeled as "normal", the number of sub-problems that involve the buckets (named α in the next sentence) can easily be calculated. Then, also counting the sub-problem MIP(K) itself, it seems reasonable to set the time allocated to solving MIP(K) to the time remaining divided by $(\alpha + 1)$. In the case that the method getFeasible is applied, each sub-problem within this method will have an allocated time that is twice as high as that of MIP(K). If the time allocated to a sub-problem in getFeasible exceeds the remaining total time for AKS(5h), the allocated time is set equal to the remaining time. In the procedure of enlarging the kernel for easy instances, each sub-problem is allocated the same amount of time as MIP(K). In the Improvement Phase, each sub-problem is allocated an equal fraction of the remaining time. When a sub-problem doesn't need all of its allocated time, the saved time is equally distributed amongst the sub-problems that have not been processed yet.

An advantageous point of AKS is that it provides a possibility to control the trade-off between solution quality and computation time. The trade-off is modulated by several variants of AKS. A variant of AKS that has a stronger focus on low computation time, compared to the previously introduced AKS(5h), is *AKS-TL*. AKS-TL imposes additional time and node limits. For the restricted MIP associated with the initial kernel (MIP(K)) and each restricted MIP in the Improvement Phase, I limit the time without finding a feasible solution to 700 seconds and the time without improving the incumbent solution to 1200 seconds. For instances labeled as "easy" the time limits regarding finding a feasible solution and improving the incumbent solution are slightly different, 600 seconds and 1500 seconds respectively. The amount of nodes explored before finding a feasible solution is limited to 10,000,000. In a similar way, the amount of nodes explored before improving the incumbent solution is limited to 80,000,000. To clarify, none of these limits are in operation for the root node, the LP relaxation and any restricted MIP solved in getFeasible.

Finally, a variant of AKS that is most tilted towards a low computation time is introduced: This variant is named *AKS-TBL*. AKS-TBL adopts all time limits of AKS-TL. Additionally, the time allocated to solving the root node is limited to 7200 seconds. Furthermore, the time spent on the rest of the heuristic is bounded to 3600 seconds. Last, the maximum amount of buckets considered ($\widetilde{NB}$) is set to five.

Now, the parameters that were introduced in Section 3 are set. The parameter w that is used to regulate the amount of added variables in the method getFeasible is given a value of 0.30. The parameter q, functioning to determine the amount of variables to add for easy instances, is set to 0.35. For AKS(5h) and AKS-TL, the maximum amount of buckets considered in the Improvement Phase ($\widetilde{NB}$) is calibrated such that all buckets are taken into account. For AKS-TBL, as mentioned above, $\widetilde{NB}$ is set to five. The time limit for an instance to be considered easy (t_{easy}) equals 10 seconds. Contrary to t_{easy}, t_{hard} is instance-specific, meaning that it will be determined during the execution of the AKS heuristic. The value of t_{hard} is the time allocated to the last MIP of the Initialization Phase (this is MIP(K) or the last restricted MIP solved in the getFeasible method).

4.2 Comparison of CPLEX and AKS

In this subsection, the performance of AKS in my implementation is assessed. Running the CPLEX solver for five hours (CPLEX(5h)) and running the CPLEX solver for 1 hour (CPLEX(1h)) serve as benchmark methods. To clarify, all results in this subsection are obtained by my implementation of AKS and running the CPLEX solver on my computer.

As in the original paper, the results will be presented in the form of comparison with respect to running the CPLEX solver for 5 hours. Let z denote the best discovered objective value of CPLEX(5h) and z_{ot} denote the best discovered objective value of the other method. Then, z_{dif} is the percentage change of the best discovered objective value belonging to the other method, compared to z. This is calculated by the following formula: $z_{dif} = 100\% \times \frac{z_{ot}-z}{|z|}$. Throughout the rest of this thesis, z_{dif} will also be identified as the gap. The variable t represents the running time in seconds. When a method used all of the available computation time, thus reaching the time limit, the value T.L. is assigned to t. The results are shown in Table 1.

Before commenting on the results of Table 1, it is necessary to point out that the selected subset of instances contains a bias. The testing of the total set of instances used in Guastaroba et al. (2017) never led to a high deterioration of the average solution quality in the AKS variants, when compared with CPLEX(5h). For the subset of instances that I selected, a fairly large deterioration is present in the results of Guastaroba et al. In their results the average deterioration of these 25 instances is 4.10% for AKS(5h), 7.75% for AKS-TL and 8.46% for AKS-TBL. These percentages of deterioration should be considered the standard when analyzing my results.

Table 1 reveals that the average deterioration of the solution quality is 12.77% for AKS(5h). This is notably higher than the standard of 4.10%. This high value is heavily influenced by the instances *markshare2* and *ns1456591*, with deviations of 100.00% and 157.44% respectively. In the original paper, respective deviations of 0.00% and 61.94% were noted for these instances. AKS(5h) managed to outperform CPLEX(5h) on three of the instances. A better or equal solution was achieved for 14 of the 25 instances. For AKS-TL, the average dete-

Table 1: Comparison of AKS and CPLEX

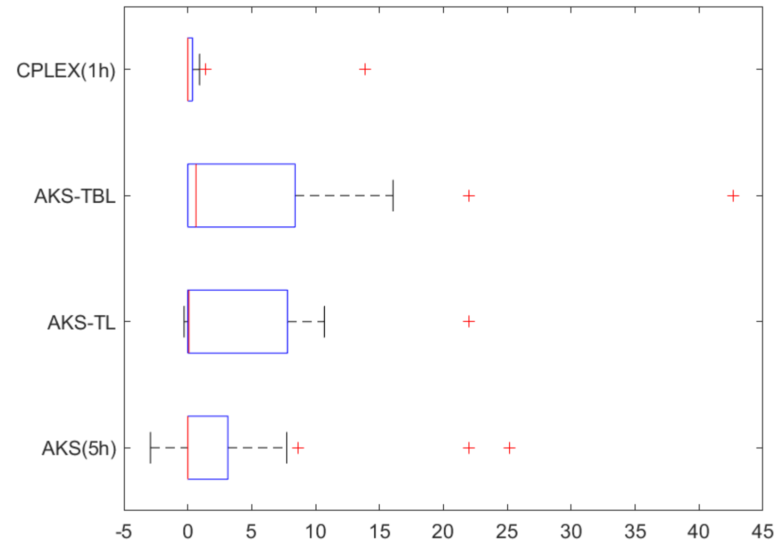
	CPLEX(5h)		CPLEX(1h)		AKS(5h)		AKS-TL		AKS-TBL	
Instances	z	t	$z_{dif}(\%)$	t	$z_{dif}(\%)$	t	$z_{dif}(\%)$	t	$z_{dif}(\%)$	t
danoint	65.67	317	0.00	317	0.00	957	0.00	606	0.00	606
markshare2	5.0	T.L.	60.00	T.L.	100.00	T.L.	60.00	3980	140.00	2170
mkc	-563.85	T.L.	0.00	T.L.	0.00	1206	0.00	1653	2.06	934
swath	478.03	T.L.	0.00	T.L.	21.98	T.L.	21.98	T.L.	21.98	3606
berlin_5_8_0	62.0	1101	0.00	1101	0.00	T.L.	0.00	1684	0.00	1713
d10200	12432.0	T.L.	0.03	T.L.	0.00	T.L.	0.04	4930	0.00	2814
germanrr	$4.70959 \cdot 10^7$	1450	0.00	1450	0.09	4661	0.09	3910	0.09	2905
maxgasflow	$-4.45657 \cdot 10^7$	T.L.	0.00	T.L.	0.00	T.L.	0.00	3079	0.00	1985
opm2-z10-s2	-33826.0	17563	0.61	T.L.	0.00	T.L.	0.06	3535	0.06	2674
probportfolio	16.88	T.L.	0.00	T.L.	1.60	5020	1.60	1918	1.60	1915
queens-30	-39.0	T.L.	0.00	T.L.	0.00	T.L.	0.00	2503	0.00	2491
set3-10	195072.1	T.L.	0.05	T.L.	-2.92	T.L.	7.98	2959	7.98	2864
seymour-disj-10	287.0	T.L.	0.00	T.L.	0.00	T.L.	0.00	2692	0.00	2732
triptim3	13.53	T.L.	0.00	T.L.	0.00	10486	0.00	6708	0.06	1881
tw-myciel4	10.0	T.L.	0.00	T.L.	0.00	T.L.	0.00	T.L.	0.00	3613
dano3mip	685.94	T.L.	0.27	T.L.	0.17	T.L.	2.02	4463	0.95	3606
lectsched-1-obj	81.0	T.L.	0.00	T.L.	0.00	12020	2.47	12025	3.70	2458
momentum3	342681.86	T.L.	13.90	T.L.	7.74	7696	7.74	7710	16.06	2173
n15-3	46379.0	T.L.	0.00	T.L.	-0.92	7685	0.00	T.L.	0.00	5983
ns1456591	1118.04	T.L.	0.00	T.L.	157.44	T.L.	157.44	T.L.	157.44	3608
rmine14	-4267.17	T.L.	82.82	T.L.	-0.31	T.L.	-0.29	10085	2.84	4326
sct32	-17.77	T.L.	0.92	T.L.	8.64	9189	9.56	12998	9.64	3617
shipsched	117160.0	T.L.	1.40	T.L.	25.20	T.L.	10.69	6461	42.68	2555
sing2	$1.73476 \cdot 10^7$	T.L.	0.10	T.L.	0.43	T.L.	0.52	17456	0.64	3652
sing245	$2.56164 \cdot 10^7$	T.L.	0.29	T.L.	0.10	8763	0.12	9845	0.28	3421
Average		**15 937**	**6.42**	**3 283**	**12.77**	**13 507**	**11.28**	**7 728**	**16.32**	**2 812**

rioration of the solution quality is 11.28%. This comprises a difference with the standard of 7.75%. The difference with the standard is predominantly caused by the instance *ns1456591*. Although Guastaroba et al. reported a massive deterioration of 61.94% for this instance, the value of 157.44% in Table 1 clearly surpasses it. AKS-TL improved upon CPLEX(5h) on one instance. The solution provided by AKS-TL was at least as good as the solution provided by CPLEX(5h) for 10 of the 25 instances. Interestingly, the deterioration for AKS-TL is lower than for AKS(5h). This is caused by the relatively large gains in the instances *markshare2* and *shipsched*. These cases should be regarded exceptions, as an improving value for AKS-TL (compared to AKS(5h)) is never accomplished elsewhere. Conversely, AKS(5h) arrives at a better solution than AKS-TL for 10 out of 25 instances. The perception that AKS(5h) is more focused on solution quality than AKS-TL therefore remains valid. For AKS-TBL, the average deterioration is 16.32%, much higher than the expected standard of 8.46%. Again, the instances *markshare2* and *ns1456591* have a dominant share in this dissimilarity with the standard. The values of 140.00% and 157.44% in Table 1 are accompanied by values of 50.00% and 61.94% in the original paper. AKS-TBL was not able to find a better solution than CPLEX(5h) for any of the instances. A solution of equal quality was found for 8 of the 25 instances.

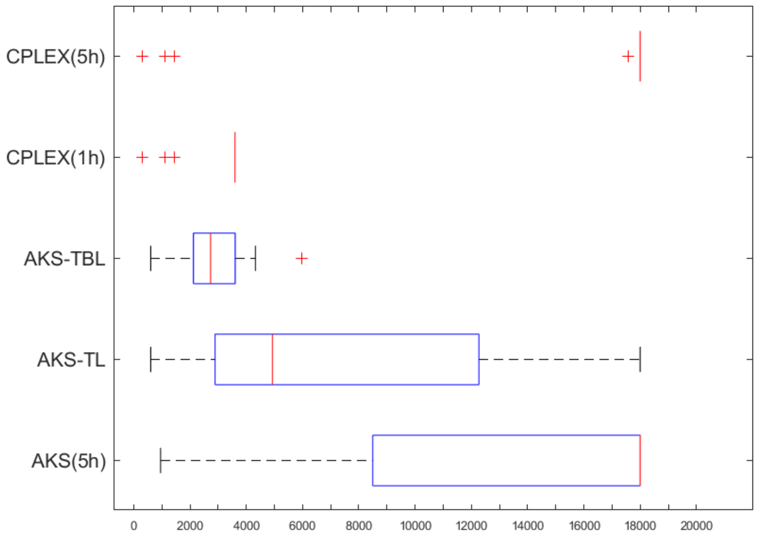
The average computation time for CPLEX(5h) is 15937 seconds. AKS(5h) is able to, on average, reduce the computation time to 13507 seconds. The average time required for AKS-TL is less than 1/2 of the time needed for CPLEX(5h), namely 7728 seconds. AKS-TBL comes with a highly reduced running time, slightly more than 1/3 of the time that was needed for AKS-TL. AKS-TBL also manifested itself as a faster method than CPLEX(1h).

Some results in Table 1 might appear counter-intuitive at first sight. For example, at instance *d10200* AKS-TBL arrives at a better solution than AKS-TL (gaps of 0.00 and 0.04 respectively). This can be explained by the fact that the limit of 3600 seconds caused the instance to be labeled as "hard" in AKS-TBL, leading to the fixation of variables. Due to the effective fixation of variables, AKS-TBL was able to find better solutions in the Improvement Phase. Another example is the running time of AKS-TL and AKS(5h) for instance *sct32*. AKS-TL needed 12998 seconds, whereas AKS(5h) was completed after 9189 seconds. In this case, the superior upper bound of AKS(5h) caused the difference. While AKS-TL struggled to prove infeasibility in the Improvement Phase, especially for the last five sub-problems, AKS(5h) was able to continue much faster. These two examples serve to illustrate that, although the three different variants of AKS can be ordered by overall focus on solution quality or computational time, the framework of AKS justifies the occurrence of exceptions.

The results of the comparison of AKS and CPLEX are presented graphically in the box-and-whisker plots in Figure 1. The left boundary of the blue box represents the first quartile (Q1), whereas the right boundary represents the third quartile (Q3). The red line inside the box indicates the median. The outliers are shown by crosses. A result is considered an outlier if its value is more than $1.5 \times (Q3 - Q1)$ below the first quartile or more than $1.5 \times (Q3 - Q1)$ above the third quartile. The black whisker indicates the non-outlier that is furthest from the box. In the box-and-whisker plots in Figure 1(a), to keep the figure readable, outliers with a gap smaller than -5% or larger than 45% are not reported. Figure 1(a) confirms that outliers are the main cause for the average solution quality of AKS-TL being slightly better than the average solution quality of AKS(5h). The Q3 boundary belonging to the box of AKS(5h) takes a much smaller value than the Q3 boundary belonging to AKS-TL. This indicates that the solutions of AKS(5h) are generally accompanied by less deterioration to CPLEX(5h). Figure 1(b) provides a visualization of the computation times of the methods. One can conclude that AKS(5h) has used all time available (18 000 seconds) for more than half of the instances. The boxes belonging to AKS(5h) and AKS-TL are very wide, indicating that the computation times are widespread. For AKS-TBL, on the other hand, the box is narrow. The



(a) Box-and-whisker plots for the solution quality, $z_{dif}(\%)$. Outliers smaller than -5% or bigger than 45% are not reported.



(b) Box-and-whisker plots for the computation times, t

Figure 1: Box-and-whisker plots

conclusion arises that for AKS-TBL most instances require a computation time that is close to 3600 seconds, which is the maximum time that was reserved after solving the root node.

4.3 Comparison with original paper

In this subsection, the obtained results for the three different AKS variants are compared to the AKS results in the original paper.

There are multiple factors that can cause differences between my results and the results of Guastaroba et al. (2017). All the replicated results were run on a HP Pavilion computer with 16.0 GB RAM and an AMD Ryzen 5 processor. The operating system is Windows 10. AKS was coded in Java, with the default amount of threads. This is not identical to the computer that was used by the authors of the original paper. Next to the computer, the authors noted that they used CPLEX version 12.6.0. I first attempted to use the same CPLEX version, but unfortunately was unable to extract the root node LP relaxation from this version. After consulting my supervisor, it turned out that extracting the root node LP relaxation was simple in CPLEX version 12.10.0. As the root node LP relaxation is a crucial part of the heuristic, CPLEX 12.10.0 was used to obtain my results. The magnitude of the differences in computer and CPLEX version was tested, the results are shown in Table A.1 in the Appendix. Fortunately, on average, the differences appeared to be relatively small. Another difference is that the reduced costs of the LP relaxation were used in my implementation, whereas the reduced costs of the root node LP relaxation were used in the implementation of the authors of the original paper. The consequence is that the order of the ordered list is possibly modified, leading to different sub-problems.

The notation for the comparison of AKS is as follows. Let ω denote the objective value of my results and ω_g denote the objective value found by the authors of the original paper. Then, ω_{dif} is the percentage change of ω with respect to ω_g. More specifically, $\omega_{dif} = 100\% \times \frac{\omega - \omega_g}{|\omega_g|}$. The variable t represents the running time (in seconds) taken to obtain my results. When a method used all of the available computation time, thus reaching the time limit, t is assigned the value T.L. The running time for the results of the authors (in seconds) is represented by t_g. The difference in time, t_{dif}, is calculated by the following formula: $t_{dif} = 2 \times \frac{t}{t+t_g}$. If the running time are approximately equal, t_{dif} will take a value close to 1. This formula ensures that the impact of extreme outliers, which ought to be expected when comparing running times, is limited. The results of the comparison of the AKS variants are reported in Table 2.

There was an additional difference between my results and the results of Guastaroba et al. In their paper, Guastaroba et al. state that the method getFeasible was invoked infrequently. In fact, only three out of all 137 instances that they tested needed the support of the getFeasible method. One of these three instances belongs to the subset that I selected, namely *lectsched-1-obj*. Surprisingly, there were substantially more instances that needed the getFeasible method in my implementation (e.g. 9 out of 25 for AKS-TBL). This suggests

Table 2: AKS - Comparison to original paper

	AKS(5h)				AKS-TL				AKS-TBL			
Instances	ω	ω_{dif}(%)	t	t_{dif}	ω	ω_{dif}(%)	t	t_{dif}	ω	ω_{dif}(%)	t	t_{dif}
danoint	65.67	0.00	957	1.38	65.67	0.00	606	1.17	65.67	0.00	606	1.17
markshare2	10.0	150.0	T.L.	1.00	8.0	33.33	3980	1.22	12.0	100.00	2170	0.94
mkc	-563.85	0.00	1206	0.13	-563.85	0.00	1633	1.02	-552.25	2.05	934	0.75
swath	583.11	21.05	T.L.	1.58	583.11	21.05	T.L.	1.63	583.11	21.05	3606	1.09
berlin_5_8_0	62.0	0.00	TL	1.00	62.0	0.00	1684	0.88	62.0	0.00	1713	0.89
d10200	12432.0	0.00	T.L.	1.00	12437.0	0.03	4930	1.13	12432.0	-0.03	2814	1.04
germanrr	$4.71363 \cdot 10^7$	-0.04	4661	0.41	$4.71363 \cdot 10^7$	-0.04	3910	0.88	$4.71363 \cdot 10^7$	-0.04	2905	0.95
maxgasflow	$-4.45658 \cdot 10^7$	0.00	T.L.	1.00	$-4.45658 \cdot 10^7$	0.00	3079	0.99	$-4.45645 \cdot 10^7$	0.00	1985	0.78
opm2-z10-s2	-33826.0	-0.07	T.L.	1.00	-33806	-0.01	3535	0.53	-33806.0	-8.27	2674	0.81
probportfolio	17.15	0.00	5020	1.79	17.15	0.00	1918	1.52	17.15	0.00	1915	1.53
queens-30	-39.0	0.00	T.L.	1.00	-39.0	0.00	2503	1.11	-39.0	0.00	2491	1.11
set 3-10	189378.27	-4.11	T.L.	1.00	210638.84	2.18	2959	1.33	210638.84	2.18	2864	1.33
seymour-disj-10	287.0	0.00	T.L.	1.00	287.0	0.00	2692	1.01	287.0	0.00	2732	1.02
triptim3	13.53	0.00	10486	0.74	13.53	0.00	6708	0.78	13.54	0.06	1881	0.64
tw-myciel4	10.0	0.00	T.L.	1.00	10.0	0.00	T.L.	1.93	10.0	0.00	3613	1.71
dano3mip	687.07	0.09	T.L.	1.00	699.79	-0.65	4463	1.27	692.43	-1.69	3111	1.07
lectsched-1-obj	81.0	10.96	12020	0.80	83.0	13.70	12025	0.80	84.0	10.53	2458	0.81
momentum3	369209.68	11.07	7696	0.60	369209.68	-7.93	7710	1.43	397720.69	-3.68	2173	0.75
n15-3	45954.0	1.29	7685	0.60	46379.0	2.23	T.L.	1.60	46379.0	2.23	5983	1.13
ns1456591	2878.31	58.98	T.L.	1.00	2878.31	58.98	T.L.	1.82	2878.31	58.98	3608	1.33
rmine14	-4280.33	-0.06	T.L.	1.00	-4279.72	-4.15	10085	1.16	-4145.98	-0.18	4326	0.94
sct32	-16.23	-51.18	9189	0.68	-16.07	-48.66	12998	1.05	-16.05	-56.44	3617	1.00
shipsched	146707.0	9.77	T.L.	1.00	129705.0	-12.61	6461	1.11	167181.0	12.64	2555	0.83
sing2	$1.74217 \cdot 10^7$	0.45	T.L.	1.00	$1.74379 \cdot 10^7$	0.52	17456	1.27	$1.74581 \cdot 10^7$	0.45	3652	1.00
sing245	$2.56428 \cdot 10^7$	-0.03	8763	1.21	$2.56428 \cdot 10^7$	-0.03	9845	1.35	$2.56884 \cdot 10^7$	-0.03	3421	1.13
Average		**8.32**		**0.957**		**2.32**		**1.200**		**5.59**		**1.030**

that there was a difference in the construction of the initial kernel (the method getFeasible is only called when MIP(K) is infeasible). Note that the initial kernel is constructed by adding all variables with a strictly positive value in the final linear program solved at the root node. It is most likely that the final linear program solved at the root node was not always completely consistent with the final linear program of the authors of the original paper. Although this difference presumably affects all instances, the impact was particularly large for two instances. This concerns the instances *ns1456591* and *swath*. The initial kernel of these instances contained a tiny fraction of all binary and general integer variables. Subsequently, when the method getFeasible was invoked, very few variables were added to the kernel. The consequence was that all three AKS variants were not able to exit the getFeasible method before reaching the total time limit. This is reflected by the relatively poor results on both instances.

The results in Table 2 indicate that the solution quality of my results is lower than the solution quality of the authors of the original paper. For AKS(5h), the gap ($\omega_{dif}(\%)$) takes an average value of 8.32%. The average gap for AKS-TL equals 2.32% and for AKS-TBL it is 5.59%. These percentages are mainly caused by a bad performance on a few instances. As mentioned, all three AKS variants performed poorly on the instances *ns1456591* and *swath*. Furthermore, the results on instance *markshare2* were clearly inferior to the results of the authors of the original paper. On this particular instance, the solutions found by the CPLEX solver (see Table A.1 in the Appendix) were also worse than the results of the authors. It is conceivable that the new CPLEX version contains different techniques that turn out to be disadvantageous for *markshare2*. All three AKS variants improved massively on the instance *sct32*, in line with the major improvement in the CPLEX solver for this instance (also shown in Table A.1 in the Appendix). In total, my results achieved a better value for AKS(5h) for 6 out of 25 instances and a worse value for 9 instances. The gap is between -1% and 1% for 16 out of 25 instances. A better value for AKS-TL was achieved 8 out of 25 times. A worse value also occurred 8 times. The gap for AKS-TL is between -1% and 1% for 15 of the 25 instances. For AKS-TBL, my implementation resulted in a better value for 8 out of 25 instances. A worse value was obtained for 10 instances. For AKS-TBL, the gap is between -1% and 1% for 13 out of 25 instances.

Table 2 exhibits that the average value of t_{dif} in AKS(5h) is 0.957. This is slightly lower than 1 and thus suggests that AKS(5h) is slightly faster in my implementation. For AKS-TL, the average value of t_{dif} is 1.200. This is relatively far from 1. The increase is consistent with the fact that the getFeasible method was invoked frequently, as the time and node limits for AKS-TL are not in operation within getFeasible. Last, the average value of t_{dif} corresponding to AKS-TBL is 1.030. Presumably, the negative effect of the getFeasible method on the running time was largely diminished by the time limit of 3600 seconds imposed on AKS-TBL.

One can observe from Table 2 that the AKS variants, in general, tend to perform analogously when compared to the results of the authors of the original paper. For example, all AKS variants were accompanied by a large deterioration of the solution quality for the instances *markshare2* and *lectsched-1-obj*. A considerable improvement was achieved on instance *sct32*. There are exceptions regarding this pattern, such as the remarkably good solution of AKS-TL on instance *shipsched* and the noticeably poor solution of AKS(5h) on instance *momentum3*.

4.4 AKS-EXT

In this subsection, the results of the extension are shown and discussed. The framework of the extension (AKS-EXT) was already explained in Section 3.5.

Two additional parameters were introduced for AKS-EXT. After solving the modified MIP, with the variables from the other buckets that are added as continuous variables, a maximum of λ solutions are passed on to the RENS procedure. Preliminary experiments indicated that performing the RENS procedure on a particular solution is often accompanied with a short computation time. Therefore, it seems advantageous to check a lot of solutions, and λ is set to 50. The other parameter that was introduced is γ. This parameter determines when the variables in the other buckets, that were added as continuous variables, will be added to the kernel. I chose γ equal to 10^{-5}.

Next, the allocation of time for the AKS-EXT variant will be discussed. The time reserved for solving the root node is 7200 seconds. The time spent on the rest of the heuristic is also limited to 7200 seconds. The division of time for the solving of MIP(K) with the initial kernel, any restricted MIP in the method getFeasible or the MIPs in the procedure for "easy" instances is in line with the three other AKS variants. Note that there are two steps in the Improvement Phase of AKS-EXT; the solving of the modified MIP with the variables of the other buckets added as continuous variables and the MIP_{RENS} that results after applying the RENS procedure. For both steps, the time and node limits regarding the search for a feasible or an improving solution are identical to those of AKS-TL and AKS-TBL. For example, for normal and hard instances, the time limit for improving the incumbent solution is set to 1200 seconds. Recall that $\widetilde{NB}$ is set to five in AKS-EXT, meaning that MIP$^*(K \cup A_i)$ will be solved with a maximum of five different buckets A_i. These different MIPs are allocated an equal amount of time. The time reserved for each MIP solved after the RENS procedure is half the time that was allocated to the MIP$^*(K \cup A_i)$ that was solved prior to it.

It is good to note that I tested a few parameters within the CPLEX solver that could positively influence the results of AKS-EXT. In the current version of AKS-EXT, the λ solutions with the lowest objective value are passed to the RENS procedure. It is imaginable that this is not the most efficient regulation. When these λ solutions are similar to a large extent, it is likely that many solutions violate the same constraint(s). Therefore, two parameters that encourage solution diversity were examined. The parameter SolnPoolCapacity, defining the maximum amount of solutions that are retained, was set to both 10 and 50. Then, the parameter SolnPoolReplace was set to 2, in order to keep the most diverse solutions in the solution pool. In preliminary experiments, any combination of these two additional parameters appeared to influence the results in a negative way. For this reason, I decided to omit them.

Like the results for the other AKS variants, the performance of AKS-EXT is expressed in comparison to CPLEX(5h). Let z denote the objective value of the best solution found by CPLEX(5h). Then, z_{ot} represents the best discovered objective value of AKS-EXT. The percentage change (z_{dif}) is defined as follows: $z_{dif} = 100\% \times \frac{z_{ot}-z}{|z|}$. Last, t stands for the computational time that was needed for AKS-EXT. Because the main difference of AKS-EXT is the modification to the Improvement Phase, it is useful to mention the instances for which the Improvement Phase resulted in an ameliorating solution. In Table 3, these instances are indicated by an asterisk ($*$) after their name.

The results on the solution quality and computation time of AKS(5h), AKS-TL and AKS-TBL were already presented in Table 1. A brief overview of the comparison of AKS-EXT to the other AKS variants will be provided in this subsection. I introduce a measure to compare the solution quality. *Comp(5h)* represents the comparison on solution quality to AKS(5h), *Comp(TL)* represents the comparison to AKS-TL and *Comp(TBL)* marks the comparison to AKS-TBL. If the z_{dif} belonging to AKS-EXT is at least 2% lower than the z_{dif} of one of the other methods, the notation "++" is given in the corresponding comparison. When the gap of AKS-EXT is lower, but less than 2% lower, the notation is "+". No difference in z_{dif}(rounded to two decimal places) is indicated by "$\pm$". The procedure for a higher z_{dif} of AKS-EXT is alike. If the gap is at least 2% higher the notation "$--$" is given, whereas other cases with a higher z_{dif} are marked by "$-$". The results are shown in Table 3.

Table 3 reports that the average gap of AKS-EXT, with respect to CPLEX(5h), is 12.74%. Recall that a detailed comparison of the solution quality and computation time among the three AKS variants and AKS-EXT can be obtained by inspecting the results in Table 1 and Table 3 simultaneously. Looking back at Table 1, the average gap of AKS(5h) was 12.77%. The average gaps of AKS-TL and AKS-TBL were 11.28% and 16.32% respectively. On average, AKS-EXT thus delivers better solutions than both AKS(5h) and AKS-TBL, and worse solutions than AKS-TL. The "plus-minus" notation provides more detail in the comparison. Interestingly, AKS-EXT outperformed all three other AKS variants on the instances *probportfolio* and *shipsched*. AKS-EXT was able to improve the solution of AKS(5h) for 2 of the 25 instances. A large improvement was made for the instance *shipsched*, heavily impacting the average gap comparison between AKS-EXT and AKS(5h). The solution of AKS(5h) turned out to be superior for 12 instances. AKS-EXT supplied a better solution than AKS-TL for 3 of the 25 instances. Conversely, the solution of AKS-TL was superior for 10 instances. AKS-EXT found a better solution than AKS-TBL for 7 out of 25 instances. Three of these seven improvements were marked with a "++". For 4 instances, the solution of AKS-EXT was inferior to the solution of AKS-TBL. None of these four instances had major declining solutions, as they were all marked by a single "-". Overall, AKS-EXT appears to produce lower quality solutions than the variants AKS(5h) and AKS-TL. AKS-EXT is successful in obtaining higher quality solutions than the variant AKS-TBL.

Furthermore, Table 3 exhibits that the average computation time required for AKS-EXT was 4522 seconds. Examining Table 1 again I note that, on average, 13507 seconds were needed for AKS(5h), 7728 seconds were needed for AKS-TL and AKS-TBL required 2812 seconds. AKS-EXT can thus be placed between AKS-TL and AKS-TBL in terms of computation time.

Last, it is noteworthy that AKS-EXT was able to find at least one solution in the Improvement Phase for 13 of the 25 instances. For the vast majority of the remaining 12 instances, the other three AKS variants also didn't obtain a solution in the Improvement Phase. Hence, this should not be regarded as a weakness of AKS-EXT. Examples of possible causes for not exploiting the Improvement Phase are the discovering of the optimal solution in the earlier

Table 3: Extension results.

	AKS-EXT				
Instances	$z_{dif}(\%)$	t	$Comp(5h)$	$Comp(TL)$	$Comp(TBL)$
danoint	0.00	605	$\pm$	$\pm$	$\pm$
markshare2	100.00	2768	$\pm$	$--$	$++$
mkc ($*$)	2.06	4669	$--$	$--$	$\pm$
swath	21.98	7205	$\pm$	$\pm$	$\pm$
berlin_5_8_0 ($*$)	0.00	1732	$\pm$	$\pm$	$\pm$
d10200 ($*$)	0.00	3893	$\pm$	$+$	$\pm$
germanrr ($*$)	0.45	3446	$-$	$-$	$-$
maxgasflow ($*$)	0.00	4850	$\pm$	$\pm$	$\pm$
opm2-z10-s2	0.06	3556	$-$	$\pm$	$\pm$
probportfolio ($*$)	0.00	1859	$+$	$+$	$+$
queens-30 ($*$)	0.00	2490	$\pm$	$\pm$	$\pm$
set 3-10	7.98	2947	$--$	$\pm$	$\pm$
seymour-disj-10	0.00	1952	$\pm$	$\pm$	$\pm$
triptim3 ($*$)	0.00	4672	$\pm$	$\pm$	$+$
tw-myciel4	0.00	7212	$\pm$	$\pm$	$\pm$
dano3mip	2.10	4372	$-$	$-$	$-$
lectsched-1-obj	3.70	4850	$--$	$-$	$\pm$
momentum3	9.93	3668	$--$	$--$	$++$
n15-3	0.00	9761	$-$	$\pm$	$\pm$
ns1456591	157.44	7208	$\pm$	$\pm$	$\pm$
rmine14 ($*$)	3.56	6845	$--$	$--$	$-$
sct32 ($*$)	11.26	6331	$--$	$-$	$-$
shipsched ($*$)	-2.82	5836	$++$	$++$	$++$
sing2 ($*$)	0.62	6565	$-$	$-$	$+$
sing245 ($*$)	0.21	3754	$-$	$-$	$+$
Average	**12.74**	**4522**			

($*$) AKS-EXT found a solution in the Improvement Phase

phases or never reaching the Improvement Phase due to getting stuck in the getFeasible method.

5 Conclusion

In this thesis, I replicated the results of Guastaroba et al. (2017). I implemented the three different variants of the Adaptive Kernel Search heuristic, named *AKS(5h)*, *AKS-TL* and *AKS-TBL*. The variants provide options in the trade-off between a high solution quality and a low computation time. AKS(5h) is mainly focused on solution quality. AKS-TL takes a step towards a lower computation time, inevitably accompanied by a lower solution quality. AKS-TBL is the variant that is most tilted towards a low computational time and therefore also expected to have the lowest solution quality. In order to assess the performance of the three AKS variants, a set of 25 instances was tested.

In Guastaroba et al. (2017) the average gap of the solution quality, when compared to running the CPLEX solver with a time limit of five hours, is reported. The subset of 25 instances that I selected contained a bias. For all three AKS variants, the average gap that the authors of the original paper reported on these 25 instances is much higher than the average gap that the authors reported over their full set of 137 instances, therefore constituting a bias. The average gap of these 25 instances in the results of Guastaroba et al. (2017) was 4.10% for AKS(5h), 7.75% for AKS-TL and 8.46% for AKS-TBL. These percentages should be seen as the standard. In my implementation of AKS, the average gap was higher than the standard for all three variants. I reported an average gap of 12.77% for AKS(5h), 11.28% for AKS-TL and 16.32% for AKS-TBL. For all variants the differences to the standard appear to be substantial at first sight, but could largely be explained by a small number of outliers.

The fact that AKS-TL, on average, arrives at less deteriorating solutions compared to AKS(5h) can be seen as surprising. However, as I have illustrated, this unexpected difference is caused by a few outliers. In general, the claim that AKS(5h) lays more focus on solution quality than AKS-TL remains valid.

The AKS variants all go along with shorter average computation times than running the CPLEX solver with a time limit of five hours (CPLEX(5h)). The decrease of computation time was not very large for AKS(5h). AKS-TL, on the other hand, required less than half of the computation time that was needed for CPLEX(5h). The difference for AKS-TBL was the most remarkable, as it required only slightly more than $\frac{1}{6}$ of the computation time that CPLEX(5h) used.

Even when taking the impact of the outliers into account, it is fair to state that my implementation of AKS delivers solutions of a slightly lower quality than the CPLEX solver with a time limit. However, compared to the CPLEX solver with a time limit of five hours, AKS-TL and AKS-TBL go along with highly reduced computation times. Remarkably, AKS-TBL required less time than the CPLEX solver with a time limit of one hour. Concluding, there is no reason to contest the overall conclusion of the authors of the original paper, that AKS is a valuable addition to the toolbox of researchers and commercial practitioners who desire to obtain high-quality solutions to MIPs in a reasonable amount of time.

For the comparison of my results to the results of the authors of the original paper, the average gap was 8.32% for AKS(5h). For AKS-TL, the average gap was 2.32%. The value of the average gap was 5.59% for AKS-TBL. These dissimilarities appear to be substantial at first sight. Inspecting the results closely, the conclusion that the high dissimilarities are mainly caused by a few outliers emerges. For each of the three AKS variants, a gap in the interval [-1.00%,1.00%] is established for more than half of the instances. I attempted to find factors that constitute significant differences between my implementation and the implementation of the authors of the original paper. First of all, it was noted that different computers and different versions of CPLEX were utilized. Fortunately, an experiment indicated that the effect of these two factors is minor. Next, a possible difference in the ordered list of non-kernel variables was noted. The consequence of such a difference is that the sub-problems of the AKS heuristic are altered. Although this can severely impact the results of specific instances, it seems unlikely that it can account for the considerably high average gaps in the AKS variants. The most dominant difference between my implementation and the implementation of the authors was discovered during the execution of the AKS heuristic. The method getFeasible, required when the initial kernel can't form a feasible solution, was employed much more frequently in my results. The most likely explanation is that the final linear program solved in the root node, which is used for the construction of the initial kernel, was not always identical to the final linear program of the authors. This presumably affected the results on many instances, and it proved to have exceptionally negative effects for some instances. All in all, the differences between my implementation and the implementation of the authors of the original paper can directly explain some outliers. Of course, there remain a few instances for which a large deterioration or improvement can't wholly be attributed to a specific difference in the implementation. However, it is still well possible that the origin of these large deteriorations or improvements lie in the noted differences between the implementations. I conclude that the differences that I've noted can explain the dissimilarities between my results and the results of Guastaroba et al. (2017) reasonably well.

As an extension, I introduced a variant of AKS that imposes a major adjustment to the Improvement Phase of the AKS heuristic. Similar to the three predefined AKS variants, the solution quality of the extension variant (AKS-EXT) was measured in comparison with CPLEX(5h). For two instances, AKS-EXT managed to outperform all three other AKS variants, in terms of solution quality. The average gap of the extension variant was lower than the average gap belonging to AKS(5h) and AKS-TBL. AKS-EXT was associated with a higher average gap than AKS-TL. Examining the results more closely, I observed that a small amount of instances disproportionately influenced the average gap comparison. Overall, the solutions of AKS-EXT ought to be seen as superior to the solutions of AKS-TBL, and inferior to the solutions of AKS(5h) and AKS-TL. The extension variant requires significantly less computation time than AKS(5h). Judging by computation time, AKS-EXT can be placed in between AKS-TL and AKS-TBL, being in closer proximity to AKS-TBL.

The subset containing 25 instances can already be considered relatively small. For AKS-EXT, the Improvement Phase led to an improving solution for only 13 instances. On the vast majority of the other 12 instances, AKS(5h), AKS-TL and AKS-TBL likewise didn't find any solutions in the Improvement Phase. Accordingly, the noted performance of AKS-EXT compared to the other three AKS variants is based on a small amount of observations. Therefore, although the results of AKS-EXT seem promising, it has to be stated that further research is required in order to provide a representative image of its performance.

References

Achterberg, T., & Berthold, T. (2007). Improving the feasibility pump. *Discrete Optimization*, *4*(1), 77–86.

Adamo, T., Ghiani, G., Guerreiro, E., & Manni, E. (2020). A learn-and-construct framework for general mixed-integer programming problems. *International Transactions in Operational Research*, *27*(1), 9–25.

Angelelli, E., Mansini, R., & Speranza, M. (2010). Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers & Operations Research*, *37*(11), 2017–2026.

Baena, D., & Castro, J. (2011). Using the analytic center in the feasibility pump. *Operations Research Letters*, *39*(5), 310–317.

Balas, E., Ceria, S., Dawande, M., Margot, F., & Pataki, G. (2001). Octane: A new heuristic for pure 0-1 programs. *Operations Research*, *49*(2), 207–225.

Berthold, T. (2007). Rens-relaxation enduced neighborhood search. *ZIB-Report 07-28*.

Fischetti, M., Glover, F., & Lodi, A. (2005). The feasibility pump. *Mathematical Programming*, *104*, 91–104.

Fischetti, M., & Lodi, A. (2011). Heuristics in mixed integer programming. *Wiley Encyclopedia of Operations Research and Management Science*.

Fischetti, M., & Salvagnin, D. (2009). Feasibility pump 2.0. *Mathematical Programming Computation*, *1*(2), 201–222.

Gamrath, G., Berthold, T., Heinz, S., & Winkler, M. (2019). Structure-driven fix-and-propagate heuristics for mixed integer programming. *Mathematical Programming Computation*, *11*, 675–702.

Ghiani, G., Laporte, G., & Manni, E. (2015). Model-based automatic neighborhood design by unsupervised learning. *Computers & Operations Research*, *54*, 108–116.

Guastaroba, C., Savelsbergh, M., & Speranza, M. (2017). Adaptive kernel search: A heuristic for solving mixed integer linear programs. *European Journal of Operational Research*, *263*(3), 789–804.

Lazic, J., Hanafi, S., Mladenovic, N., & Urosevic, D. (2010). Variable neighbourhood decomposition search for 0–1 mixed integer programs. *Computers & Operations Research*, *37*(6), 1055–1067.

A Comparison with original paper - CPLEX

I use the following notation for the CPLEX comparison in Table A.1. Let ψ denote the objective value of my results and ψ_g denote the objective value found by the authors of the original paper. Then, ψ_{dif} is the percentage change of ψ with regard to ψ_g. More specifically, $\psi_{dif} = 100 \times \frac{\psi - \psi_g}{|\psi_g|}$. Furthermore, t stands for the running time in my results, and t_g represents the running time in the results of the authors (both measured in seconds). The difference in time, t_{dif}, is calculated by the following formula: $t_{dif} = 2 \times \frac{t}{t+t_g}$. A value of T.L. indicates that the time limit was reached for the instance.

Table A.1: CPLEX - Comparison with original paper

	CPLEX(5h)				CPLEX(1h)			
Instances	ψ	ψ_{dif}(%)	t	t_{dif}	ψ	ψ_{dif}(%)	t	t_{dif}
danoint	65.67	0.00	317	1.15	65.67	0.00	T.L.	1.15
markshare2	5.0	25.00	T.L.	1.00	8.0	33.33	T.L.	1.00
mkc	-563.85	0.00	T.L.	1.00	-563.85	0.00	T.L.	1.00
swath	478.03	2.27	T.L.	1.00	478.03	2.27	T.L.	1.00
berlin_5_8_0	62.0	0.00	1101	0.12	62.0	0.00	1101	0.47
d10200	12432.0	-0.03	T.L.	1.00	12436.0	1.51	T.L.	1.00
germanrr	$4.70959 \cdot 10^7$	0.00	1450	0.15	$4.70959 \cdot 10^7$	0.00	1450	0.57
maxgasflow	$-4.45657 \cdot 10^7$	-0.01	T.L.	1.00	$-4.45648 \cdot 10^7$	-0.01	T.L.	1.00
opm2-z10-s2	-33826.0	-1.47	17563	0.99	-33621.0	-0.85	T.L.	1.00
probportfolio	16.88	0.85	T.L.	1.00	16.88	0.00	T.L.	1.00
queens-30	-39.0	0.00	T.L.	1.00	-39.0	0.00	T.L.	1.00
set 3-10	195072.07	-0.36	T.L.	1.00	195172.82	-2.16	T.L.	1.00
seymour-disj-10	287.0	0.00	T.L.	1.00	287.0	0.00	T.L.	1.00
triptim3	13.53	0.00	T.L.	1.00	13.53	0.00	T.L.	1.00
tw-myciel4	10.0	0.00	T.L.	1.00	10.0	0.00	T.L.	1.00
dano3mip	685.94	0.90	T.L.	1.00	687.81	-1.02	T.L.	1.00
lectsched-1-obj	81.0	6.58	T.L.	1.00	81.0	0.00	T.L.	1.00
momentum3	342681.86	-10.25	T.L.	1.00	390314.36	-13.60	T.L.	1.00
n15-3	46379.0	0.18	T.L.	1.00	46379.0	0.18	T.L.	1.00
ns1456591	1118.04	0.00	T.L.	1.00	1118.04	0.00	T.L.	1.00
rmine14	-4267.17	0.21	T.L.	1.00	-733.05	33.17	T.L.	1.00
sct32	-17.77	-3.29	T.L.	1.00	-17.60	-5.23	T.L.	1.00
shipsched	117176.0	1.73	T.L.	1.00	118811.0	3.15	T.L.	1.00
sing2	$1.73476 \cdot 10^7$	-0.07	T.L.	1.00	$1.73655 \cdot 10^7$	-0.15	T.L.	1.00
sing245	$2.56164 \cdot 10^7$	-0.28	T.L.	1.00	$2.56895 \cdot 10^7$	-0.15	T.L.	1.00
Average		**0.88**		**0.936**		**2.02**		**0.968**