Cutting-Plane Separation Strategies for Semidefinite Programming Models to Solve Single-Row Facility Layout Problems

by

Ginger Yen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Management Sciences

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The single-row facility layout problem (SRFLP) is concerned with finding the optimal linear placement of n departments with different lengths in a straight line. It is typically achieved by minimizing the cost associated with the interactions between the departments. The semidefinite programming (SDP) relaxation model that incorporates cutting planes proposed recently by Anjos, Kennings, and Vannelli (AKV) was considered a breakthrough in the field. This thesis presents a new SDP model AKV' and compares the two relaxations. The AKV' is largely based on the previous model, but it reduces the number of linear constraints from $O(n^3)$ to $O(n^2)$. Therefore, it reduces the computing time at the expense of a slightly weaker lower bound. However, AKV' is observed to pay off as the instance size increases. By examining the gap for both the AKV and AKV' relaxations, we notice that both relaxations generate very small gaps at the root node, which demonstrates the effectiveness of the relaxations.

Six different strategies are presented to separate the cutting planes for the medium-sized SRFLP. In combination with the two SDP relaxations, we compare the six strategies using three instances of different characteristics. An overall best strategy is deduced from the computational results, but the best choice of relaxations and the best number of cuts added at each iteration changes depending on the characteristics of the instances. Two new cutting plane strategies are proposed for large instances. This allows the solution to optimality of new instances with 36 departments, which is higher than previously published results in literature. We also briefly point out how the computing time can vary greatly between different sets of data of the same size due to the characteristics of the department lengths.

Acknowledgements

First, I would like to thank my supervisor, Professor Miguel Anjos, whose patience, encouragement, guidance, and enthusiasm truly helped me moving forward. He is the best supervisor I could ever ask for. Also, many thanks to my friends from KWCAC, who supported me through prayers and faithful friendship. Above all, I would like to thank my beloved family, whose love and care are what I hold most dearly. Finally, I would like to give thanks to God, who led me to the Department of Management Sciences, gave me the purpose to work hard, and continues guiding and watching over me.

Dedication

This thesis is dedicated to my beloved mother, Chun-Feng Yang, for her endless patience, selflessness, love, and care.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The facility layout problem (FLP) determines the most efficient arrangement of n individual departments within a facility. It is a well-studied combinatorial optimization problem that can be employed in many different applications. Many expensive applications contain numerous important functional objects to be arranged on a very restricted area, and achieving the most efficient arrangement leads to cost saving. Some classical examples of the facility layout problem applications include integrated circuit design, control panel layout design, wiring design, building layout, urban planning [10, 30], and multiple-floor facilities [9]. The single-row facility layout problem (SRFLP) is a special case of the general layout problem where the n departments are to be arranged on a straight line. The SRFLP also has many practical applications, such as the arrangement of departments on one side of a corridor in supermarkets, hospitals, or offices [36], the assignment of disk cylinders to files [33], the assignment of airplanes to gates in an airport terminal [39], and the arrangement of machines along a straight path travelled by an automated guided vehicle (AGV) in flexible manufacturing systems [20].

The problem instance consists of the length ℓ_i of each department i and an $n \times n$ matrix F, where F_{ij} represents the travel intensity between department i and j. The objective of the problem is to arrange the departments in order to minimize the weighted sum of the distances between all department pairs, which is often expressed in terms of material handling cost [30]. Some of the common constraints in a facility layout problem include limiting the departments so that they are contained within the allowable space boundary. Another common constraint is to ensure that the departments do not overlap [30]. Depending on the solution approaches and models, the constraints may be expressed differently. When the lengths of all the departments are the same, the SRFLP becomes the linear ordering (or linear arrangement) problem, see [16] and [26] for more details. The linear ordering problem is also a special case of the well-studied quadratic assignment problem (QAP), see [11] for more details.

With 50 years of history since the first publication on the QAP by Koopmans and Beckman in 1957 [23], substantial research effort has been put in to search for better ways to solve the FLP. Many new solution approaches, models, and solution algorithms have been introduced. However, it is still widely recognized that the facility layout problem is a very difficult problem class. For instance, when the QAP was first proposed, it was seen as unsolvable for practical problems. In 1986 the largest QAP problem that had been solved optimally only had 15 departments [25]. By 1996, the number had only been improved slightly to 18 departments, when solving on a routine basis [30]. By 2002, a QAP with 30 departments was solved, but vast amount of computation was required, which is unrealistic on a routine basis [7]. Even now, QAP instances with $n > 30$ cannot be solved within reasonable time [27]. Other than QAP, it is widely recognized that SRFLP is strongly NP-hard [1]. Needless to say, many heuristics have been proposed for the SRFLP, such as [13], [14], [17], [19], [20], [21], [24], [31], and [37]. However, this research thesis focuses on the exact solution approach using the semidefinite programming (SDP) formulation with the help of different cutting plane strategies.

The contribution of this thesis is to empirically examine the new matrix-based SDP formulation of SRFLP, which was proposed by Anjos and Yen [6]. In addition, the work of optimization using SDP and cutting planes by Anjos and Vannelli [5] is improved upon by constructing and evaluating various cutting plane strategies that allow the process to become dynamic. In Chapter 2, background on the SRFLP is presented. In Chapter 3, the new SDP model is presented and discussed in detail. An empirical comparison between the two models is also given. In Chapter 4, six cutting plane strategies are evaluated and compared. This comparison is further enhanced by incorporating the analysis of the two SDP models. Furthermore, a best model-strategy combination will be presented to be used to solve large instances that were unsolved in the past. These results are presented in Chapter 5. Finally, conclusions and possible directions for future research are discussed in Chapter 6.

Chapter 2

Background

2.1 Optimization Solution Scheme

This section will clarify and define some of the terms that will be used extensively throughout this thesis. Figure 2.1 displays a roadmap of how an optimization problem is typically solved.

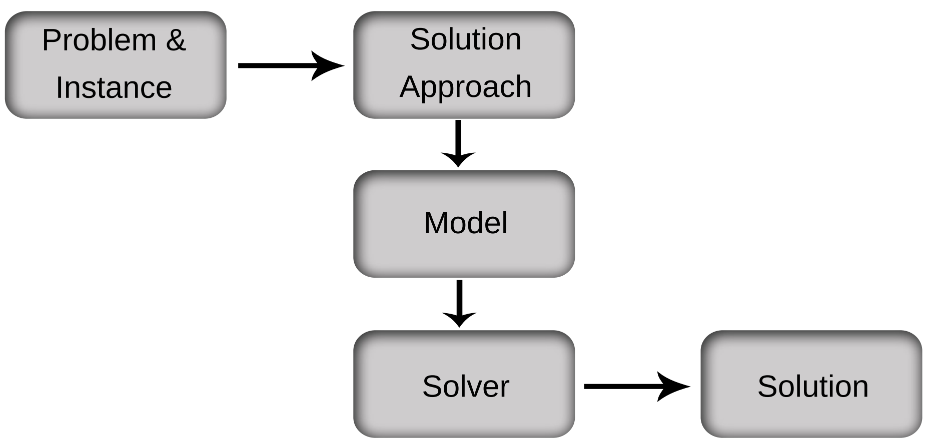

Figure 2.1: Optimization solution roadmap

2.1.1 Problem and Instance

The SRFLP can arise in many practical problems. An interesting example is the problem of assigning incoming aircrafts to airport gates [39]. Suppose that within a short time frame, several flights that carry many connecting passengers arrive from and depart to various cities. The *problem* for the airline management is to minimize the inconvenience of the connecting passengers, which is measured by the distance travelled in between the connecting flights. Therefore, each flight has an associated interaction value with each other, which is determined by the number of connecting passengers. Two flights with significant numbers of connecting passengers should be placed as close to each other as possible. This problem can be expressed with an interaction flow matrix that specifies the level of interaction between each flight. In this case, the distance between each aircraft is fixed by the distance between gates, regardless of the size of the aircraft. Therefore, the length vector can be assumed as a vector of all ones. This problem is thus a linear ordering problem, which is a special case of the QAP.

The *instance* of five aircrafts ($n = 5$) can be expressed this way [39]:

$$F = \begin{bmatrix} 0 & 1 & 5 & 5 & 7 \\ 1 & 0 & 8 & 3 & 4 \\ 5 & 8 & 0 & 1 & 5 \\ 5 & 3 & 1 & 0 & 7 \\ 7 & 4 & 5 & 7 & 0 \end{bmatrix} \quad \text{and} \quad \ell = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Figure 2.2: The problem of airplane-to-gate assignment

2.1.2 Solution Approaches

When one encounters a *problem* with the collected data as the *instance*, under certain assumptions, one must first decide on a suitable *solution approach*. Based on that approach, one can then build a *model* with an objective function and constraints. To reach global optimality for the SRFLP, there are the branch-and-bound approach [36], the dynamic programming approach [22], [33], the mixed-integer linear programming (MILP) approach [16], [34], [28], [18], [1], and the SDP approach [4]. Most recently, Amaral and Letchford [3] have also used the polyhedral approach to formulate the SRFLP.

On the other hand, if only a local optimum is needed, the SRFLP can be solved using a nonlinear programming (NLP) approach [21], the metaheuristic approach, or simply solved by the interchange approach, such as 2-Opt. 2-Opt is a heuristic formulation that consists of sequence of pairwise exchange of departments. If the exchange results in improvement, the two departments are swapped. Otherwise, they stay in the same spot and the algorithm goes on to find the next exchange pair. The process continues until no more changes can be made. It is a computationally inexpensive algorithm that is used in several parts of the cutting plane algorithm in this thesis to reduce the search space more rapidly. The input variable of 2-Opt is any permutation π and the function will return an improved (or the same) permutation. The Matlab code of 2-Opt can be found in Appendix A. As for metaheuristics, there are many examples in literature, including the simulated annealing method [35], [19], and the greedy heuristic [24].

2.1.3 Models

Through each *solution approach*, the *problem* can be formulated mathematically in different *models*. For example, the MILP approach was used by many researchers to build models that solve SRFLP [16], [34], [28], [18], [1]. Although with the same approach, different researchers can express the problem differently. For instance, Heragu [18] proposed the model LMIP1 using the MILP approach. LMIP1, though a different model, is similar to another MILP model by Love and Wong [28]. The main difference between the two models is in the calculation of inter-departmental distance, where Heragu uses centroids of departments i and j, while Love and Wong uses the endpoint location of each department to calculate the distance. However, both models are known to provide poor global lower bounds while requiring long computational time. Other models that adapt the MILP approach were proposed by Grötschel et al. [16], and Reinelt [34]. In 2006, Amaral presented another model using the MILP approach, which has shown an improvement from all of the earlier MILP-based models [1]. Although Amaral's model uses the same number of zero-one variables, it presents a smaller number of continuous variables than the preceding models in literature. It is also shown to improve the lower bound and the computation time in comparison to Love and Wang's model [1]. However, Anjos and Vannelli [5] pointed out that these MILP-based models, although they guarantee global optimality, also require high computational time and memory requirements. Most recently in 2008, Amaral proposed a new lower bound in [2], which is yet to be investigated in detail.

Heragu and Kusiak [21] presented ABSMODEL1 for the problem using NLP approach. In this model, the absolute value of the distance between the centroids of each department is used, which makes the model non-linear. Therefore, the selection of the initial point is very important when solving a SRFLP using the ABSMODEL1.

Anjos, Kennings, and Vannelli [4] presented a model, AKV, using the SDP solution approach. In [4], a heuristic method was also presented to convert a relaxed solution to a feasible solution. AKV presented the first non-trivial global lower bound for the SRFLP in the published literature [4]. More recently, a new version of this matrix-based model, AKV', shows some promising improvement [6], which will be discussed in Chapter 3 .

2.1.4 Solvers

Finally, each *model* may be solved by different *solvers*. For instance, the NLP-based models can be solved by BARON, CONOPT, MINOS, SNOPT, and PATH [29]. There are also many solvers available for the SDP-based models, such as CSDP, SeDuMi, and SDPT3. While there are many solvers for the linear programming (LP) approach, such as CPLEX, SDP solvers such as SDPT3 and SeDuMi can also solve linear problems. As indicated by the list of solvers for the various formulations, one can observe that some solvers are solution approach-specific, while others can be used to solve models from a number of different solution approaches.

2.1.5 Solution

After an iterative process of computation, a *solution* can be achieved. The solution indicates the most efficient arrangement of the different departments and also the objective value, which is often expressed as material handling cost. For many practical problems, only a near-optimal solution can be obtained, since most practical problems are relatively large in size. Fortunately, for practical purposes, high precision is normally not required. Therefore, the analyst can resort to heuristic methods in a case like this. It is thus an important judgment for an analyst to assess the required level of accuracy and precision before finalizing what solution approach, model, and solver to employ.

2.2 Review of Recent Mathematical Programming SRFLP Models

2.2.1 ABSMODEL1

Heragu and Kusiak proposed ABSMODEL1 in [21], where they set the decision variable x_i to represent the location of department i, measured from the reference point zero to the centroid of department i. There are a total of n departments, where f_{ij} denotes the interaction frequency cost between department i and j, and ℓ_i represents the length of the horizontal side of department i. Both f_{ij} and ℓ_i are input parameters from the problem instances.

$$
\begin{aligned}
&\min \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij} |x_i - x_j| \\
&\text{s.t.} \\
&\qquad |x_i - x_j| \geq 0.5(\ell_i + \ell_j) \quad \text{for all pairs } 1 \leq i < j \leq n
\end{aligned}
$$

With the employment of absolute terms to denote centre-to-centre distance, we are not concerned whether department i is to the left or to the right of department j. Furthermore, the constraint ensures no overlap between any two departments. Since the constraints of ABSMODEL1 are not convex, solving a SRFLP using this model is a heuristic (local optimum) search technique.

2.2.2 LMIP1

LMIP1 is a MILP-based model, which is similar to another MILP model proposed by Love and Wong [28]. LMIP1 is discussed in detail in this thesis because, instead of measuring interdepartmental distance from the endpoint of the department like in [28], it measures distance from the centroid of each department, which is consistent with all the other models introduced in this thesis.

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij}(x_{ij}^{+} - x_{ij}^{-}) \\
\text{s.t.} \quad & \\
& x_i - x_j + M\alpha_{ij} \geq 0.5(\ell_i + \ell_j) \quad \text{for all pairs } 1 \leq i < j \leq n \\
& x_j - x_i + M(1 - \alpha_{ij}) \geq 0.5(\ell_i + \ell_j) \\
& x_i - x_j = x_{ij}^{+} - x_{ij}^{-} \\
& x_{ij}^{+} \geq 0 \text{ and } x_{ij}^{-} \geq 0 \\
& \alpha_{ij} \in \{0, 1\} \\
& x_i > 0 \quad \text{for all } i = 1, \ldots n
\end{aligned}
$$

The transformation of ABSMODEL1 to LMIP1 is shown in [18], where the absolute term is replaced by $x_{ij}^{+} + x_{ij}^{-}$. The parameter M is a sufficiently large positive number. Similar to ABSMODEL1, the decision variable x_i represents the location of department i, measured from the reference point zero to the centroid of department i. The two new variables x_{ij}^{+} and x_{ij}^{-} represent the distance between department i and j, and they are defined as below:

$$
x_{ij}^{+} := \begin{cases} x_i - x_j, & \text{if } (x_i - x_j) > 0, \\ 0, & \text{otherwise,} \end{cases}
$$

$$
x_{ij}^{-} := \begin{cases} x_j - x_i, & \text{if } (x_i - x_j) \leq 0, \\ 0, & \text{otherwise.} \end{cases}
$$

One interesting fact about the SRFLP is its natural symmetry, in which any solution can be expressed by two opposite permutations. The binary variable α_{ij} serves to break the natural symmetry of the department arrangement by forcing one of the first two constraints trivial. This means that department i will be either to the left or to the right of department j. The binary variable α_{ij} is defined as below:

$$
\alpha_{ij} := \begin{cases} 1, & \text{if } x_i < x_j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}
$$

Other than the new decision variables listed above, the meaning of the parameters f_{ij}, ℓ_i, and n are the same as in ABSMODEL1. Similarly, the objective function $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij}(x_i^+ - x_j^+)$ also seeks to minimize the total weighted sum of centre-to-centre distance between department i and j. The first two constraints ensure no overlap.

2.2.3 Amaral's Model

Amaral proposed the following MILP model in [1]. The main difference between this model and the earlier MILP-based models is in the new decision variable d_{ij}, which is defined as the distance between the centroids of department i and j. Another decision variable in this model is the binary variable α_{ij}, which is also defined as in Equation (2.1) in LMIP1. Similar to LMIP1, the objective function $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij} d_{ij}$ also seeks to minimize the total weighted sum of centre-to-centre distance between department i and j.

Let x_i be the location (or coordinate) of department i. It can be expressed as:

$$x_i = \frac{\ell_i}{2} + \sum_{k=1, k \neq i}^{n} \ell_k \; \alpha_{ki}. \tag{2.2}$$

The new decision variable d_{ij} is defined as

$$d_{ij} = \max\{(x_i - x_j), (x_j - x_i)\} \quad \text{for } 1 \leq i < j \leq n,$$

which can be rewritten as

$$d_{ij} := \begin{cases} x_j - x_i, & \text{if } x_j > x_i, \\ x_i - x_j, & \text{otherwise}, \end{cases} \quad \text{for } 1 \leq i < j \leq n,$$

$$\text{or} \qquad d_{ij} \geq x_i - x_j, \;\; d_{ij} \geq x_j - x_i, \quad \text{for } 1 \leq i < j \leq n. \tag{2.3}$$

By substituting Equation (2.2) into the new expression of d_{ij} in Equation (2.3), we get

$$\begin{aligned} x_i - x_j \;\; &= \sum_{k=1, k \neq i}^{n} \ell_k \alpha_{ki} - \sum_{k=1, k \neq j}^{n} \ell_k \alpha_{kj} + (\ell_i - \ell_j)/2 \\ &= \sum_{k<i} \ell_k \alpha_{ki} + \sum_{k>i} \ell_k (1 - \alpha_{ik}) - \sum_{k<j} \ell_k \alpha_{kj} - \sum_{k>j} \ell_k (1 - \alpha_{jk}) + (\ell_i - \ell_j)/2. \end{aligned}$$

Therefore, for the case of $d_{ij} \geq x_i - x_j$,

$$d_{ij} \geq \sum_{k<i} \ell_k \alpha_{ki} + \sum_{k>i} \ell_k (1 - \alpha_{ik}) - \sum_{k<j} \ell_k \alpha_{kj} - \sum_{k>j} \ell_k (1 - \alpha_{jk}) + (\ell_i - \ell_j)/2.$$

Finally, Amaral's model for the SRFLP is given by:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij} d_{ij} \\
\text{s.t.} \quad & \\
& d_{ij} \geq \sum_{k<i} \ell_k \alpha_{ki} + \sum_{k>i} \ell_k (1 - \alpha_{ik}) - \sum_{k<j} \ell_k \alpha_{kj} - \sum_{k>j} \ell_k (1 - \alpha_{jk}) + (\ell_i - \ell_j)/2, \\
& d_{ij} \geq - \sum_{k<i} \ell_k \alpha_{ki} - \sum_{k>i} \ell_k (1 - \alpha_{ik}) + \sum_{k<j} \ell_k \alpha_{kj} + \sum_{k>j} \ell_k (1 - \alpha_{jk}) + (\ell_j - \ell_i)/2 \\
& \qquad \text{for } 1 \leq i < j \leq n, \\
& \alpha_{ij} + \alpha_{jk} - \alpha_{ik} \leq 1 \qquad \text{for } 1 \leq i < j < k \leq n, \\
& -\alpha_{ij} - \alpha_{jk} + \alpha_{ik} \leq 0 \qquad \text{for } 1 \leq i < j < k \leq n, \\
& \alpha_{ij} \in \{0, 1\} \qquad \text{for } 1 \leq i < j \leq n, \\
& d_{ij} \geq (\ell_i + \ell_j)/2 \qquad \text{for } 1 \leq i < j \leq n.
\end{aligned}$$

The triangle inequality constraint set helps to make the definition of left and right consistent. The last constraint sets the minimal distance between each department pair to ensure no overlap.

2.2.4 The AKV Model

The AKV model proposed by Anjos, Kenning, and Vannelli in [4] has a similar structure with the SDP model for the max-cut problems by Goemans and Williamson [15]. Both models set the diagonal elements of the positive semidefinite variable X to one. Furthermore, the first constraint in AKV is similar to the triangular constraints in the max-cut model. When disregarding the rank constraint, AKV becomes the relaxation model that can be used for lower bound computation. The

AKV model is given by,

$$
\begin{aligned}
\min \quad & K - \sum_{i<j} \frac{f_{ij}}{2} \left[\sum_{k<i} \ell_k X_{ki,kj} - \sum_{i<k<j} \ell_k X_{ik,kj} + \sum_{k>j} \ell_k X_{ik,jk} \right] \\
\text{s.t.} \quad & \\
& X_{ij,jk} - X_{ij,ik} - X_{ik,jk} = -1 \text{ for all triplets } i<j<k \\
& \operatorname{diag}(X) = e \\
& \operatorname{rank}(X) = 1 \\
& X \succeq 0
\end{aligned}
\tag{2.4}
$$

where $K := \left(\sum_{i<j} \frac{f_{ij}}{2} \right) \left(\sum_{k=1}^{n} \ell_k \right)$, $\operatorname{diag}(X)$ denotes a vector formed by the diagonal elements of X, e denotes the vector of all ones, and $X \succeq 0$ signifies that matrix X is positive semidefinite. The derivation of the constant K will be discussed later.

The entire AKV model is built upon the binary variables R, which are given by,

$$
R_{ij} := \begin{cases} 1, & \text{if facility } i \text{ is to the right of facility } j, \\ -1, & \text{if facility } i \text{ is to the left of facility } j. \end{cases}
$$

It is clear that one of the two possibilities must hold for every feasible arrangement of the departments and that $R_{ij} = -R_{ji}$. The purpose of variable R_{ij} is similar to the α_{ij} in Equation (2.1) for LMIP1 and Amaral's model. The minor difference between the two binary variables is that $R_{ij} \in \{-1, 1\}$, while $\alpha_{ij} \in \{0, 1\}$. Also, the left-right position of facility i is defined differently. By listing all R_{ij} with $i < j$, a vector v can be formed with length $\binom{n}{2}$, where n is the number of departments. Using v, the rank-one matrix X is constructed as $X = vv^T$, such that element $X_{ij,kl} = R_{ij}R_{kl}$. Therefore, the diagonal elements of X are 1 since $X_{ij,ij} = R_{ij}^2 = 1$. Also it should be noted that the matrix X is of size $\binom{n}{2} \times \binom{n}{2}$.

To accurately model the problem, we must make sure that the relationship of left and right of each department triplet is maintained. Therefore, the following condition is required to hold:

$$
\text{if } R_{ij} = R_{jk}, \text{ then } R_{ij} = R_{ik}.
$$

This means that if i is to the right of j, and j is to the right of k, then i must be right of k. This expression can be rewritten as $(R_{ij} + R_{jk})(R_{ij} - R_{ik}) = 0$. After expansion, we get $R_{ij}R_{jk} - R_{ij}R_{ik} - R_{ik}R_{jk} = -1$. Finally, when expressed in terms of variable X, we obtain the following constraint:

$$X_{ij,jk} - X_{ij,ik} - X_{ik,jk} = -1.$$

The following steps illustrate how any given feasible set of R_{ij} can be interpreted and mapped to the more intuitive format of a permutation π. A permutation lays out the department numbers under a given arrangement. These steps are also the backbone of the AKV Heuristic, which will be discussed in Section 2.2.5.

1. For each department $k = 1 \ldots n$, sum up R_{kj} by

   $$P_k = \sum_{j \neq k} R_{kj}$$

   which can be interpreted as how far to the right department k should be positioned. All the P_k values belong to the set $\{-(n-1), -(n-3), \ldots (n-3), (n-1)\}$.

2. Map the numbers to the set $\{1, 2, \ldots n\}$ by substituting into the formula $p_k = (P_k + n + 1)/2$. But p_k still can be interpreted as how much to the right department k should be placed.

3. Sort the p_k to achieve the permutation π.

It should be noted that if every R_{ij} variable is replaced by its negative, the arrangement of the departments remains the same, and it creates no change to the model. This is how the AKV model can implicitly take into account of the natural symmetry of the SRFLP.

The objective function is to minimize the total weighted sum of centre-to-centre distance between all department pairs, which is originally expressed as:

$$\sum_{i<j} f_{ij} \left[\frac{1}{2}\ell_i + D_\pi(i,j) + \frac{1}{2}\ell_j \right]. \tag{2.5}$$

$D_\pi(i,j)$ signifies the sum of the lengths of the departments between departments i and j in a given arrangement π. It can be rewritten as:

$$D_\pi(i,j) = \sum_{k \neq i,j} \ell_k \left(\frac{1 - R_{ki}R_{kj}}{2} \right). \tag{2.6}$$

This formula is valid because index k is between i and j *iff* $R_{ki}R_{kj} = -1$. Therefore, only the lengths of the departments that are positioned between the given department pair i and j are summed up. After substituting Equation (2.6) into Equation (2.5), the objective function can be rewritten this way:

$$\begin{aligned}
& \sum_{i<j} f_{ij} \left[\tfrac{\ell_i+\ell_j}{2} + \sum_{k\neq i,j} \ell_k \left(\tfrac{1-R_{ki}R_{kj}}{2} \right) \right] \\
= \; & \sum_{i<j} f_{ij} \left[\left(\sum_{k=1}^{n} \tfrac{\ell_k}{2} \right) - \sum_{k\neq i,j} \ell_k \tfrac{R_{ki}R_{kj}}{2} \right] \\
= \; & \left(\sum_{i<j} \tfrac{f_{ij}}{2} \right) \left(\sum_{k=1}^{n} \ell_k \right) - \sum_{i<j} \tfrac{f_{ij}}{2} \left[\sum_{k<i} \ell_k R_{ki} R_{kj} - \sum_{i<k<j} \ell_k R_{ik} R_{kj} + \sum_{k>j} \ell_k R_{ik} R_{jk} \right]
\end{aligned}$$

where $\left(\sum_{i<j} \frac{f_{ij}}{2} \right) \left(\sum_{k=1}^{n} \ell_k \right)$ is the constant K in (2.4).

2.2.5 AKV Heuristic

Goemans and Williamson [15] applied a randomized rounding heuristic for the max-cut problems to derive a feasible solution from the lower bound solution. By using a different methodology, the AKV Heuristic also extracts a feasible permutation π from the optimal solution X^* of the relaxation. The concept of mapping from R_{ij} to π is briefly explained in the previous section. This section will give a more detailed explanation to the implementation of the translation from the optimal solution X^* of relaxed AKV or AKV' to a feasible permutation π.

1. Calculate R_{ij} by using X^* from the lower bound calculation:

   By the definition of matrix X, we know that the first row of X is

   $$R_{12} \cdot v^T = (R_{12}R_{12} \quad R_{12}R_{13} \quad R_{12}R_{23} \quad R_{12}R_{14} \ldots R_{12}R_{(n-1)n}).$$

   Thus by setting $R_{12} = 1$, all of R_{ij} can be calculated by using the first row of X. Note that since the lower bound X^* is from the relaxation, which means it is very likely not rank-one, the elements X_{ij} are not $\in \{-1, 1\}$. Therefore R_{ij} can be any value between -1 and $+1$.

2. Translate R_{ij} to permutation π by first calculating P_k for each department k by summing R_{kj} as followed:

   $$P_k = \sum_{j\neq k} R_{kj}.$$

P_k can be seen as the weight of how far to the right department k should be positioned.

3. Sort the departments by the weight value P_k in descending order, since R_{12} is assumed to be 1 and we prefer to see the facilities in an order such that $i < j$.

In the Matlab implementation, X^* is $\binom{n}{2} \times \binom{n}{2}$, so there are $\binom{n}{2}$ sets of possible feasible solutions: one for each row. All the rows of the X^* are checked through and compared to ensure the best-known feasible solution is obtained. The heuristic algorithm 2-Opt is also incorporated after obtaining a permutation to improve it further. In the experiments for this thesis, high-quality feasible solution is often observed at root node. Please see Appendix B for the Matlab code of the AKV Heuristic.

Chapter 3

Comparison of the SDP Models

In this chapter, AKV', a new matrix-based SDP model is presented. Later in the chapter, a lower bound comparison between the original AKV and the new AKV' relaxation model is made to study the tradeoff.

3.1 The AKV' Model

The AKV' model is first introduced in [6]. This SDP-formulated model is largely based on (2.4), but it reduces the number of linear constraints from $O(n^3)$ to $O(n^2)$. Other than the reduction in the number of linear constraints, everything else in the new model remains the same as in AKV.

The AKV' model is presented in the following way:

$$
\begin{array}{ll}
\min & K - \sum\limits_{i<j} \frac{f_{ij}}{2} \left[\sum\limits_{k<i} \ell_k X_{ki,kj} - \sum\limits_{i<k<j} \ell_k X_{ik,kj} + \sum\limits_{k>j} \ell_k X_{ik,jk} \right] \\
\text{s.t.} & \\
& \sum\limits_{k \neq i,j,k=1}^{n} X_{ij,jk} - \sum\limits_{k \neq i,j,k=1}^{n} X_{ij,ik} - \sum\limits_{k \neq i,j,k=1}^{n} X_{ik,jk} = -(n-2) \text{ for all pairs } i<j \\
& \operatorname{diag}(X) = e \\
& \operatorname{rank}(X) = 1 \\
& X \succeq 0
\end{array}
\tag{3.1}
$$

Removing the rank-one constraint also results in an SDP relaxation. It should be noted that, although the number of constraints is now reduced to $O(n^2)$, which leads to savings in computation time, the quality of the solution also deteriorates slightly. The tradeoff is studied in Section 3.3.

3.2 Model Equivalency

The AKV' relaxation is essentially relaxed from the AKV relaxation. It is therefore interesting to verify whether the AKV' model (3.1) is equivalent to AKV (2.4) with the rank-1 constraint. Namely, we want to find out whether the feasible sets of the two models are equal.

Theorem 1 *The feasible sets of (2.4) and (3.1) are identical.*

Proof: First we will show that X feasible for (3.1) is also feasible for (2.4). Rewrite the first constraint of (3.1) as

$$\sum_{k \neq i,j,k=1}^{n} (X_{ij,jk} - X_{ij,ik} - X_{ik,jk}) = -(n-2) \text{ for all pairs } i < j$$

Suppose X is feasible for (3.1). Then the constraints $\operatorname{diag}(X) = e$ and $\operatorname{rank}(X) = 1$ together imply that $X_{ij,k\ell} = \pm 1$ for all entries of X. Furthermore, $X \succeq 0$ implies that $X_{ij,jk} - X_{ij,ik} - X_{ik,jk} \geq -1$ for all distinct i, j, k. Hence,

$$\sum_{k \neq i,j,k=1}^{n} (X_{ij,jk} - X_{ij,ik} - X_{ik,jk}) \geq -(n-2).$$

Therefore, it is clear that each term $X_{ij,jk} - X_{ij,ik} - X_{ik,jk}$ must equal -1. This means X is feasible for (2.4).

It is then straightforward to show that X feasible for (2.4) is also feasible for (3.1). By summing all the k terms from 1 to n for all pairs $i < j$, the first constraint in (2.4) becomes the first constraint in (3.1). ■

3.3 Comparison of Lower Bound Computation

New test instances were generated by using the connectivity data from some of the well-known Nugent QAP Problems [32]. The facility lengths were randomly generated, with the exception of all the instances with their names ending in "1". These instances have all the department lengths equal to unity.

The computation results in this section was generated on a Sun Fire V890 8*1.2GHz with 64Gb of RAM. The SDP problems were solved using the interior-point solver CSDP (version 5.0) of [8] in conjunction with the ATLAS library of routines [41].

First, we compare the two SDP relaxations for problems with 25 to 42 facilities. This comparison aims to provide a sense of how much the lower bounds are weakened by the reduction in the number of constraints in AKV'. The results are reported in Table 3.1.

The gap is calculated as the percentage difference between the lower bound and the best feasible solution by the AKV heuristic. Roughly speaking, the smaller the gap, the shorter the computation time one would expect to eventually reach global optimality. By examining the gap for both the AKV and AKV' relaxations, we notice that both relaxations generate very small gaps at the root node, which demonstrates the effectiveness of the relaxations. Furthermore, it is evident that while the CPU times are significantly smaller for the new AKV', the resulting gaps still remain small, mostly between 3% to 7% (with only 1 exception out of 20 test instances). The savings in computation time are especially significant for larger instances. In particular, for the instances of size 42, the CPU time for the original AKV relaxation is about 2.5 times greater than the new AKV' relaxation, while the average gap only decreases to 3.16% from 5.11%. Moreover, if we compare the two lower bounds directly, the relative gap between the two lower bounds is very small with an average value of 1.64%.

Instance	# of fac.	AKV from (2.4) Lower bound	CPU time (sec)	Best layout by AKV heuristic	Gap	AKV' from (3.1) Lower bound	CPU time (sec)	Best layout by AKV heuristic	Gap	Gap between lower bounds
SRFLP-nug25-1	25	4515.0	44	4622.0	2.37%	4463.5	39	4626.0	3.64%	1.15%
SRFLP-nug25-2	25	36355.5	44	37641.5	3.54%	35960.5	42	37346.5	3.86%	1.10%
SRFLP-nug25-3	25	23691.0	43	24537.0	3.57%	23398.0	41	24609.0	5.18%	1.25%
SRFLP-nug25-4	25	47330.0	43	48887.5	3.29%	46798.5	40	48811.5	4.30%	1.14%
SRFLP-nug25-5	25	15304.5	44	15767.0	3.02%	15148.0	42	15783.0	4.19%	1.03%
SRFLP-nug30-1	30	8061.0	192	8305.0	3.03%	7975.5	128	8310.0	4.19%	1.07%
SRFLP-nug30-2	30	21188.5	195	21663.5	2.24%	20921.5	128	21672.5	3.59%	1.28%
SRFLP-nug30-3	30	44518.5	194	45712.0	2.68%	43986.0	133	45703.0	3.90%	1.21%
SRFLP-nug30-4	30	55947.5	194	56922.5	1.74%	55181.0	136	57060.5	3.41%	1.39%
SRFLP-nug30-5	30	113072.0	186	115776.0	2.39%	111828.5	129	115986.0	3.72%	1.11%
SRFLP-ste36-1	36	10087.5	884	10301.0	2.12%	9851.0	471	10328.0	4.84%	2.40%
SRFLP-ste36-2	36	175387.0	843	181910.0	3.72%	170759.5	435	182649.0	6.96%	2.71%
SRFLP-ste36-3	36	98739.0	809	102179.5	3.48%	96090.0	436	104041.5	8.28%	2.76%
SRFLP-ste36-4	36	94650.5	850	96080.5	1.51%	91103.0	439	96854.5	6.31%	3.89%
SRFLP-ste36-5	36	89533.0	852	91893.5	2.64%	87688.0	441	92563.5	5.56%	2.10%
SRFLP-sko42-1	42	24807.0	3032	25724.0	3.70%	24517.0	1160	25779.0	5.15%	1.18%
SRFLP-sko42-2	42	210785.0	3056	217296.5	3.09%	207357.0	1174	218117.5	5.19%	1.65%
SRFLP-sko42-3	42	169944.5	3206	173854.5	2.30%	167783.5	1164	174694.5	4.12%	1.29%
SRFLP-sko42-4	42	133429.5	3030	138829.0	4.05%	131536.0	1115	139630.0	6.15%	1.44%
SRFLP-sko42-5	42	242925.5	3075	249327.5	2.64%	238669.5	1172	250501.5	4.96%	1.78%
Average Gap					2.86%				4.88%	1.65%

Table 3.1: Comparison of the two SDP relaxations

Chapter 4

Cutting Plane Separation Strategies

One typical way to tighten the semidefinite relaxation of an integer optimization problem is to add inequalities as cutting planes, such as the triangle inequalities. For more information on different classes of inequalities, see [12]. Anjos and Vannelli [5] use a simple scheme in combination with the AKV model to detect and add violated triangle inequalities to solve SRFLPs with up to 30 departments to global optimality. In this thesis, we improve upon the work in [5] by a thorough investigation of more sophisticated cutting-plane strategies. The objective is to compare the various strategies in combination with the AKV and AKV' relaxations and come up with the best overall combination.

The triangle inequalities to be considered are valid for the integer feasible points. There are four types, each with $\binom{\binom{n}{2}}{3}$ inequalities:

$$\begin{aligned} X_{p1,p2} + X_{p1,p3} + X_{p2,p3} &\geq -1 \\ X_{p1,p2} - X_{p1,p3} - X_{p2,p3} &\geq -1 \\ -X_{p1,p2} - X_{p1,p3} + X_{p2,p3} &\geq -1 \\ -X_{p1,p2} + X_{p1,p3} - X_{p2,p3} &\geq -1 \end{aligned} \tag{4.1}$$

where $p1$, $p2$, $p3$ are three distinct pairs. Therefore, there are a total of $4\binom{\binom{n}{2}}{3}$ additional inequality constraints, which means $O(n^6)$, that can be added to the relaxation. Obviously, these are too many to include simultaneously for a practical problem with large n. Consequently, an algorithm is required to filter and select a number of violated inequalities that can create the greatest impact for lower bound improvement and the attainment of the global optimum as quickly as possible. The general approach for such an algorithm in this thesis begins by solving the AKV or AKV' relaxation, then adding some violated inequalities, re-optimizing, and repeating until no more violations can be found. Due to the strength of the SDP relaxations, the algorithm never runs out of cuts before global optimality is attained. In essence, the six strategies considered differ mainly in the ways that violated inequalities are detected. Also, in Strategies 4, 5, and 6, a scheme that removes the inequality constraints with positive slack at the relaxed optimum solution after each re-optimization step is incorporated. This feature helps to keep the size of the SDP small. In Strategies 5 and 6, an algorithm that performs re-search for the violated inequalities is included when the total number of violations found is less than half of the anticipated number set by the user. A more detailed description of each strategy is presented in the following sections.

4.1 The Six Strategies

Because there are too many possible constraints to be added all at once, an algorithm that ranks and selects the cuts is developed to collaborate with the AKV and AKV' relaxations. The process was made dynamic by using the parameter `vioRHS`. It is the dynamic condition that determines whether an inequality is considered to be violated. When expressed mathematically, it is the right-hand-side value for the triangle inequalities in Equation (4.2):

$$\begin{aligned} X_{p1,p2} + X_{p1,p3} + X_{p2,p3} + 1 &\geq vioRHS \\ X_{p1,p2} - X_{p1,p3} - X_{p2,p3} + 1 &\geq vioRHS \\ -X_{p1,p2} - X_{p1,p3} + X_{p2,p3} + 1 &\geq vioRHS \\ -X_{p1,p2} + X_{p1,p3} - X_{p2,p3} + 1 &\geq vioRHS \end{aligned} \tag{4.2}$$

Therefore, the closer `vioRHS` is to zero, the closer the above inequalities (4.2) are to the actual triangle inequalities (4.1. Consequently, more violations can be found as the inequalities in the algorithm becomes closer to the actual inequalities. The more violations that are detected, the longer it takes for the algorithm to sort and generate the cuts. However, if the `vioRHS` value is set too high, the algorithm cannot find cuts, and it will conclude erroneously that the gap has closed and global optimality has been reached. In other words, by manipulating `vioRHS`, we can control the number of cuts found, and thereby control the computation time of finding the cuts. This algorithm manipulates the `vioRHS` parameter dynamically based on the state of the optimization process, so that shorter computation times can be achieved while ensuring the accuracy of the conclusion.

Another important parameter that affects the computation effort is `numcut`, which represents the number of cuts to add to each sub-problem. While `vioRHS` significantly affects the computation time by controlling the number of possible cuts that can be found, `numcut` affects the computation time by regulating the number of cuts that can be added out of all the found cuts. The higher the `numcut`, the more rapidly the size of the SDP problem grows, and hence the faster the growth in optimization time. Although this trend may sound unfavourable, a high value of `numcut` can also lead to a reduction in the number of iterations required. Therefore, a lot of observation and fine-tuning is necessary to bring the computation time down.

The basic logic of the algorithm is presented in Figure 4.1. This flow chart depicts the dynamic cutting plane methodology for Strategies 1, 2, and 3. Each strategy differs by the way `vioRHS` is adjusted in each iteration. The extensions to the general logic are explained respectively for each strategy.

When the problem instance is fed to the algorithm, it starts optimizing the first sub-problem to find the lower bound solution X^* and the lower bound objective value Z_{lb}. The solvers used are CSDP version 5.0 [8] and SDPT3 version 4.0 [40]. With the newly obtained X^* and the appropriate `vioRHS` value, the algorithm carries out the calculation as laid out in Equation (4.2) to assess violations. If the left-hand-side value is less than `vioRHS`, a violation occurs. The indices and the left-hand-side value are recorded for later use. Note that the initial `vioRHS` is chosen to be -0.4. The initial `vioRHS` should not be too high (in terms of the magnitude), or otherwise no violations will be found as the standard is too slack. On the other hand, if the initial `vioRHS` is set too low, then it will take a very long time for the initial round of violation assessment, since no cuts have been added in the first round and there are still plenty of potential violations that can be detected.

In the case when no violation is detected, the algorithm will exit the loop. This usually happens when the initial `vioRHS` is too high for smaller instances, or when

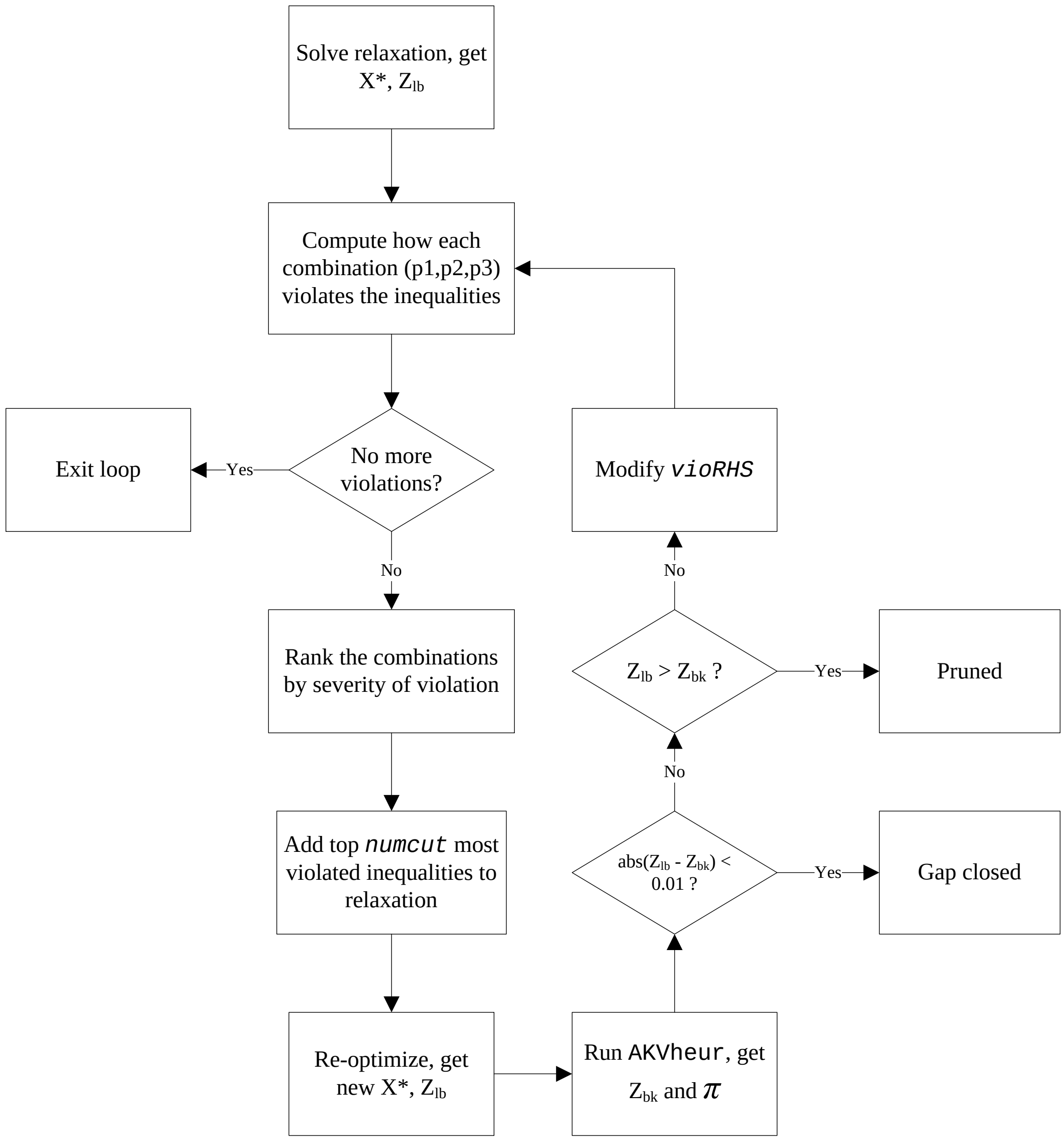

Figure 4.1: General cutting plane algorithm

`vioRHS` has not been reduced quickly enough in the middle of the process. However, new feature has been added to the newer cutting plane strategy (Strategies 4, 5, and 6) to combat these short-comings that may disrupt the computation and cause premature termination. This new feature is detailed in Section 4.1.4. On a side note, if branch and cut were to be used, branching would take place at this step upon exiting the loop. However, since the relaxations used in this thesis are good enough, branching was never necessary.

If there are any violations detected, these violated inequalities will be sorted by their recorded left-hand-side value, which signifies the severity of violation. The higher the left-hand-side value in terms of magnitude, the further away the inequality is from zero, and the more severe the violation. Therefore, the inequalities are sorted in a decreasing order of severity. A preset number of the inequalities from the top of the list are then chosen to be added to the relaxation sub-problem. This preset number is `numcut`. Parameter `numcut` sets the maximum number of cuts that can be added for each iteration. If less than `numcut` violations were found, all of them will be added, but the `vioRHS` will need adjustment so that more violations can be found. The modification of `vioRHS` will be discussed in more detail later.

By adding a number of most violated inequalities, a new relaxation sub-problem is obtained. By solving the new sub-problem, a new lower bound solution X^* and objective value Z_{lb} are obtained. Since there were already a number of inequalities added as new constraints, the new Z_{lb} should be higher and hence closer to the optimal solution. Using the newly obtained solution, the function `AKVheur` will utilize the AKV Heuristic with some help of 2-opt to find a set of feasible solution: π, which represents the permutation of departments, and Z_{bk}, which denotes the best-known objective value or the upper bound. These newly-obtained solution helps us to calculate the gap between the lower bound and the upper bound. The gap tells us about the state and condition of the cutting plane optimization process. If the lower bound Z_{lb} and the upper bound Z_{bk} are very close to each other, then the gap is closed and optimality is reached. For the first three strategies, we used $|Z_{lb} - Z_{bk}| \leq 0.01$ to declare the gap closed, but it is sufficient to define the condition of gap closed as $|Z_{lb} - Z_{bk}| \leq 0.49$, because by examining the make-up of the objective function (2.5) it is evident that the objective values will always be half-integer, given that the input data are all integer. The latter criterion was used starting with Strategy 4. On the other hand, if the lower bound Z_{lb} becomes higher than the best known Z_{bk}, the sub-problem becomes invalid and hence pruned. If not, the cutting plane process will continue to the next step where `vioRHS` is modified based on the state of the optimization process. After the adjustment of `vioRHS`, the standard of the violation assessment is changed, and the algorithm will try to find new violated inequalities with the newly-obtained information.

4.1.1 Strategy 1

Strategy 1 follows closely the general approach illustrated in Figure 4.1. Figure 4.2 demonstrates the methodology of `vioRHS` modification in Strategy 1. The parameter `vioRHS` starts off at -0.4. During steady improvement, i.e. the percentage difference between the new Z_{lb} from the current iteration and the old Z_{lb} from the previous iteration exceeds 0.1%, the magnitude of `vioRHS` is increased by 1%. However, if the improvement of Z_{lb} stagnates such that the percentage difference is less than 0.1%, the magnitude of `vioRHS` will be reduced by 0.2 or by half, whichever results in a smaller change. Nevertheless, the change will not let `vioRHS` fall below -0.03. However, if the problem runs low on the number of cuts found, i.e. number of cuts found is less than `numcut`, a bigger reduction is required to keep the problem running. The parameter `vioRHS` will be automatically reduced by 75% or by 0.2, whichever results in a smaller drop.

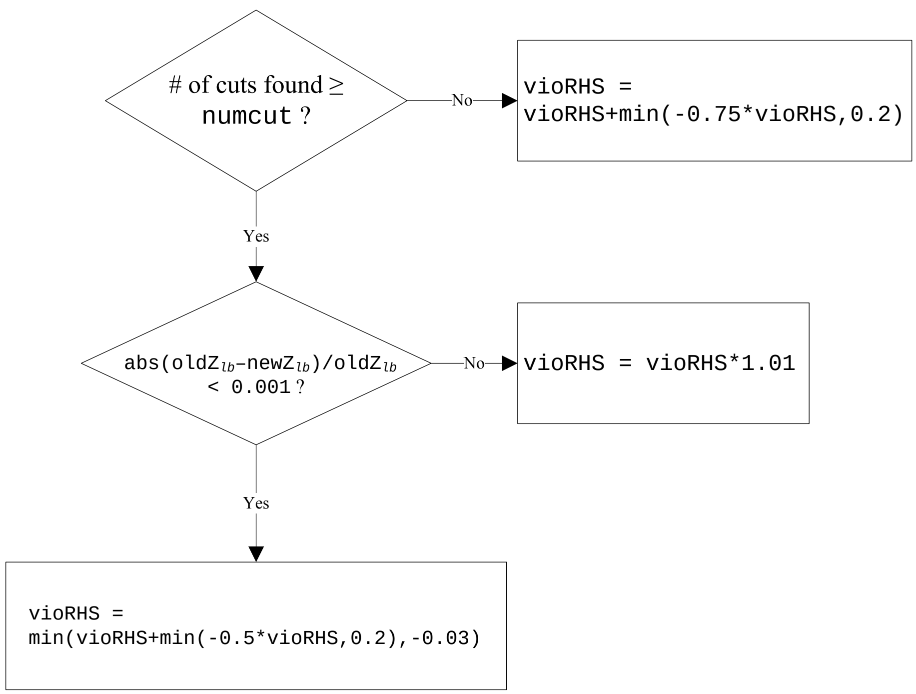

Figure 4.2: Strategy 1 on modification of `vioRHS`

4.1.2 Strategy 2

Strategy 2, as illustrated in Figure 4.3 is similar to Strategy 1 with some minor changes in parameters. For instance, the improvement of Z_{lb} is considered steady if the percentage difference between the new and the old Z_{lb} exceeds 0.13%, instead of 0.1% as in Strategy 1. When the improvement is steady, the magnitude of `vioRHS` is increased by 1%. Otherwise, the magnitude of `vioRHS` will be cut down by 0.2 or by 20%, instead of by half as in Strategy 1, whichever results in a smaller change. Similar to Strategy 1, the change will not let `vioRHS` drop below -0.03. Also, if the number of cuts found is less than `numcut`, `vioRHS` will be given a bigger adjustment of 75% reduction or by 0.2, whichever results in a smaller change.

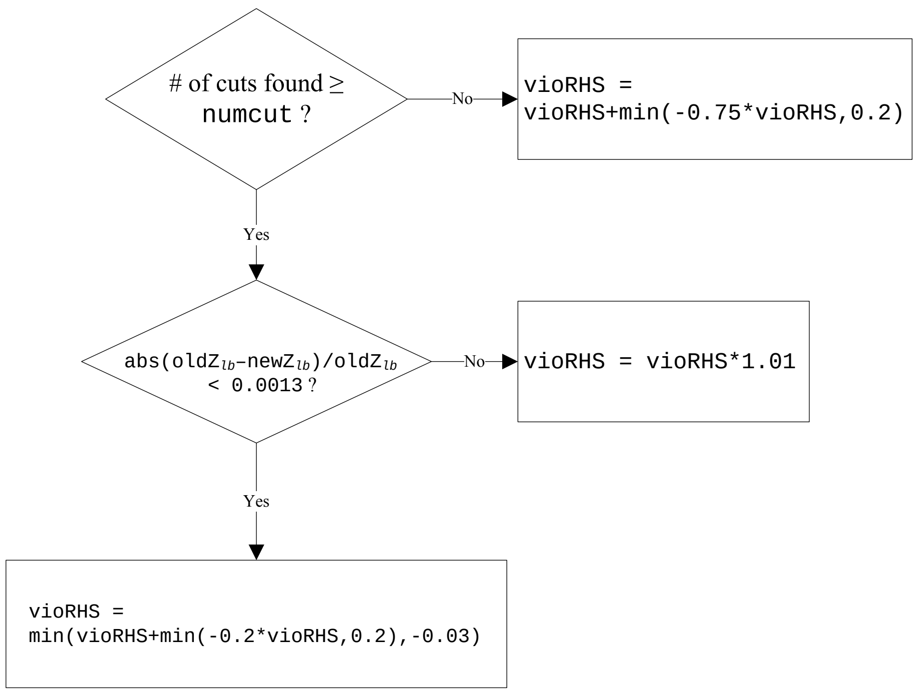

Figure 4.3: Strategy 2 on modification of `vioRHS`

The comparison of results for Strategy 1 and 2 is detailed in Section 4.2.1. Table 4.1 and Table 4.2 are the computation breakdown of the two circled data points in Figure 4.9. The table is explained in detail in Section 4.2.1. The three circled time durations in Table 4.1 are the time intervals for finding and sorting the cuts after the algorithm decides that the improvement for Z_{lb} is not fast enough, and hence it lowers the `vioRHS` by 50%. Consequently, the time required for finding and sorting the cuts surged up because the change of 50% is too aggressive. There are suddenly too many potential cuts that can be found and sorted. Therefore, in Strategy 2, we changed the cut in `vioRHS` from 50% to 20% when the improvement is not steady enough. The resultant change in computing time for cuts is drastically shortened as seen in Table 4.2. Because the reduction in `vioRHS` is smaller in Strategy 2, we can start decreasing `vioRHS` earlier, in the senes that the standard for steady improvement of Z_{lb} is now higher. In Strategy 2, the percentage difference of the current and the previous Z_{lb} has to be above 0.13% to be qualified as improving steadily. Consequently, the algorithm reacts to make minor adjustment to `vioRHS` sooner and more frequently in the process.

4.1.3 Strategy 3

In Strategy 3, the gap between Z_{lb} and Z_{bk} is introduced as another criterion to assess the adjustment of `vioRHS`. Figure 4.4 shows that given the number of cuts found is higher than `numcut`, if the percentage difference between Z_{lb} and Z_{bk} is less than 0.2%, `vioRHS` will not be changed. Otherwise, `vioRHS` will be adjusted in the same way as in Strategy 2. By keeping `vioRHS` unchanged when the gap is small, the modification of `vioRHS` becomes smoother, which is observed to yield shorter computation time. Figure 4.12 compares the two strategies, and Tables 4.3 and 4.4 illustrate the small improvement as the result of Strategy 3.

4.1.4 Strategy 4

Two new features are added in Strategy 4. As shown in Figure 4.5, when the algorithm cannot find any violations, it will check whether there has been any triangle inequality constraints added since the beginning. If there is none, it means that the initial `vioRHS` of -0.4 is probably too high for this particular instance. It will happen if the instance is small, such as when $n \leq 10$. Therefore, the algorithm will reduce the magnitude of `vioRHS` by 75% to start all over again. Otherwise, it means that the problem has run out of cuts and hence the cutting plane algorithm terminates.

Another new function in Strategy 4 is to remove non-binding inequality constraints. For numerical reason, the positive slack is considered non-binding if it is greater than 0.1. Removing non-binding inequality constraints help to keep the

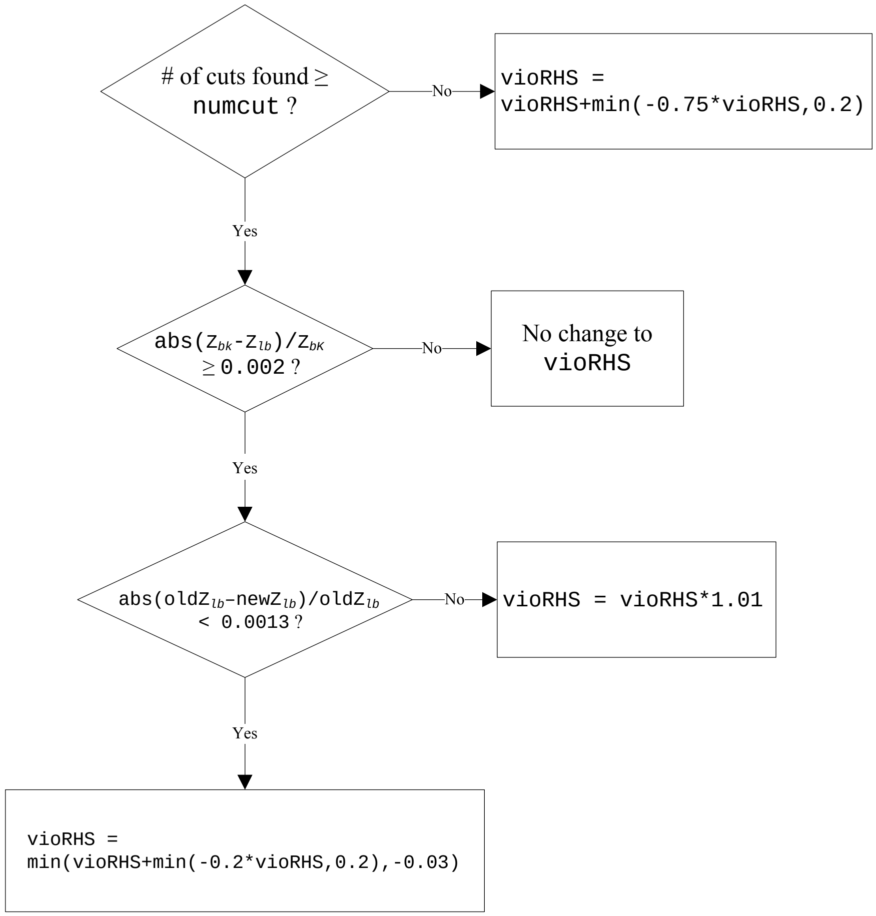

Figure 4.4: Strategy 3 on modification of `vioRHS`

problem size small and gives more room for future cut addition. This is because the algorithm gets rid of a number of constraints, say `numslack`, at the end of an iteration, but in the next iteration, `numslack` additional cuts on top of the given number `numcut` can be added to the new sub-problem. This approach facilitates the pace of lower bound improvement, which is observed in Figure 4.15. Tables 4.9 and 4.10 also demonstrate the experimental result of this anticipated improvement, which is explained in Section 4.2.3.

Strategy 4 modifies `vioRHS` as in Strategy 3. See Figure 4.4 for the illustration of the algorithm.

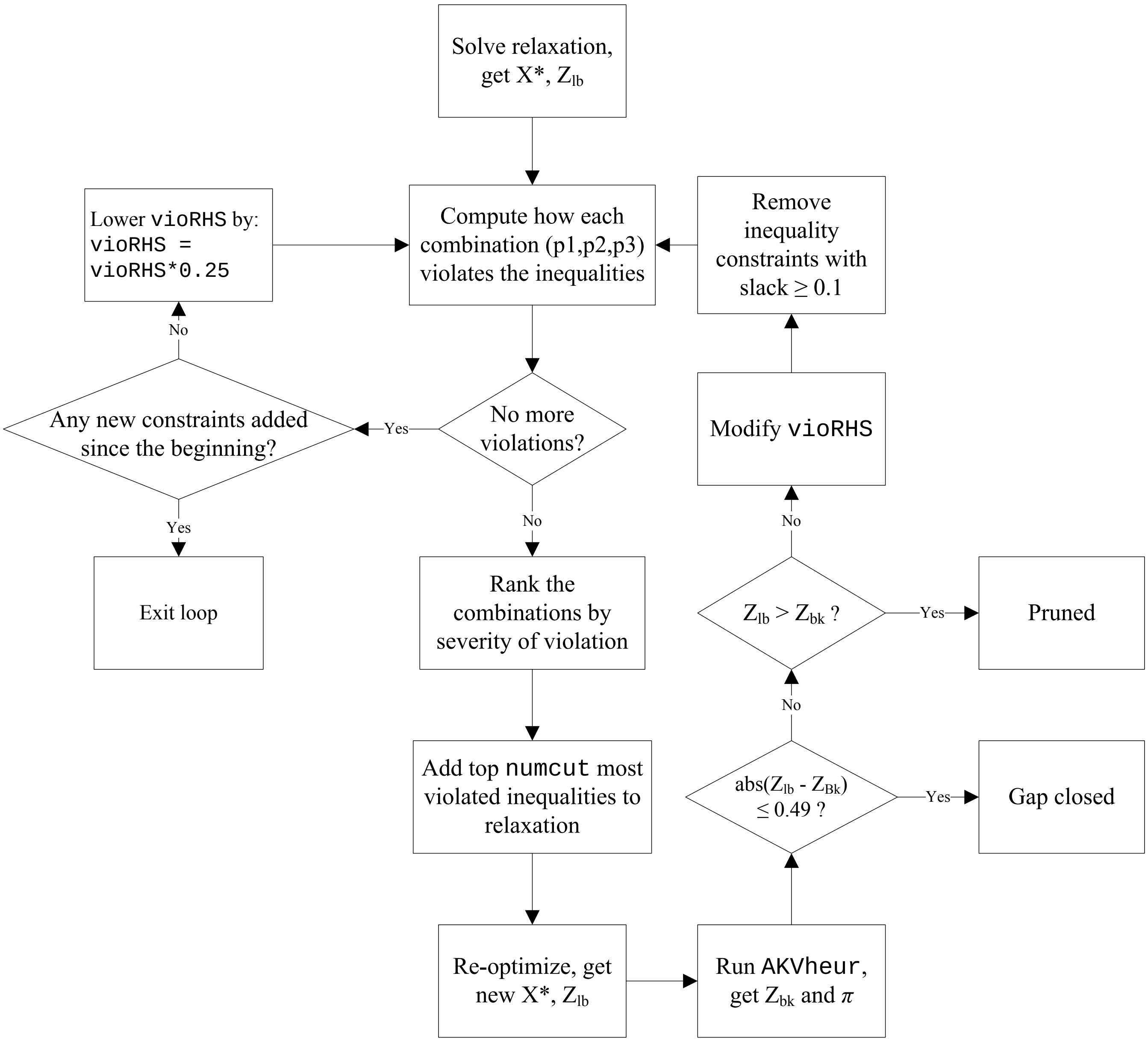

Figure 4.5: Cutting plane algorithm for Strategy 4

4.1.5 Strategy 5

Strategy 5 includes two new features. One feature is that if the number of cuts found is less than half of `numcut`, `vioRHS` will be reduced to re-start the violations search with the new standard. This approach bypasses the time-consuming optimization calculation when the number of inequality constraints to be added is low and hence has smaller impact on lower bound improvement. This is especially helpful when the sub-problem becomes large after many inequality constraints have been added.

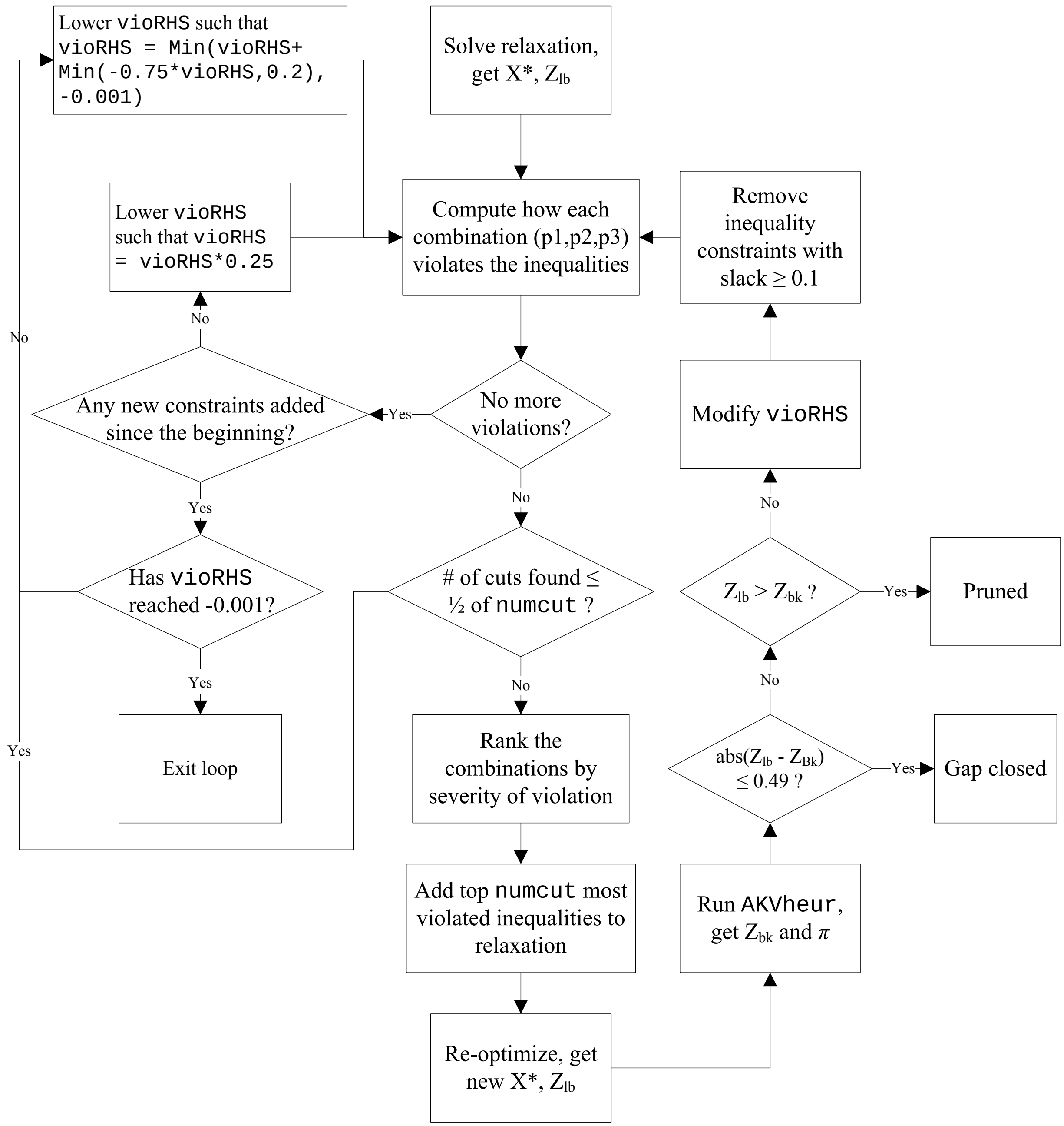

Figure 4.6: Cutting plane algorithm for Strategy 5

The other new feature of Strategy 5 is the continued search for violations to avoid premature termination. After detecting that no violations are found and that it is not a small-instance issue, `vioRHS` will be reduced further until it reaches -0.001, a very small number sufficiently close to zero. Please refer to Figure 4.6

for the methodology of the cutting plane algorithm in Strategy 5. On the other hand, the modification of `vioRHS` is executed the same way as in Strategy 3. See Figure 4.4 for the illustration of the algorithm.

The two new features not only successfully prevent premature termination, but they also allow the cutting plane process to be more efficient and hence lower the computing time. The success of Strategy 5 can be observed in Figure 4.18, as well as in Tables 4.11 and 4.12.

4.1.6 Strategy 6

Strategy 6 is similar to Strategy 5 other than the way `vioRHS` is adjusted. As reflected in Figures 4.7 and 4.8, this new approach ensures that the magnitude of each adjustment to `vioRHS` will not exceed 0.1. This technique further smoothes the process of `vioRHS` reduction and thus lowers the computation time. See Figures 4.21, 4.22, and 4.23 for the comparison graphs of Strategies 5 and 6 for instances AV25-2, AV25-1, and HeKu20. The labeled data points in Figure 4.21 show the lowest computing time thus far, and they are detailed in Tables 4.17 and 4.18.

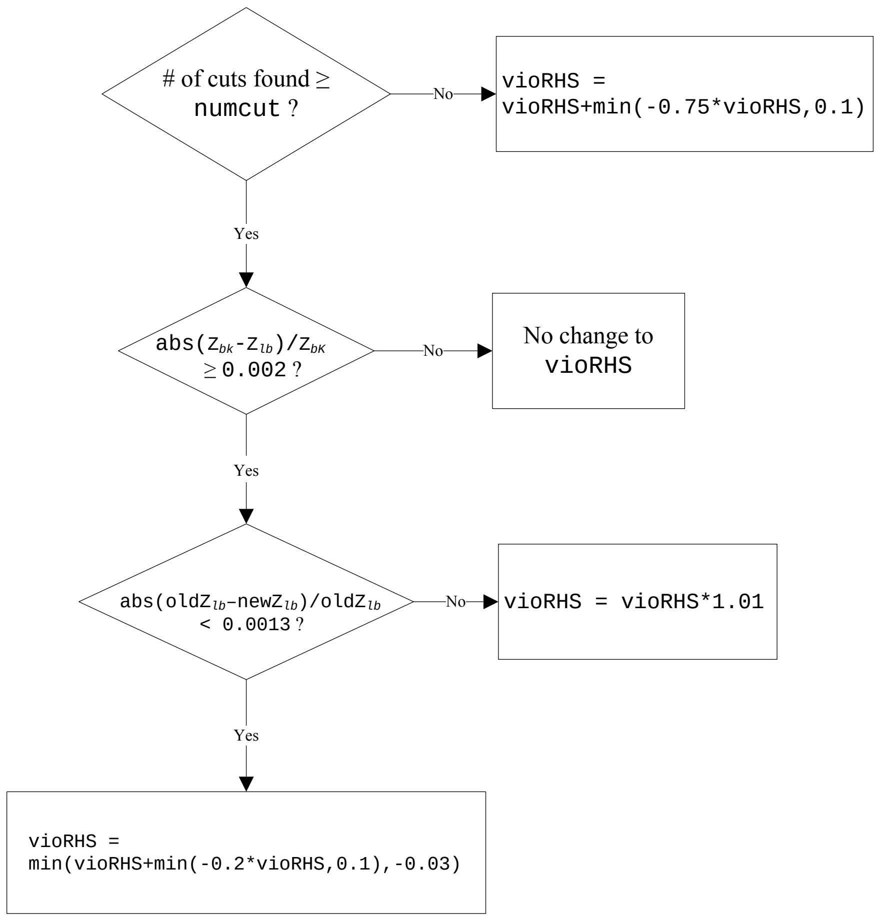

Figure 4.7: Strategy 6 on modification of `vioRHS`

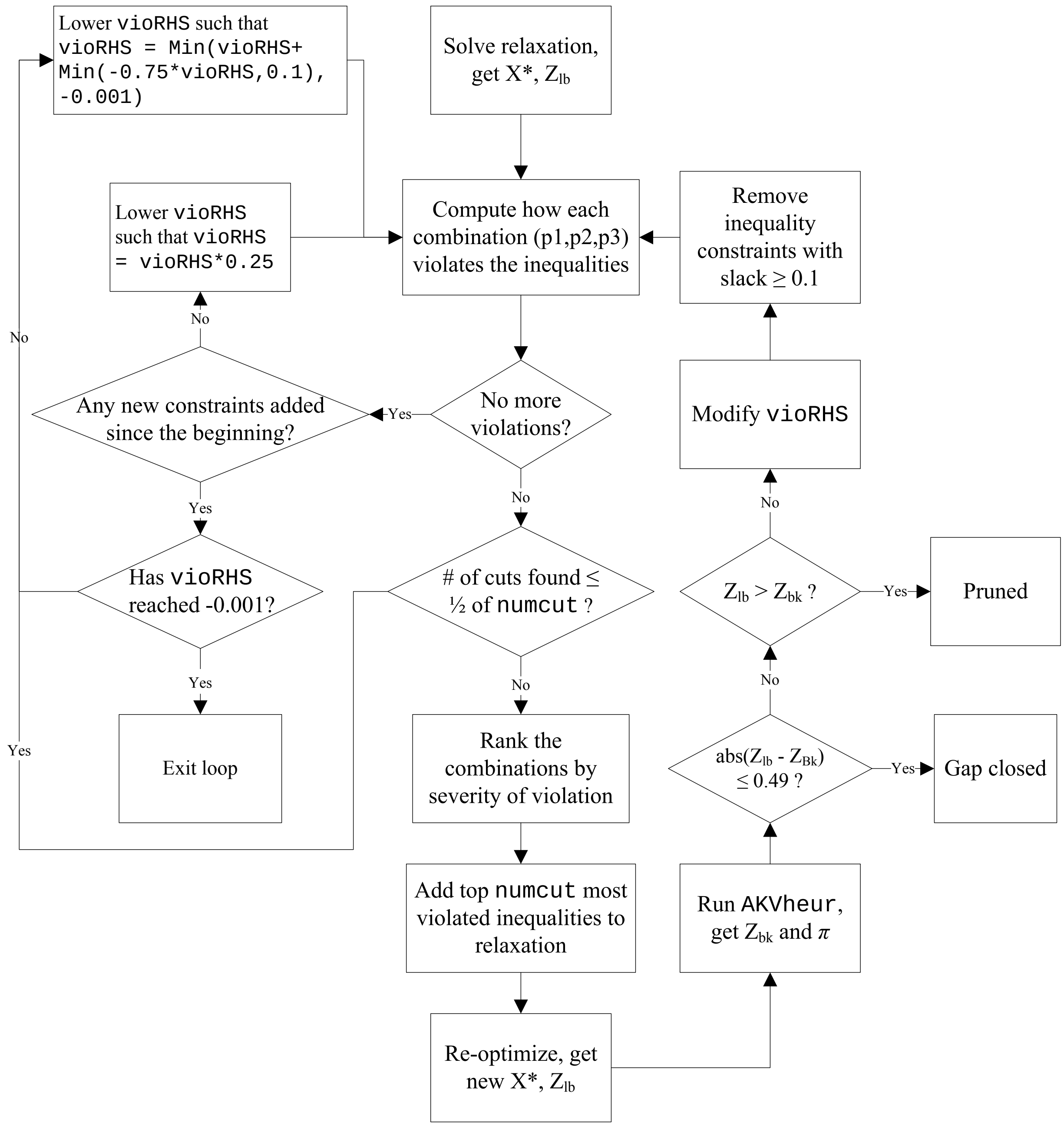

Figure 4.8: Cutting plane algorithm for Strategy 6

4.2 Performance of the Six Strategies

This section discusses the performance of each strategy and how each strategy is developed based on the earlier results. For this section on the development of the basis strategies, medium-sized instances such as HeKu20, AV25-1, and AV25-2 were used. A few larger instances, such as HeKu30 and STE36-1 were attempted, but even the best-performing strategy out of the six basic strategies were too slow. Therefore some minor modification was made to create another two strategies for the large instances, which will be discussed in Chapter 5. HeKu20 and HeKu30 are from Heragu and Kusiak in [20], while AV25-1 and AV25-2 are from Anjos and Vannelli in [5]. The other larger instances will be explained later in Chapter 5. Please see Appendix C for the complete listing of all the instances used in this thesis.

The medium-sized instances were solved by AKV and AKV' using SDPT3 version 4.0 [40] on a 2.0GHz Dual Opteron with 16Gb of RAM. Each method was run 15 times using different `numcut` setting, ranging from 100 to 900. Several graphs were generated to study the behaviour of each method and the effect of `numcut` on computing time. We would also like to find out a pattern of the effect of `numcut` so that we can use the most effective `numcut` value to solve larger problems.

4.2.1 From Strategy 1 to Strategy 2

The changes between Strategy 1 and Strategy 2 may seem small, but the improvement in computing time is drastic. Figure 4.9 compares AKV and AKV' for Strategy 1 and 2 when solving instance AV25-2. It should be noted that AKV'1 denotes the combination of AKV' using Strategy 1. Also, there are two missing points in this graph, namely AKV'1 and AKV'2 at `numcut` = 100. Any missing point in the curves means that the corresponding trial is incomplete. This may be due to limitations of the algorithm, especially in the earlier stategies, or running out of memory, which happens when solving large instances. After a few versions of modifications on the algorithm, the problem of running out of cuts is eliminated for Strategy 5 and 6.

When doing an overall comparison of AKV and AKV', Figure 4.9 clearly tells us that AKV' outperforms AKV, since both AKV curves are almost always above

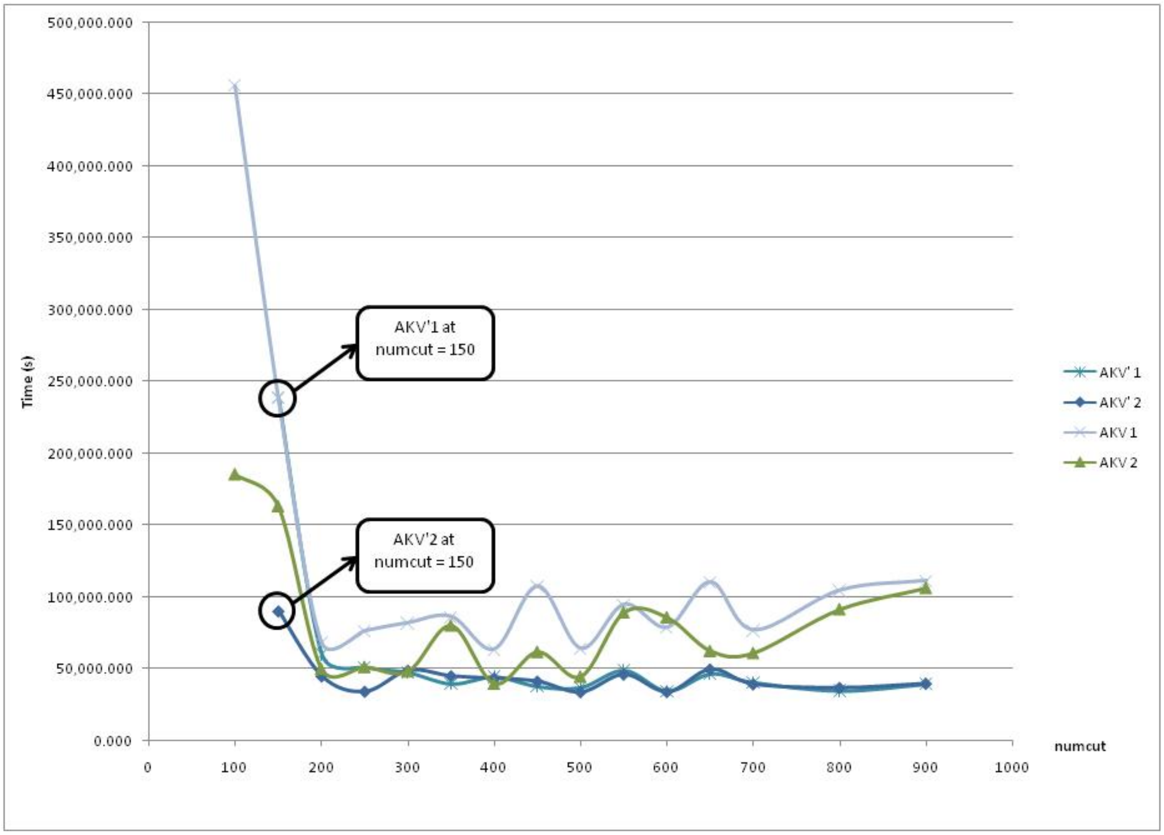

Figure 4.9: Comparison of Strategy 1 and Strategy 2 for AV25-2

the AKV' curves. This distinction is especially obvious for small `numcut`. While the AKV' curves steady off at low computation time as `numcut` increases, the AKV curves climb up and deviate away from the AKV' curves.

When comparing Strategy 1 and 2, we need to compare AKV1 with AKV2, and AKV'1 with AKV'2. For AKV1 and AKV2, the AKV2 curve is almost always below the AKV1 curve. At `numcut` = 100, it takes AKV1 nearly 2.5 times the computation time for AKV2. For AKV'1 and AKV'2, the difference in computing time at `numcut` = 150 is also very high, where the total computing time for AKV'1 is 2.6 times of AKV'2. But the two AKV' curves seem to converge as `numcut` increases, and hence the distinction becomes very small. However, we can still conclude that the change in Strategy 2 makes an improvement for the computation effeciency.

The conclusion also applies to the other instances as seen in Figure 4.10 for AV25-1 and Figure 4.11 for HeKu20. It should be noted that the behaviour in

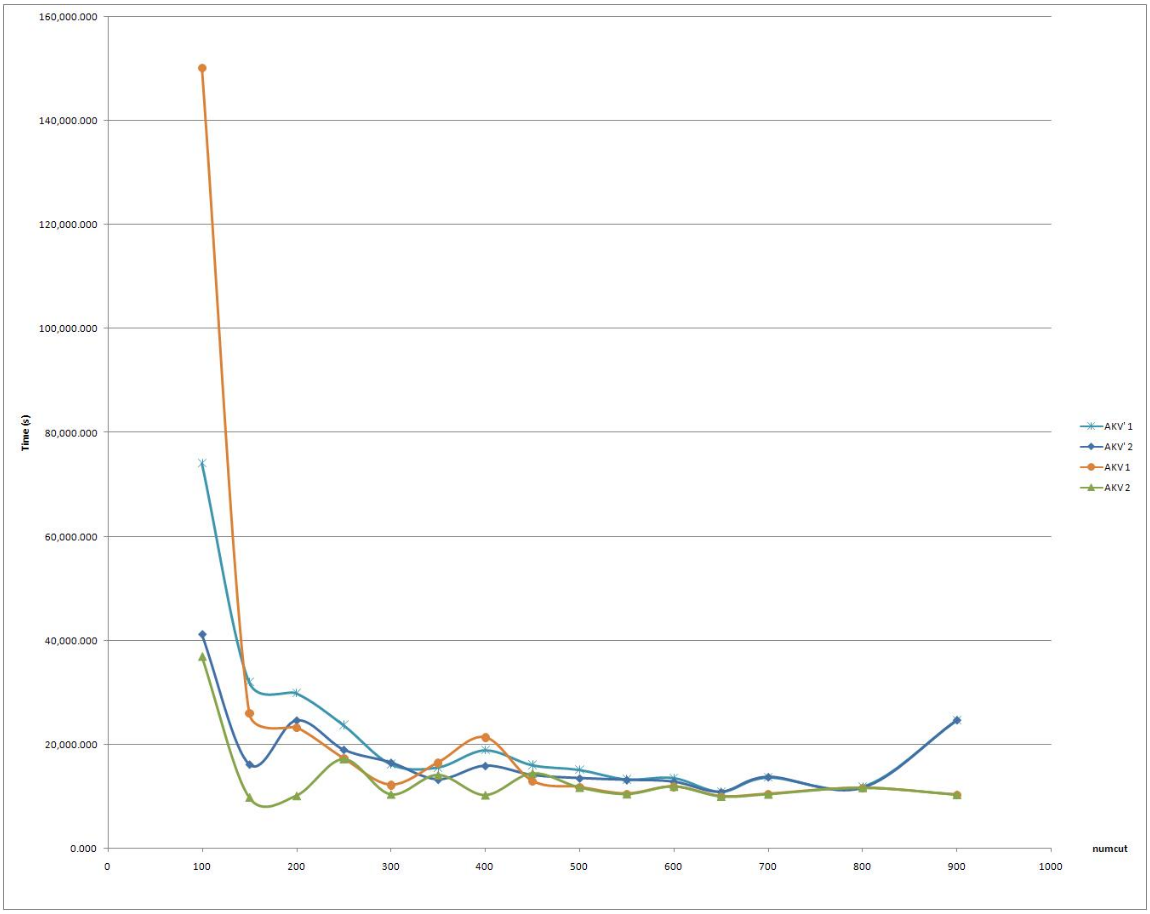

Figure 4.10: Comparison of Strategy 1 and Strategy 2 for AV25-1

AV25-1 is quite different from the other two instances because the AKV and AKV' curves seem to steady off and converge as `numcut` increases. The difference between Strategies 1 and 2 also seems to diminish as `numcut` increases. Although at `numcut` = 100, the performances of AKV'1 and AKV'2 are similar, the computing time for AKV'2 is still much smaller than AKV1. Therefore, we can still confirm the improvement of AKV' over AKV and Strategy 2 over Strategy 1.

Tables 4.1 and 4.2 summarize the duration of each iteration of the cutting plane process and how `vioRHS` affects the computing time. The circled time duration shows the most impactful results due to the change in algorithm, which is discussed in detail in Section 4.1.2. The fourth column in Table 4.1 records the accumulative clock time in second from the beginning to the end of a trial. The third column is the duration of each iteration, which is calculated by taking the difference between the two subsequent clock times. The shaded duration represents the time spent

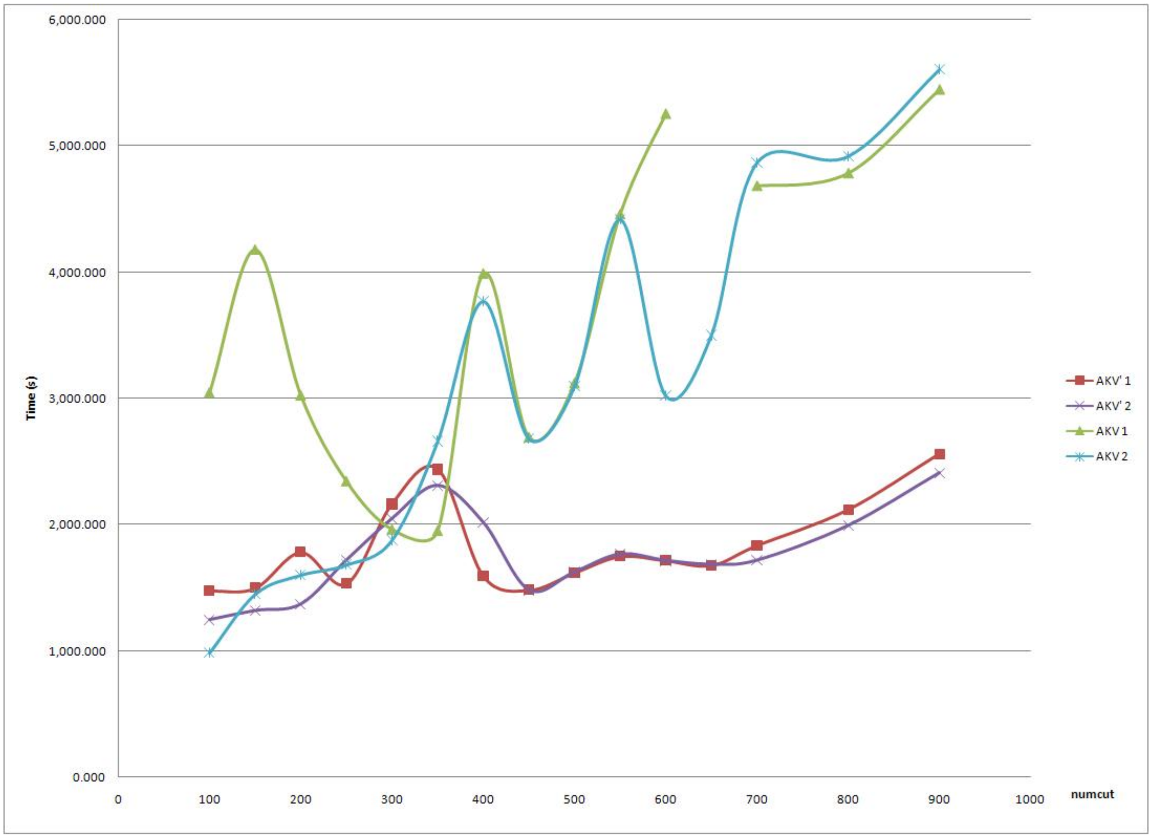

Figure 4.11: Comparison of Strategy 1 and Strategy 2 for HeKu20

in the optimization solver. The unshaded time interval denotes the amount of time taken in between the optimization steps, which includes calculating the Z_{bk} and π, finding and determining violations, sorting and forming the cuts. The first shaded duration is the total time taken to calculate the lower bound at root node with no cuts added, while the shaded number in the fifth column is the lower bound objective value in root node. The second column in Table 4.1 lists out the `vioRHS` at each iteration. As shown in Figure 4.1, `vioRHS` is modified after the condition check after exiting the optimization solver. Hence the `vioRHS` values are placed beside the unshaded time interval, during which the `vioRHS` is modified. Occasionally, a number may sit above a `vioRHS` value, e.g. the 4 above `vioRHS` of -0.2080 in Table 4.1. This number represents the number of cuts found in this trial. This number is recorded if the number of cuts found is smaller than `numcut`. The first column calculates the change in `vioRHS` by taking the fraction of new `vioRHS` by the previous `vioRHS`.

AKV' 1		numcut	150						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			1.613						
		59.890	61.503	36,355.5					
	-0.4000	15.799	77.303						
		108.429	185.731	36,464.0	37,177.5	108.5	0.298%	713.5	1.919%
1.01	-0.4040	83.594	269.325						
		122.291	391.616	36,577.5	37,137.5	113.5	0.311%	560	1.508%
1.01	-0.4080	84.191	475.806						
	4	127.345	603.151	36,591.5	37,137.5	14.0	0.038%	546	1.470%
0.51	-0.2080	1,825.353	2,428.504						
		143.803	2,572.307	36,675.5	37,137.5	84.0	0.230%	462	1.244%
1.01	-0.2101	197.870	2,770.178						
		161.359	2,931.537	36,710.0	37,137.5	34.5	0.094%	427.5	1.151%
0.50	-0.1051	8,358.544	11,290.081						
		187.009	11,477.090	36,753.5	37,137.5	43.5	0.118%	384	1.034%
1.01	-0.1061	5,340.500	16,817.590						
		230.918	17,048.508	36,799.5	37,137.5	46.0	0.125%	338	0.910%
1.01	-0.1072	3,795.133	20,843.640						
		284.273	21,127.914	36,830.5	37,137.5	31.0	0.084%	307	0.827%
0.50	-0.0536	13,915.678	35,043.592						
		326.482	35,370.074	36,855.5	37,116.5	25.0	0.068%	261	0.703%
0.56	-0.0300	23,810.458	59,180.532						
		369.904	59,550.436	36,891.5	37,116.5	36.0	0.098%	225	0.606%
1.00	-0.0300	23,180.167	82,730.603						
		404.736	83,135.338	36,923.5	37,116.5	32.0	0.087%	193	0.520%
1.00	-0.0300	17,736.896	100,872.235						
		479.898	101,352.133	36,950.5	37,116.5	27.0	0.073%	166	0.447%
1.00	-0.0300	14,580.197	115,932.330						
		537.639	116,469.969	36,971.5	37,116.5	21.0	0.057%	145	0.391%
1.00	-0.0300	14,245.535	130,715.504						
		589.392	131,304.896	36,985.5	37,116.5	14.0	0.038%	131	0.353%
1.00	-0.0300	13,641.240	144,946.136						
		646.160	145,592.296	37,011.5	37,116.5	26.0	0.070%	105	0.283%
1.00	-0.0300	11,316.666	156,908.962						
		714.260	157,623.222	37,015.0	37,116.5	3.5	0.009%	101.5	0.273%
1.00	-0.0300	10,756.318	168,379.540						
		778.025	169,157.565	37,028.5	37,116.5	13.5	0.036%	88	0.237%
1.00	-0.0300	9,259.262	178,416.828						
		837.664	179,254.492	37,038.5	37,116.5	10.0	0.027%	78	0.210%
1.00	-0.0300	8,629.014	187,883.506						
		941.271	188,824.777	37,051.5	37,116.5	13.0	0.035%	65	0.175%
1.00	-0.0300	8,028.163	196,852.940						
		1,022.171	197,875.110	37,061.5	37,116.5	10.0	0.027%	55	0.148%
1.00	-0.0300	6,156.170	204,031.280						
		1,128.751	205,160.032	37,071.5	37,116.5	10.0	0.027%	45	0.121%
1.00	-0.0300	5,010.351	210,170.383						
		1,237.389	211,407.772	37,078.0	37,116.5	6.5	0.018%	38.5	0.104%
1.00	-0.0300	4,024.221	215,431.993						
		1,326.392	216,758.385	37,088.0	37,116.5	10.0	0.027%	28.5	0.077%
1.00	-0.0300	2,390.861	219,149.246						
		1,419.914	220,569.160	37,094.0	37,116.5	6.0	0.016%	22.5	0.061%
1.00	-0.0300	1,818.116	222,387.276						
		1,518.393	223,905.669	37,101.0	37,116.5	7.0	0.019%	15.5	0.042%
1.00	-0.0300	1,049.185	224,954.854						
		1,656.013	226,610.867	37,104.5	37,116.5	3.5	0.009%	12	0.032%
1.00	-0.0300	797.633	227,408.500						
		1,803.750	229,212.251	37,111.0	37,116.5	6.5	0.018%	5.5	0.015%
1.00	-0.0300	91.254	229,303.505						
		1,974.303	231,277.808	37,113.0	37,116.5	2.0	0.005%	3.5	0.009%
1.00	-0.0300	89.281	231,367.088						
		2,143.160	233,510.248	37,115.5	37,116.5	2.5	0.007%	1	0.003%
1.00	-0.0300	84.196	233,594.444						
		2,369.327	235,963.771	37,116.0	37,116.5	0.5	0.001%	0.5	0.001%
1.00	-0.0300	83.950	236,047.721						
		2,465.805	238,513.526	37,116.5	37,116.5	0.5	0.001%	0	0.000%
		104.147	238,617.673						

Table 4.1: Computing AV25-2 using AKV'1 with `numcut` = 150

AKV' 2		numcut	150						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.092						
		100.403	102.495	35,960.5					
	-0.4000	15.984	118.480						
		155.253	273.733	36,208.5	37,133.5	248.0	0.690%	925	2.491%
1.01	-0.4040	83.851	357.584						
		181.755	539.339	36,311.0	37,133.5	102.5	0.283%	822.5	2.215%
1.01	-0.4080	82.901	622.241						
	1	176.836	799.076	36,314.0	37,133.5	3.0	0.008%	819.5	2.207%
0.51	-0.2080	745.708	1,544.784						
		190.759	1,735.543	36,432.0	37,133.5	118.0	0.325%	701.5	1.889%
1.01	-0.2101	224.930	1,960.473						
		210.804	2,171.277	36,518.0	37,133.5	86.0	0.236%	615.5	1.658%
1.01	-0.2122	143.260	2,314.537						
		231.824	2,546.361	36,559.5	37,116.5	41.5	0.114%	557	1.501%
0.80	-0.1698	320.687	2,867.048						
		253.911	3,120.959	36,620.0	37,116.5	60.5	0.165%	496.5	1.338%
1.01	-0.1715	157.481	3,278.439						
		277.772	3,556.211	36,669.5	37,116.5	49.5	0.135%	447	1.204%
1.01	-0.1732	95.998	3,652.209						
		293.565	3,945.774	36,701.5	37,116.5	32.0	0.087%	415	1.118%
0.80	-0.1386	396.915	4,342.690						
		296.532	4,639.222	36,734.5	37,116.5	33.0	0.090%	382	1.029%
0.80	-0.1108	676.000	5,315.222						
		309.452	5,624.674	36,786.5	37,116.5	52.0	0.142%	330	0.889%
1.01	-0.1120	405.896	6,030.570						
		370.725	6,401.295	36,816.5	37,116.5	30.0	0.082%	300	0.808%
0.80	-0.0896	785.406	7,186.702						
		397.509	7,584.210	36,857.0	37,116.5	40.5	0.110%	259.5	0.699%
0.80	-0.0717	2,135.008	9,719.218						
		517.306	10,236.524	36,870.0	37,116.5	13.0	0.035%	246.5	0.664%
0.80	-0.0573	3,442.916	13,679.440						
		550.418	14,229.858	36,907.5	37,116.5	37.5	0.102%	209	0.563%
0.80	-0.0459	4,060.986	18,290.843						
		571.686	18,862.529	36,926.0	37,116.5	18.5	0.050%	190.5	0.513%
0.80	-0.0367	5,902.289	24,764.819						
		611.288	25,376.107	36,948.5	37,116.5	22.5	0.061%	168	0.453%
0.82	-0.0300	9,032.898	34,409.005						
		630.756	35,039.761	36,975.0	37,116.5	26.5	0.072%	141.5	0.381%
1.00	-0.0300	6,041.253	41,081.013						
		686.648	41,767.661	36,990.0	37,116.5	15.0	0.041%	126.5	0.341%
1.00	-0.0300	5,272.728	47,040.389						
		710.685	47,751.075	37,005.0	37,116.5	15.0	0.041%	111.5	0.300%
1.00	-0.0300	4,348.373	52,099.448						
		772.603	52,872.051	37,023.5	37,116.5	18.5	0.050%	93	0.251%
1.00	-0.0300	3,198.914	56,070.965						
		811.676	56,882.640	37,040.0	37,116.5	16.5	0.045%	76.5	0.206%
1.00	-0.0300	2,838.567	59,721.208						
		938.548	60,659.755	37,046.5	37,116.5	6.5	0.018%	70	0.189%
1.00	-0.0300	2,688.438	63,348.193						
		963.391	64,311.584	37,060.0	37,116.5	13.5	0.036%	56.5	0.152%
1.00	-0.0300	2,050.888	66,362.472						
		1,055.367	67,417.839	37,068.5	37,116.5	8.5	0.023%	48	0.129%
1.00	-0.0300	1,699.838	69,117.677						
		1,114.629	70,232.306	37,075.5	37,116.5	7.0	0.019%	41	0.110%
1.00	-0.0300	1,453.879	71,686.185						
		1,157.222	72,843.407	37,083.5	37,116.5	8.0	0.022%	33	0.089%
1.00	-0.0300	951.951	73,795.358						
		1,219.342	75,014.700	37,087.0	37,116.5	3.5	0.009%	29.5	0.079%
1.00	-0.0300	949.664	75,964.364						
		1,292.490	77,256.854	37,098.0	37,116.5	11.0	0.030%	18.5	0.050%
1.00	-0.0300	241.592	77,498.447						
		1,359.437	78,857.883	37,103.5	37,116.5	5.5	0.015%	13	0.035%
1.00	-0.0300	142.429	79,000.312						
		1,429.254	80,429.566	37,108.0	37,116.5	4.5	0.012%	8.5	0.023%
1.00	-0.0300	100.061	80,529.628						
		1,506.856	82,036.484	37,112.0	37,116.5	4.0	0.011%	4.5	0.012%
1.00	-0.0300	89.327	82,125.811						
		1,643.219	83,769.031	37,113.5	37,116.5	1.5	0.004%	3	0.008%
1.00	-0.0300	84.390	83,853.421						
		1,784.822	85,638.243	37,116.0	37,116.5	2.5	0.007%	0.5	0.001%
1.00	-0.0300	83.294	85,721.537						
	75	1,842.814	87,564.350	37,116.0	37,116.5	0.0	0.000%	0.5	0.001%
0.25	-0.0075	659.857	88,224.207						
		1,930.312	90,154.519	37,116.5	37,116.5	0.5	0.001%	0	0.000%
		89.478	90,243.997						

Table 4.2: Computing AV25-2 using AKV'2 with `numcut` = 150

4.2.2 From Strategy 2 to Strategy 3

The changes made to Strategy 3 are based on Strategy 2, which was explained in Section 4.1.3. The resultant improvement is marginal, as observed in Figure 4.12 and the two labeled data points, which are elaborated in Tables 4.3 and 4.4. When comparing AKV and AKV' using Figure 4.12, the observation is similar to the previous section, i.e. AKV' is faster, and hence better, than AKV, especially as `numcut` increases. However, the comparison becomes tricky as we start comparing Strategy 2 and Strategy 3. In a first glance of Figure 4.12, it is difficult to judge whether Strategy 3 outperforms Strategy 2 because while there are several data points showing Strategy 3 outperforms Strategy2, there are also several points indicating a worse result.

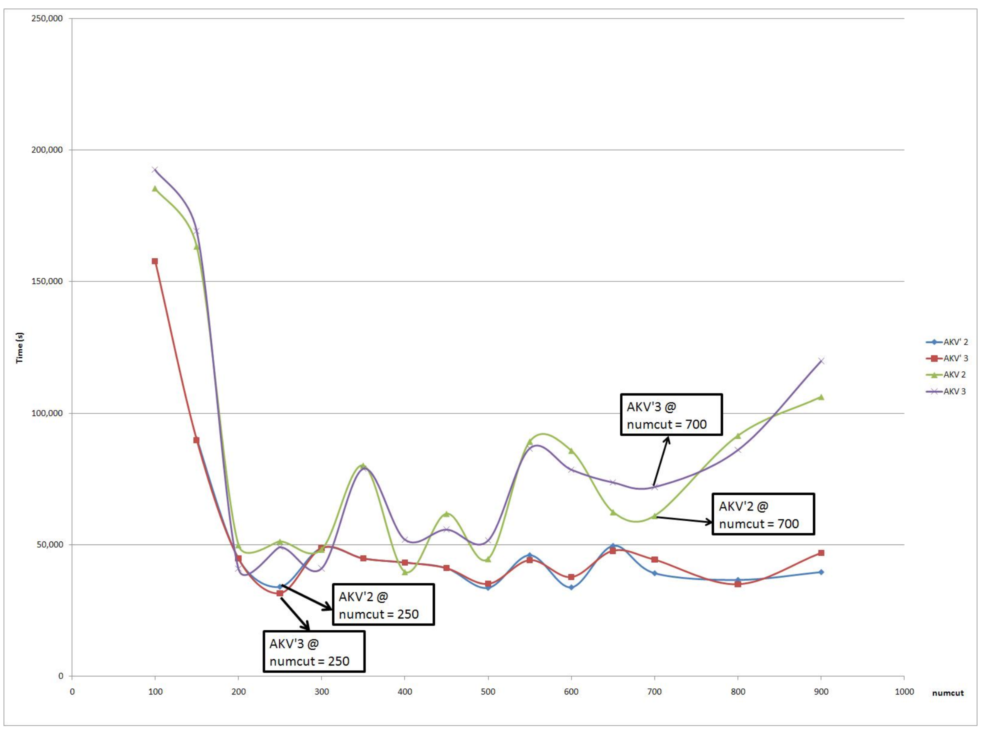

Figure 4.12: Comparison of Strategy 2 and Strategy 3 for AV25-2

Tables 4.3 and 4.4 are the time breakdowns for the two data points that exhibit a small improvement in the new strategy. In Table 4.3, `vioRHS` continues to decrease even when the gap between Z_{bk} and Z_{lb} is small. When the gap is small, too much modification to `vioRHS` may become too aggressive. Therefore, the computing time to find and sort the cuts increases considerably, as shown in the three circled time durations in Table 4.3. Table 4.4 illustrates the improvement when the `vioRHS` stays at -0.454. The three circled durations in Table 4.4 show that the increase in the sorting time dampens down much quickly in Strategy 3 than in Strategy 2, which gives a percentage improvement of 7.3% in terms of the total computing time.

However, there are also several data points exhibiting Strategy 3 yielding worse performance, such as those in Tables 4.5 and 4.6. The trial of AKV3 with `numcut` = 700 keeps `vioRHS` at -0.1359 when the gap becomes small. However, this `vioRHS` value becomes too high for the process, so the algorithm runs short of the cuts found. Only 46 cuts are added in the next iteration, which only makes a tiny improvement to the Z_{lb} while costing the overall process 5,671 seconds of optimization time. This iteration can be seen as wasted since a lot of time is invested with only a small return. This is one reason why Strategy 3 performs poorly for this trial. Because the algorithm runs out of the cuts, it tries to make a major reduction to the `vioRHS` so that it can continue finding more cuts. This major reduction is however too aggressive, which makes the following time interval for finding and sorting the cuts significantly surge up, as circled in Table 4.6. Finally, because the trial wasted one iteration adding only 46 cuts, an additional iteration is required to close the gap in AKV3. Therefore, the AKV3 trial needs to take additional 10,917 seconds to reach optimality, which is 17.9% longer than the AKV3 trial.

The problem with wasting an iteration when `vioRHS` stays too high such that the algorithm cannot find cuts was easily fixed in Strategy 5. The overly aggressive reduction in `vioRHS` was also changed in the later strategies. Finally, although the improvement for Strategy 3 seems trivial for medium-sized instances, one can expect to see a bigger difference for large instances when the process of finding and sorting cuts becomes much more complicated and hence more time-consuming. Therefore the modification made in Strategy 3 is still kept in the following strategies.

Figure 4.13 compares AKV2, AKV3, AKV'2, and AKV'3 by solving the instance AV25-1, which is a linear ordering problem since it has unity facility lengths. This special case also has interesting results. Unlike most other cases discussed this far, AKV seems to consistenly outperforms AKV'. This contradiction is considered a special case due to this particular instance at the given parameter settings. Furthermore, the four curves lie closely to each other in the middle range of the graph from `numcut` of 250 to 800. On the other hand, Figure 4.13 shows that Strategy 3

AKV' 2		numcut	250						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.477						
		100.934	103.411	35,960.5					
	-0.4000	16.051	119.463						
		169.863	289.326	36,240.5	37,119.5	280.0	0.779%	879	2.368%
1.01	-0.4040	84.548	373.873						
	159	194.437	568.310	36,315.0	37,119.5	74.5	0.206%	804.5	2.167%
0.50	-0.2040	815.894	1,384.204						
		226.390	1,610.594	36,509.0	37,119.5	194.0	0.534%	610.5	1.645%
1.01	-0.2060	149.145	1,759.739						
		275.459	2,035.198	36,566.5	37,119.5	57.5	0.157%	553	1.490%
1.01	-0.2081	89.138	2,124.336						
		323.842	2,448.178	36,644.5	37,119.5	78.0	0.213%	475	1.280%
1.01	-0.2102	86.802	2,534.980						
		491.154	3,026.134	36,695.0	37,119.5	50.5	0.138%	424.5	1.144%
1.01	-0.2123	85.202	3,111.336						
		522.858	3,634.194	36,768.0	37,119.5	73.0	0.199%	351.5	0.947%
1.01	-0.2144	84.460	3,718.654						
		578.153	4,296.807	36,803.0	37,119.5	35.0	0.095%	316.5	0.853%
0.80	-0.1715	84.538	4,381.346						
		611.576	4,992.921	36,851.5	37,119.5	48.5	0.132%	268	0.722%
1.01	-0.1732	83.573	5,076.495						
		676.622	5,753.116	36,890.5	37,119.5	39.0	0.106%	229	0.617%
0.80	-0.1386	87.062	5,840.178						
		702.274	6,542.452	36,922.0	37,119.5	31.5	0.085%	197.5	0.532%
0.80	-0.1109	90.044	6,632.496						
		765.415	7,397.911	36,951.5	37,119.5	29.5	0.080%	168	0.453%
0.80	-0.0887	143.024	7,540.935						
		843.864	8,384.799	36,984.0	37,119.5	32.5	0.088%	135.5	0.365%
0.80	-0.0710	236.576	8,621.375						
		923.238	9,544.613	37,010.0	37,119.5	26.0	0.070%	109.5	0.295%
0.80	-0.0568	329.383	9,873.996						
		1,023.611	10,897.607	37,028.0	37,119.5	18.0	0.049%	91.5	0.247%
0.80	-0.0454	687.116	11,584.723						
		1,094.311	12,679.034	37,047.0	37,119.5	19.0	0.051%	72.5	0.195%
0.80	-0.0363	1,266.827	13,945.861						
		1,205.878	15,151.739	37,060.0	37,119.5	13.0	0.035%	59.5	0.160%
0.83	-0.0300	1,530.893	16,682.632						
		1,323.950	18,006.583	37,079.5	37,119.5	19.5	0.053%	40	0.108%
1.00	-0.0300	663.013	18,669.596						
		1,451.815	20,121.411	37,089.5	37,119.5	10.0	0.027%	30	0.081%
1.00	-0.0300	392.473	20,513.885						
		1,663.169	22,177.053	37,097.5	37,119.5	8.0	0.022%	22	0.059%
1.00	-0.0300	221.168	22,398.221						
		1,822.371	24,220.592	37,105.5	37,119.5	8.0	0.022%	14	0.038%
1.00	-0.0300	97.179	24,317.771						
		2,004.023	26,321.794	37,111.0	37,119.5	5.5	0.015%	8.5	0.023%
1.00	-0.0300	85.975	26,407.769						
		2,226.434	28,634.203	37,114.5	37,116.5	3.5	0.009%	2	0.005%
1.00	-0.0300	83.825	28,718.027						
		2,509.022	31,227.050	37,115.5	37,116.5	1.0	0.003%	1	0.003%
1.00	-0.0300	83.841	31,310.890						
		2,656.317	33,967.207	37,116.5	37,116.5	1.0	0.003%	0	0.000%
		83.837	34,051.044						

Table 4.3: Computing AV25-2 using AKV'2 with `numcut` = 250

is consistenly faster than Strategy 2 for both AKV and AKV'. But similar to the earlier conclusion, the resultant improvement is small but noticeable.

For the smaller instance, HeKu20, the comparison observation is the same as for AV25-2, i.e. AKV' outperforms AKV and Strategy 3 shows a marginal improve-

AKV' 3		numcut	250						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.691						
		107.650	110.341	35,960					
	-0.4000	16.593	126.934						
		179.762	306.697	36,240	37,120	280.0	0.779%	880	2.371%
1.01	-0.4040	85.252	391.949						
	159	205.714	597.662	36,315	37,120	75.0	0.207%	805	2.169%
0.50	-0.2040	793.263	1,390.925						
		237.129	1,628.054	36,509	37,120	194.0	0.534%	611	1.646%
1.01	-0.2060	147.203	1,775.258						
		293.657	2,068.915	36,566	37,120	57.0	0.156%	554	1.492%
1.01	-0.2081	90.234	2,159.149						
		338.360	2,497.509	36,644	37,120	78.0	0.213%	476	1.282%
1.01	-0.2102	87.850	2,585.359						
		497.659	3,083.017	36,695	37,120	51.0	0.139%	425	1.145%
1.01	-0.2123	85.661	3,168.678						
		538.465	3,707.143	36,768	37,120	73.0	0.199%	352	0.948%
1.01	-0.2144	84.706	3,791.849						
		595.513	4,387.362	36,803	37,120	35.0	0.095%	317	0.854%
0.80	-0.1715	85.075	4,472.437						
		625.079	5,097.517	36,852	37,120	49.0	0.133%	268	0.722%
1.01	-0.1732	84.193	5,181.709						
		702.053	5,883.762	36,890	37,120	38.0	0.103%	230	0.620%
0.80	-0.1386	88.408	5,972.170						
		724.953	6,697.123	36,922	37,120	32.0	0.087%	198	0.533%
0.80	-0.1109	91.862	6,788.985						
		795.689	7,584.674	36,952	37,120	30.0	0.081%	168	0.453%
0.80	-0.0887	142.300	7,726.974						
		872.244	8,599.218	36,984	37,120	32.0	0.087%	136	0.366%
0.80	-0.0710	237.398	8,836.616						
		947.026	9,783.642	37,010	37,120	26.0	0.070%	110	0.296%
0.80	-0.0568	320.442	10,104.084						
		1,054.351	11,158.435	37,028	37,120	18.0	0.049%	92	0.248%
0.80	-0.0454	681.091	11,839.525						
		1,127.205	12,966.731	37,047	37,120	19.0	0.051%	73	0.197%
1.00	-0.0454	519.953	13,486.684						
		1,242.723	14,729.407	37,060	37,120	13.0	0.035%	60	0.162%
1.00	-0.0454	277.347	15,006.754						
		1,357.177	16,363.931	37,080	37,120	20.0	0.054%	40	0.108%
1.00	-0.0454	126.171	16,490.103						
		1,475.680	17,965.782	37,090	37,120	10.0	0.027%	30	0.081%
1.00	-0.0454	98.500	18,064.283						
		1,682.289	19,746.571	37,098	37,120	8.0	0.022%	22	0.059%
1.00	-0.0454	87.602	19,834.173						
		1,837.343	21,671.515	37,106	37,120	8.0	0.022%	14	0.038%
1.00	-0.0454	87.260	21,758.775						
		2,007.423	23,766.199	37,111	37,120	5.0	0.013%	9	0.024%
1.00	-0.0454	85.886	23,852.084						
		2,234.956	26,087.040	37,114	37,116	3.0	0.008%	2	0.005%
1.00	-0.0454	84.609	26,171.649						
	72	2,333.478	28,505.127	37,115	37,116	1.0	0.003%	1	0.003%
0.25	-0.0114	225.119	28,730.247						
		2,752.895	31,483.142	37,116	37,116	1.0	0.003%	0	0.000%
		88.825	31,571.967						

Table 4.4: Computing AV25-2 using AKV'3 with `numcut` = 250

ment. However, it is worth noticing is that unlike for other larger instances, the cutting plane process runs faster at lower `numcut`. This is because, as explained earlier in Section 4.1, the higher the `numcut`, the more rapidly the size of the SDP problem grows, and hence the faster the growth in optimization time. This phe-

AKV 2		numcut	700						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			1.638						
		59.550	61.188	3,655.5					
	-0.4000	15.331	76.519						
		294.876	371.395	36,560.0	37,130.5	32,904.5	900.137%	570.5	1.536%
1.01	-0.4040	83.290	454.685						
	83	307.139	761.825	36,597.0	37,130.5	37.0	0.101%	533.5	1.437%
0.50	-0.2040	1,156.528	1,918.353						
		583.731	2,502.083	36,698.5	37,166.5	101.5	0.277%	468	1.259%
1.01	-0.2060	129.091	2,631.175						
		861.045	3,492.220	36,817.5	37,166.5	119.0	0.324%	349	0.939%
1.01	-0.2081	91.577	3,583.797						
		1,288.058	4,871.855	36,852.0	37,166.5	34.5	0.094%	314.5	0.846%
0.80	-0.1665	157.057	5,028.912						
		1,823.966	6,852.878	36,935.0	37,166.5	83.0	0.225%	231.5	0.623%
1.01	-0.1681	85.905	6,938.783						
		2,808.047	9,746.830	36,946.5	37,166.5	11.5	0.031%	220	0.592%
0.80	-0.1345	88.979	9,835.810						
		3,508.265	13,344.075	36,996.0	37,166.5	49.5	0.134%	170.5	0.459%
1.01	-0.1359	86.244	13,430.319						
		4,417.566	17,847.885	37,042.5	37,166.5	46.5	0.126%	124	0.334%
0.80	-0.1087	86.883	17,934.768						
		5,698.862	23,633.630	37,066.0	37,166.5	23.5	0.063%	100.5	0.270%
0.80	-0.0870	89.154	23,722.784						
		6,924.772	30,647.557	37,090.0	37,166.5	24.0	0.065%	76.5	0.206%
0.80	-0.0696	86.509	30,734.065						
		8,308.746	39,042.812	37,108.5	37,166.5	18.5	0.050%	58	0.156%
0.80	-0.0556	87.077	39,129.889						
		9,976.786	49,106.675	37,114.0	37,166.5	5.5	0.015%	52.5	0.141%
0.80	-0.0445	86.710	49,193.384						
		11,774.874	60,968.258	37,116.5	37,166.5	2.5	0.007%	50	0.135%
		85.881	61,054.140						

Table 4.5: Computing AV25-2 using AKV2 with `numcut` = 700

nomenon can be observed by the two labeled data points on Figure 4.14. Tables 4.7 and 4.8 show the time breakdown of the two labeled points. Table 4.7 indicates that AKV2 at `numcut` = 100 has four more iterations than AKV2 at `numcut` = 900, but its total computing time is only 18% of the trial with `numcut` = 900. This is because for a smaller instance such as HeKu20, the number of iterations required to close the gap is much smaller and the process of finding and sorting the cuts is less complicated. Therefore, although requiring more iterations to complete, the trial with smaller `numcut` is still faster than the trial with higher `numcut`.

There are two missing data points for AKV3, which means that there are two incomplete trials. This shows another weakness in Strategy 3. When the gap between Z_{bk} and Z_{lb} is small, `vioRHS` stays unchanged, which occasionaly becomes too high in the cutting plane process. The algorithm therefore thinks that it runs out of cuts and exits the cutting plane algorithm. This limitation is corrected in Strategy 5.

AKV 3		numcut	700						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			1.623						
		58.247	59.870	36,355.5					
	-0.4000	15.154	75.024						
		292.511	367.535	36,560.0	37,130.5	204.5	0.563%	570.5	1.536%
1.01	-0.4040	82.961	450.496						
	83	303.984	754.480	36,597.0	37,130.5	37.0	0.101%	533.5	1.437%
0.50	-0.2040	1,152.826	1,907.306						
		579.796	2,487.102	36,698.5	37,116.5	101.5	0.277%	418	1.126%
1.01	-0.2060	128.678	2,615.780						
		855.748	3,471.529	36,817.5	37,116.5	119.0	0.324%	299	0.806%
1.01	-0.2081	88.626	3,560.154						
		1,279.994	4,840.148	36,852.0	37,116.5	34.5	0.094%	264.5	0.713%
0.80	-0.1665	152.446	4,992.594						
		1,812.595	6,805.189	36,935.0	37,116.5	83.0	0.225%	181.5	0.489%
1.01	-0.1682	85.789	6,890.979						
		2,793.265	9,684.243	36,946.5	37,116.5	11.5	0.031%	170	0.458%
0.80	-0.1345	88.076	9,772.319						
		3,477.711	13,250.031	36,996.0	37,116.5	49.5	0.134%	120.5	0.325%
1.01	-0.1359	86.434	13,336.465						
		4,377.071	17,713.536	37,042.5	37,116.5	46.5	0.126%	74	0.199%
1.00	-0.1359	86.093	17,799.629						
		5,680.520	23,480.149	37,066.0	37,116.5	23.5	0.063%	50.5	0.136%
1.00	-0.1359	84.539	23,564.688						
	46	5,671.135	29,235.823	37,072.5	37,116.5	6.5	0.018%	44	0.119%
0.25	-0.033965	3,248.860	32,484.683						
		7,119.053	39,603.736	37,091.5	37,116.5	19.0	0.051%	25	0.067%
1.00	-0.033965	750.790	40,354.527						
		8,520.886	48,875.413	37,108.0	37,116.5	16.5	0.044%	8.5	0.023%
1.00	-0.033965	100.369	48,975.782						
		10,238.321	59,214.103	37,112.5	37,116.5	4.5	0.012%	4	0.011%
1.00	-0.033965	88.217	59,302.320						
		12,589.062	71,891.382	37,116.5	37,116.5	4.0	0.011%	0	0.000%
		79.627	71,971.009						

Table 4.6: Computing AV25-2 using AKV3 with `numcut` = 700

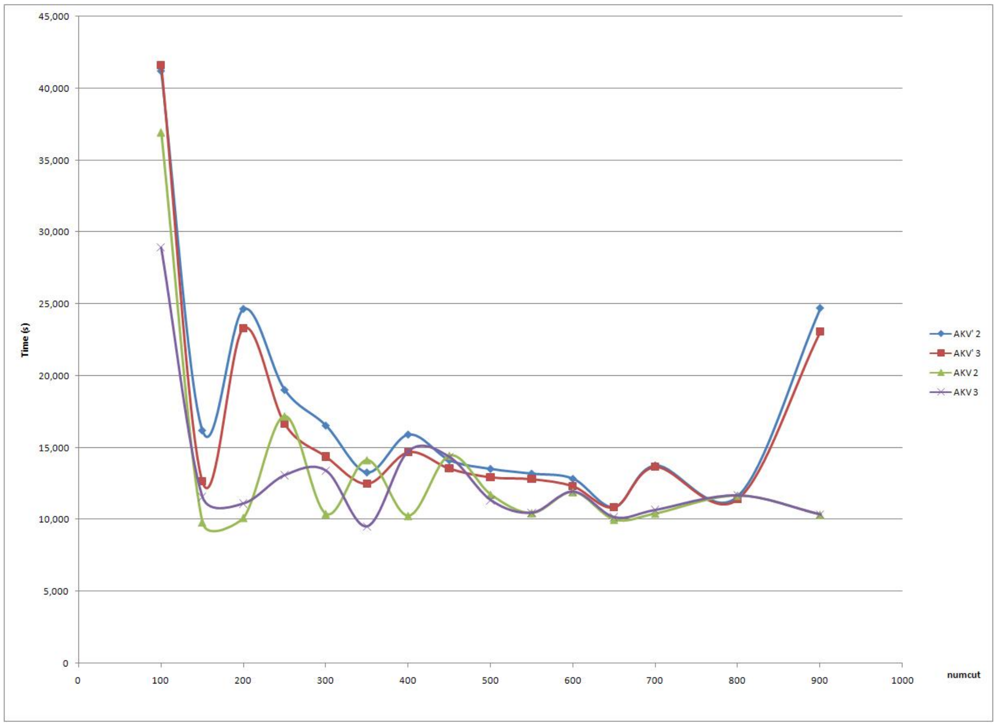

Figure 4.13: Comparison of Strategy 2 and Strategy 3 for AV25-1

Figure 4.14: Comparison of Strategy 2 and Strategy 3 for HeKu20

AKV 2		numcut	100						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			1.172						
		15.044	16.216	15,286.0					
	-0.4	72.854	89.070						
		24.210	113.280	15,358.0	15,549.0	72.0	0.471%	191.0	1.228%
1.01	-0.404	18.693	131.974						
		32.769	164.743	15,376.0	15,549.0	18.0	0.117%	173.0	1.113%
0.80	-0.3232	28.863	193.606						
		35.679	229.285	15,413.0	15,549.0	37.0	0.241%	136.0	0.875%
1.01	-0.3264	19.961	249.245						
		40.137	289.382	15,454.0	15,549.0	41.0	0.266%	95.0	0.611%
1.01	-0.3297	19.100	308.482						
		46.532	355.015	15,498.0	15,549.0	44.0	0.285%	51.0	0.328%
1.01	-0.333	18.312	373.326						
		55.831	429.157	15,506.0	15,549.0	8.0	0.052%	43.0	0.277%
0.80	-0.2664	18.508	447.665						
		62.260	509.925	15,521.0	15,549.0	15.0	0.097%	28.0	0.180%
0.80	-0.2131	18.164	528.090						
		75.769	603.859	15,536.0	15,549.0	15.0	0.097%	13.0	0.084%
0.80	-0.1705	18.273	622.132						
		88.716	710.848	15,543.0	15,549.0	7.0	0.045%	6.0	0.039%
0.80	-0.1364	18.242	729.090						
		100.609	829.699	15,546.0	15,549.0	3.0	0.019%	3.0	0.019%
0.80	-0.1091	18.320	848.019						
		117.765	965.784	15,549.0	15,549.0	3.0	0.019%	0.0	0.000%
		20.153	985.937						

Table 4.7: Computing HeKu20 using AKV2 with `numcut` = 100

AKV 2		numcut	900						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			0.942						
		14.498	15.439	15,286.0					
	-0.4	57.474	72.913						
		117.473	190.386	15,390.5	15,549.0	104.5	0.684%	158.5	1.019%
1.01	-0.404	18.932	209.318						
		302.854	512.172	15,435.0	15,549.0	44.5	0.289%	114.0	0.733%
1.01	-0.408	18.520	530.692						
	1	297.216	827.908	15,440.0	15,549.0	5.0	0.032%	109.0	0.701%
0.51	-0.20804	88.664	916.572						
		556.459	1,473.031	15,507.0	15,549.0	67.0	0.434%	42.0	0.270%
1.01	-0.2101	18.906	1,491.937						
		972.077	2,464.014	15,540.0	15,549.0	33.0	0.213%	9.0	0.058%
1.01	-0.2122	18.808	2,482.822						
	51	1,093.351	3,576.172	15,542.5	15,549.0	2.5	0.016%	6.5	0.042%
0.25	-0.0531	341.720	3,917.893						
		1,661.544	5,579.436	15,549.0	15,549.0	6.5	0.042%	0.0	0.000%
		26.844	5,606.280						

Table 4.8: Computing HeKu20 using AKV2 with `numcut` = 900

4.2.3 From Strategy 3 to Strategy 4

In Strategy 4, the feature of removing non-binding constraints is added, which results in some satisfactory improvement. Figure 4.15 illustrates that Strategy 4 outperforms Strategy 3 most of the time. As explained in Section 4.1.4, removing the non-binding constraints keeps the problem size small and allows more violated constraints to be added in the next iteration. This number is called `numslack`, which is calculated at the end of each iteration before starting to find new cuts for the next round. This approach removes the less significant constraints and adds the more important ones, which speeds up the pace of lower bound improvement. This phenomenon can be observed in the two labeled data points, which are detailed in Tables 4.9 and 4.10.

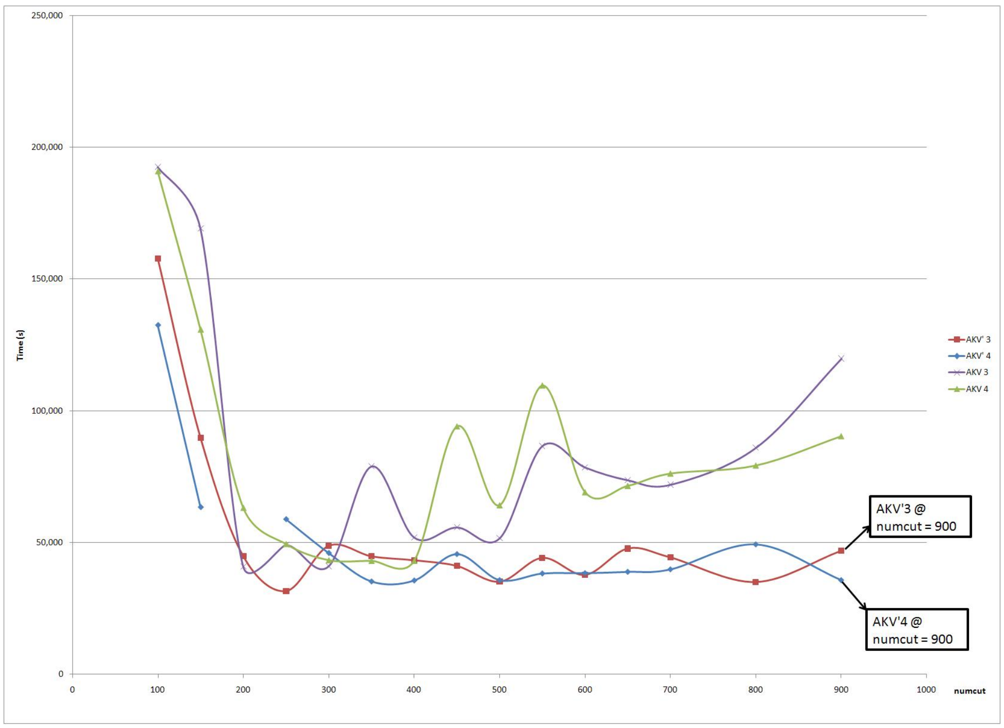

Figure 4.15: Comparison of Strategy 3 and Strategy 4 for AV25-2

The number that is placed between the gap values of the new and old Z_{lb} in the seventh column of Table 4.10 is the `numslack` for each iteration. The value of `numslack` generally decreases as the gap becomes smaller. Another fact worth of notice is circled in both tables. The circled Z_{lb} marks the point at which the effect of constraints removal becomes obvious. Starting at this point, AKV'4 improves the Z_{lb} more rapidly, and consequently finishes the computation in fewer iterations.

The arrangement of the four curves for instance AV25-1 starts to be less distinguishable in Figure 4.16. The four curves look as if they are interlaced throughout the various `numcut` values. Figure 4.16 also shows that Strategy 4 does better than Strategy 3 all the way through the middle to the ending range of `numcut`.

AKV' 3		numcut	900						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.314						
		96.645	98.959	35,960.5					
	-0.4000	15.662	114.621						
		345.149	459.770	36,325.5	37,154.5	365.0	1.015%	829	2.231%
1.01	-0.4040	84.311	544.081						
	52	345.161	889.242	36,365.5	37,154.5	40.0	0.110%	789	2.124%
0.50	-0.2040	644.042	1,533.284						
		516.767	2,050.051	36,591.5	37,130.5	226.0	0.621%	539	1.452%
1.01	-0.2060	105.264	2,155.315						
		752.354	2,907.669	36,723.0	37,130.5	131.5	0.359%	407.5	1.097%
1.01	-0.2081	86.596	2,994.265						
		1,031.393	4,025.659	36,803.5	37,119.5	80.5	0.219%	316	0.851%
1.01	-0.2102	84.865	4,110.523						
	761	1,387.710	5,498.233	36,914.5	37,119.5	111.0	0.302%	205	0.552%
0.25	-0.0525	2,492.707	7,990.940						
		1,985.577	9,976.518	36,974.0	37,119.5	59.5	0.161%	145.5	0.392%
1.01	-0.0531	1,020.877	10,997.395						
		2,718.270	13,715.665	37,022.5	37,119.5	48.5	0.131%	97	0.261%
1.01	-0.0536	461.747	14,177.412						
		3,585.985	17,763.398	37,064.0	37,119.5	41.5	0.112%	55.5	0.150%
1.00	-0.0536	138.650	17,902.048						
		4,848.069	22,750.117	37,093.0	37,119.5	29.0	0.078%	26.5	0.071%
1.00	-0.0536	88.083	22,838.200						
		6,485.045	29,323.246	37,110.5	37,116.5	17.5	0.047%	6	0.016%
1.00	-0.0536	87.303	29,410.549						
	500	7,421.006	36,831.555	37,114.0	37,116.5	3.5	0.009%	2.5	0.007%
0.25	-0.0134	452.257	37,283.812						
		9,555.527	46,839.339	37,116.5	37,116.5	2.5	0.007%	0	0.000%
		79.990	46,919.329						

Table 4.9: Computing AV25-2 using AKV'3 with `numcut` = 900

AKV' 4		numcut	900						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.667						
		107.612	110.279	35,960.5	37,172.5				
	-0.4000	88.228	198.507						
		363.806	562.313	36,325.5	37,154.5	365.0	1.015%	829	2.231%
1.01	-0.4040	86.365	648.678			239			
	52	254.447	903.124	36,365.5	37,154.5	40.0	0.110%	789	2.124%
0.50	-0.2040	668.649	1,571.773			21			
		513.765	2,085.538	36,593.5	37,154.5	228.0	0.627%	561	1.510%
1.01	-0.2060	110.876	2,196.414			142			
		747.419	2,943.833	36,725.5	37,154.5	132.0	0.361%	429	1.155%
1.01	-0.2081	89.623	3,033.456			374			
		1,014.314	4,047.770	36,821.5	37,137.5	96.0	0.261%	316	0.851%
1.01	-0.2102	85.687	4,133.457			349			
	761	1,172.154	5,305.611	36,911.5	37,130.5	90.0	0.244%	219	0.590%
0.25	-0.0525	2,976.978	8,282.589			288			
		1,571.653	9,854.242	36,994.5	37,130.5	83.0	0.225%	136	0.366%
1.01	-0.0531	943.187	10,797.429			172			
		2,242.498	13,039.927	37,023.0	37,119.5	28.5	0.077%	96.5	0.260%
0.80	-0.0425	1,307.548	14,347.475			412			
		3,055.361	17,402.835	37,077.5	37,119.5	54.5	0.147%	42	0.113%
1.00	-0.0425	171.847	17,574.683			117			
		4,215.526	21,790.209	37,094.0	37,119.5	16.5	0.045%	25.5	0.069%
1.00	-0.0425	118.642	21,908.850			37			
		5,516.688	27,425.539	37,111.0	37,116.5	17.0	0.046%	5.5	0.015%
1.00	-0.0425	90.552	27,516.091			3			
		8,123.452	35,639.543	37,116.5	37,116.5	5.5	0.015%	0	0.000%
		81.210	35,720.752						

Table 4.10: Computing AV25-2 using AKV'4 with `numcut` = 900

Figure 4.17 compares AKV3, AKV4, AKV'3, and AKV'4 for the smaller instance HeKu20. Even though there are one missing point for both AKV4 and AKV'4 and two for AKV3, the graph still shows that Strategy 4 outperforms Strategy 3. Also, AKV' is consistently observed to be a better model for this instance.

One final note about the new feature in Strategy 4 is that although it does not seem to offer significant reduction in computation time, it is expected to be crucial for solving large instances. Experience has shown that the algorithm can run out of memory for certain difficult large instances. Therefore, keeping the problem size small plays an important role in the next strategies.

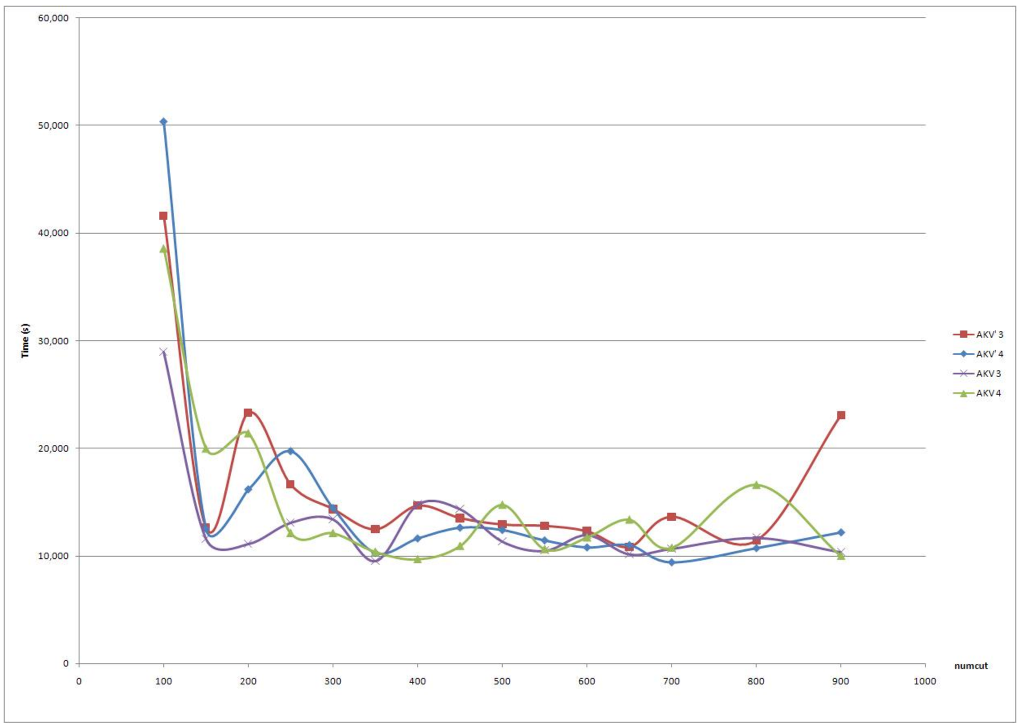

Figure 4.16: Comparison of Strategy 3 and Strategy 4 for AV25-1

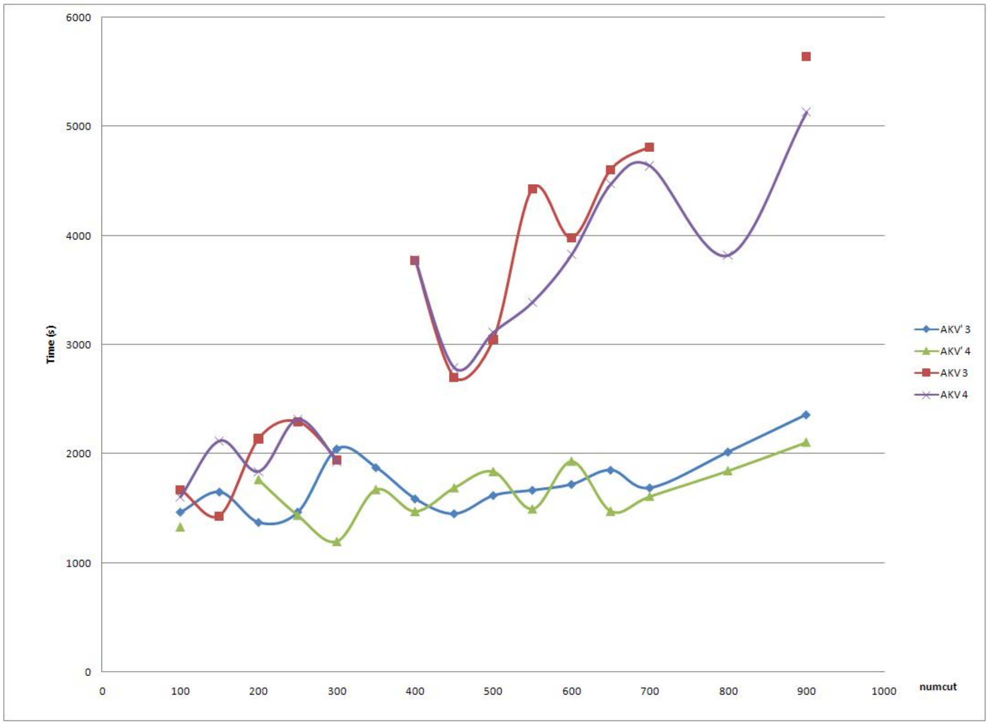

Figure 4.17: Comparison of Strategy 3 and Strategy 4 for HeKu20

4.2.4 From Strategy 4 to Strategy 5

Strategy 5 includes two important new features:

- If number of cuts found $\leq \frac{1}{2}$ `numcut`, lower `vioRHS` to search for new violations.
- If no violations are found, lower `vioRHS` unless it has reached a very point of -0.001.

These new features ensure the computation will not terminate prematurely and prevent the algorithm from wasting an iteration when only a few cuts are found. It is evident that these new features are successful since there is no more missing data point in the comparison graph; see Figure 4.18. This means that premature termination is now successfully avoided.

Figure 4.18 compares AKV4, AKV5, AKV'4, and AKV'5. It has evidently shown that AKV' outperforms AKV in general. Furthermore, AKV'5 is clearly the

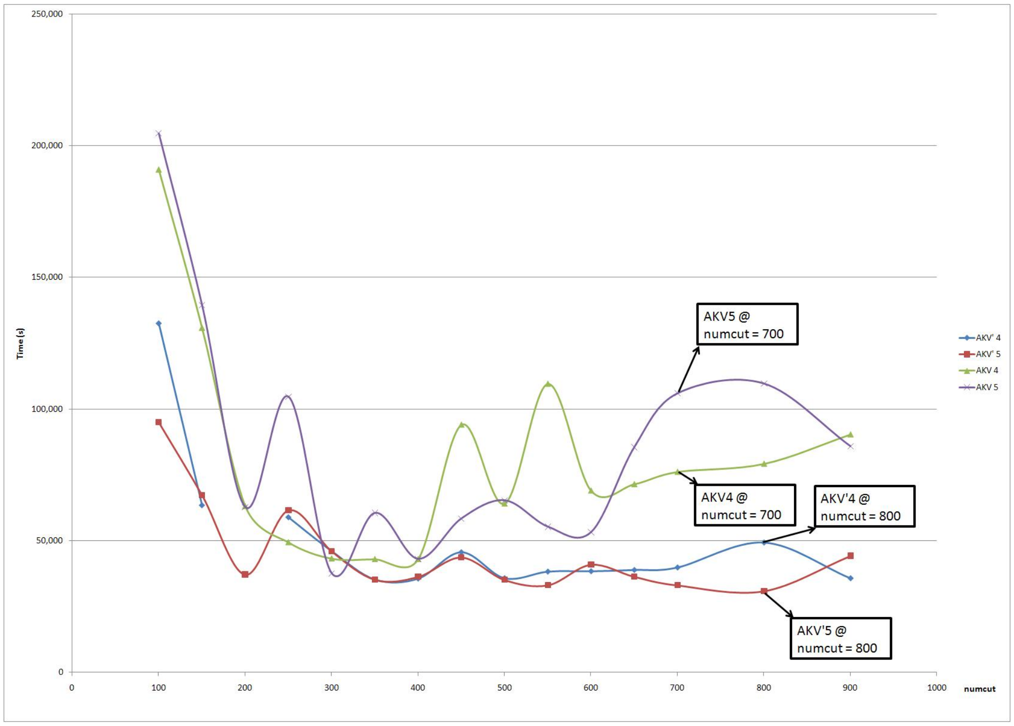

Figure 4.18: Comparison of Strategy 4 and Strategy 5 for AV25-2

overall best performing combination, which also yields the lowest computing time thus far at `numcut` = 800. In addition, AKV'5 produces the lowest small-`numcut` run time at `numcut` = 100, which has always been way above 130,000 seconds in the past. The first two labeled data point of AKV'4 and AKV'5 at `numcut` = 800, which are broken down in Tables 4.11 and 4.12, illustrate how Strategy 5 achieves a lower computation time.

The shaded clock time in Table 4.12 signifies the activation of the new feature that re-searches for new violations when the number of cuts found is less than half of `numcut`. This new feature is used twice in AKV'5 at `numcut` = 800. The first time occurs at the third iteration when the algorithm only found 40 cuts, so it lowered `vioRHS` from -0.4040 (not shown) to -0.2040 and began another search for new cuts. Because a small "detour" was taken, the time duration for finding and sorting the cuts is slightly longer than how it would normally take. Since the number of total constraints added after re-search for AKV'5 is much higher than the 40 new constraints in AKV'4, the respective optimization timee is also longer for the sub problem becomes bigger. However, because Strategy 5 takes action to

AKV' 4		numcut	800						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.159						
		104.822	106.981	35,960.5	37,172.5				
	-0.4000	87.883	194.864						
		323.098	517.961	36,317.5	37,154.5	357.0	0.993%	837	2.253%
1.01	-0.4040	85.550	603.511			197			
	40	236.365	839.876	36,355.5	37,133.5	38.0	0.105%	778	2.095%
0.50	-0.2040	724.941	1,564.817			19			
		479.864	2,044.681	36,565.5	37,133.5	210.0	0.578%	568	1.530%
1.01	-0.2060	105.970	2,150.651			142			
		680.532	2,831.183	36,697.0	37,133.5	131.5	0.360%	436.5	1.175%
1.01	-0.2081	86.154	2,917.337			217			
		961.329	3,878.667	36,803.5	37,133.5	106.5	0.290%	330	0.889%
1.01	-0.2102	84.866	3,963.533			308			
		1,367.208	5,330.741	36,878.0	37,133.5	74.5	0.202%	255.5	0.688%
1.01	-0.2123	86.643	5,417.384			133			
	144	1,342.239	6,759.622	36,913.0	37,133.5	35.0	0.095%	220.5	0.594%
0.25	-0.0531	2,180.982	8,940.605			95			
		1,769.965	10,710.570	36,985.5	37,133.5	72.5	0.196%	148	0.399%
1.01	-0.0536	897.552	11,608.121			95			
		2,435.072	14,043.193	37,025.5	37,133.5	40.0	0.108%	108	0.291%
0.80	-0.0429	1,440.323	15,483.516			89			
		3,032.451	18,515.967	37,068.5	37,133.5	43.0	0.116%	65	0.175%
1.00	-0.0429	224.226	18,740.193			55			
		3,953.298	22,693.491	37,087.0	37,133.5	18.5	0.050%	46.5	0.125%
1.00	-0.0429	123.757	22,817.247			29			
		4,962.019	27,779.267	37,111.0	37,116.5	24.0	0.065%	5.5	0.015%
1.00	-0.0429	89.987	27,869.254			3			
		6,470.148	34,339.402	37,116.0	37,116.5	5.0	0.013%	0.5	0.001%
1.00	-0.0429	86.860	34426.262			0			
	84	6,837.350	41,263.612	37,116.0	37,116.5	0.0	0.000%	0.5	0.001%
0.25	-0.0107	114.767	41378.379			0			
		7,825.056	49,203.435	37,116.5	37,116.5	0.5	0.001%	0	0.000%
		87.185	49,290.620						

Table 4.11: Computing AV25-2 using AKV'4 with `numcut` = 800

lower `vioRHS` earlier than Strategy 4, it saves one iteration to achieve a similar Z_{lb}. For instance, AKV'5 requires 4 less iterations than AKV'4 at this given `numcut`. The circled time intervals in Tables 4.11 and 4.12 show the impact of saving an extra iteration. AKV'4 in Table 4.11 requires 2 iterations, which is composed of 2 cut-searching steps and 2 optimization steps, to sufficiently lower `vioRHS` to improve its Z_{lb} and close the gap. Meanwhile, AKV'5 in Table 4.12 can sufficiently improve its Z_{lb} by one iteration. This savings in time is more impactful toward the end of the cutting plane process when the gap is small because by then the sub-problem with many added constraints has grown much bigger, which means each optimization step becomes very time-consuming.

Tables 4.11 and 4.12 present the successful case of Strategy 5, while Tables 4.13 and 4.14 show the weaker aspect of Strategy 5. Tables 4.13 and 4.14 are the other two labeled data points in Figure 4.18. The two tables show that although the

AKV' 5		numcut	800						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.655						
		101.981	104.636	35,960.5	37,172.5				
	-0.4000	87.264	191.900						
	40	326.059	517.959	36,317.5	37,154.5	357.0	0.993%	837	2.253%
0.51	-0.2040	1,316.412	1,834.371			197			
		550.183	2,384.554	36,552.0	37,154.5	234.5	0.646%	602.5	1.622%
1.01	-0.2060	134.929	2,519.483			219			
		665.908	3,185.391	36,702.5	37,151.5	150.5	0.412%	449	1.209%
1.01	-0.2081	87.659	3,273.051			284			
		1,015.228	4,288.278	36,793.0	37,151.5	90.5	0.247%	358.5	0.965%
1.01	-0.2102	84.708	4,372.987			458			
	1,019	1,106.688	5,479.675	36,926.5	37,130.5	133.5	0.363%	204	0.549%
0.25	-0.0525	2,252.944	7,732.618			206			
		1,512.678	9,245.296	36,990.0	37,130.5	63.5	0.172%	140.5	0.378%
1.01	-0.0531	870.147	10,115.443			225			
		1,954.074	12,069.517	37,042.0	37,130.5	52.0	0.141%	88.5	0.238%
1.01	-0.0536	347.032	12,416.549			113			
		2,583.046	14,999.595	37,067.5	37,130.5	25.5	0.069%	63	0.170%
1.00	-0.0536	138.982	15,138.577			57			
		3,511.091	18,649.667	37,098.0	37,130.5	30.5	0.082%	32.5	0.088%
1.00	-0.0536	88.214	18,737.881			14			
	169	4,579.723	23,317.604	37,110.0	37,116.5	12.0	0.032%	6.5	0.018%
0.25	-0.0134	1,148.714	24,466.318			11			
		6,337.763	30,804.080	37,116.5	37,130.5	6.5	0.018%	14	0.038%
		94.118	30,898.198						

Table 4.12: Computing AV25-2 using AKV'5 with `numcut` = 800

time saved in reducing one iteration is significant as circled in Table 4.13, the overly aggressive drop in `vioRHS` in AKV5 results in even longer cut-searching time as circled in Table 4.14. This weakness gives a warning that the modification to `vioRHS` will have to be changed to smooth out the reduction process.

Figure 4.19 compares Strategies 4 and 5 for solving AV25-1. The performance of the four combinations become even more indistinct as the four curves lie closely to each other in the graph. Although it may seem difficult to conclude that AKV' outperforms AKV and that Strategy 5 is better than Strategy 4 based on this graph, Strategy 5 has for certain yielded the shortest computation time thus far for low `numcut` such as at 100. This run time is only 1/3 of the run time for AKV'4 at `numcut` = 100.

For the smaller instance HeKu20 in Figure 4.20 we can derive a conclusion that is similar to AV25-2. AKV'5 is generally the best-performing combination out of the four. However, there are a few exception cases, such as the circled data point, which are summarized in Tables 4.15 and 4.16. The two tables show that although the new feature in Strategy 5 allows AKV5 to finish in one less iteration

AKV 4		numcut	700						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			1.673						
		60.485	62.158	36,355.5	37,137.5				
	-0.4000	83.745	145.904						
		292.320	438.223	36,560.0	37,130.5	204.5	0.563%	570.5	1.536%
1.01	-0.4040	83.447	521.670			22			
	83	280.210	801.880	36,597.0	37,130.5	37.0	0.101%	533.5	1.437%
0.50	-0.2040	1,171.036	1,972.915			11			
		548.786	2,521.701	36,698.5	37,116.5	101.5	0.277%	418	1.126%
1.01	-0.2060	132.499	2,654.200			30			
		860.445	3,514.645	36,819.0	37,116.5	120.5	0.328%	297.5	0.802%
1.01	-0.2081	91.409	3,606.054			25			
		1,330.854	4,936.908	36,852.5	37,116.5	33.5	0.091%	264	0.711%
0.80	-0.1665	156.888	5,093.796			61			
		1,791.007	6,884.803	36,935.0	37,116.5	82.5	0.224%	181.5	0.489%
1.01	-0.1681	84.867	6,969.670			62			
		2,633.996	9,603.667	36,938.5	37,116.5	3.5	0.009%	178	0.480%
0.80	-0.1345	88.205	9,691.872			16			
		3,454.663	13,146.535	36,993.5	37,116.5	55.0	0.149%	123	0.331%
1.01	-0.1359	85.968	13,232.503			81			
		4,584.192	17,816.695	37,014.0	37,116.5	20.5	0.055%	102.5	0.276%
0.80	-0.1087	89.344	17,906.039			327			
		5,557.824	23,463.863	37,053.0	37,116.5	39.0	0.105%	63.5	0.171%
1.00	-0.1087	85.526	23,549.389			61			
		6,982.790	30,532.179	37,078.0	37,116.5	25.0	0.067%	38.5	0.104%
1.00	-0.1087	85.733	30,617.912			40			
	414	7,918.787	38,536.699	37,091.0	37,116.5	13.0	0.035%	25.5	0.069%
0.25	-0.0272	2,864.198	41,400.897			10			
		9,388.035	50,788.932	37,110.0	37,116.5	19.0	0.051%	6.5	0.018%
1.00	-0.0272	126.856	50,915.789			1			
		11,137.655	62,053.444	37,114.0	37,116.5	4.0	0.011%	2.5	0.007%
1.00	-0.0272	92.204	62,145.648			12			
		13,954.358	76,100.006	37,116.5	37,116.5	2.5	0.007%	0	0.000%
		96.081	76,196.087						

Table 4.13: Computing AV25-2 using AKV4 with `numcut` = 700

than AKV4, AKV5 still takes longer to reach optimality. As seen in the two circled optimization time intervals in Table 4.15, the optimization time of AKV5 in the last two iterations become too time-consuming due to containing more constraints than in AKV4. As a result, the sub-problem becomes too large for this small instance, and consequently the time saved in running fewer iterations for AKV5 cannot even pay off the substantial increase in optimization time.

AKV 5		numcut	700						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			1.667						
		60.528	62.195	36,355.5	37,137.5				
	-0.4000	84.009	146.204						
	83	301.380	447.584	36,560.0	37,130.5	204.5	0.563%	570.5	1.536%
0.51	-0.2040	2,989.616	3,437.200			22			
		523.291	3,960.491	36,696.0	37,130.5	136.0	0.372%	434.5	1.170%
1.01	-0.2060	156.863	4,117.354			55			
		818.758	4,936.112	36,790.5	37,130.5	94.5	0.258%	340	0.916%
1.01	-0.2081	100.543	5,036.655			86			
		1,251.143	6,287.798	36,856.5	37,130.5	66.0	0.179%	274	0.738%
1.01	-0.2102	84.101	6,371.899			75			
	204	1,687.200	8,059.099	36,928.0	37,130.5	71.5	0.194%	202.5	0.545%
0.25	-0.0531	7,938.256	15,997.355			185			
		2,320.204	18,317.559	36,989.5	37,119.5	61.5	0.167%	130	0.350%
1.01	-0.0536	3,266.457	21,584.016			57			
		3,630.029	25,214.045	36,999.0	37,119.5	9.5	0.026%	120.5	0.325%
0.80	-0.0429	5,974.219	31,188.264			93			
		4,518.300	35,706.564	37,040.0	37,119.5	41.0	0.111%	79.5	0.214%
0.80	-0.0343	6,552.204	42,258.768			51			
		5,608.162	47,866.930	37,062.5	37,119.5	22.5	0.061%	57	0.154%
1.00	-0.0343	3,598.114	51,465.044			68			
		7,126.283	58,591.327	37,086.0	37,119.5	23.5	0.063%	33.5	0.090%
1.00	-0.0343	1,258.184	59,849.511			23			
		8,478.388	68,327.899	37,105.5	37,116.5	19.5	0.053%	11	0.030%
1.00	-0.0343	99.725	68,427.624			5			
		10,179.379	78,607.003	37,111.5	37,116.5	6.0	0.016%	5	0.013%
1.00	-0.0343	90.460	78,697.463			1			
		12,383.459	91,080.922	37,116.0	37,116.5	4.5	0.012%	0.5	0.001%
1.00	-0.0343	86.809	91,167.731			0			
		14,730.038	105,897.769	37,116.5	37,116.5	0.5	0.001%	0	0.000%
		83.100	105,980.870						

Table 4.14: Computing AV25-2 using AKV5 with `numcut` = 700

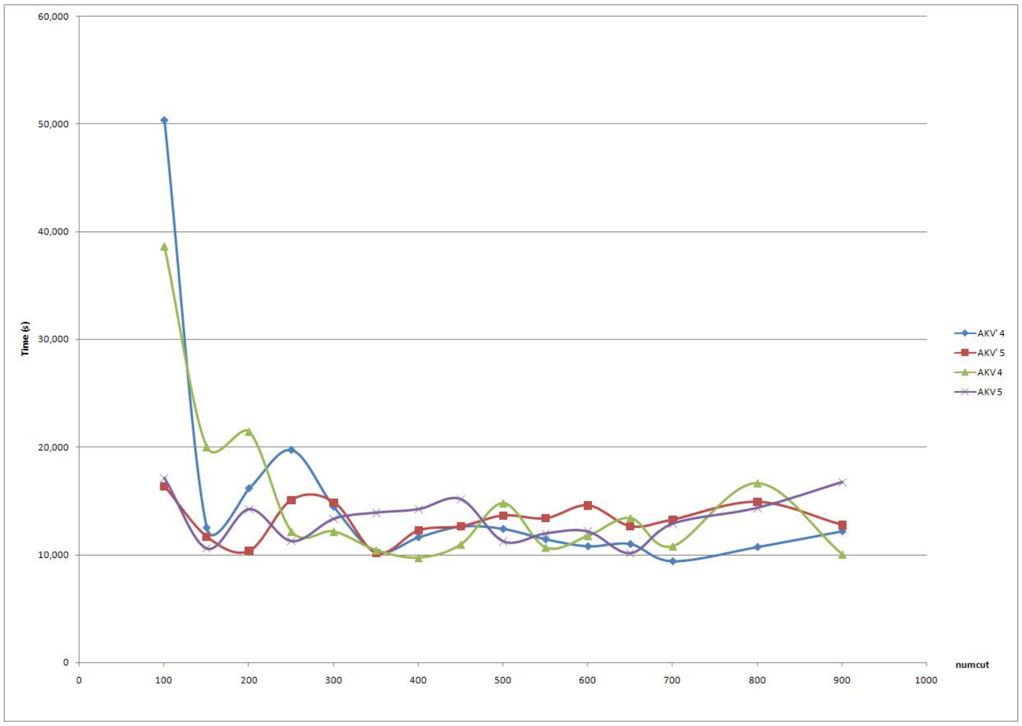

Figure 4.19: Comparison of Strategy 4 and Strategy 5 for AV25-1

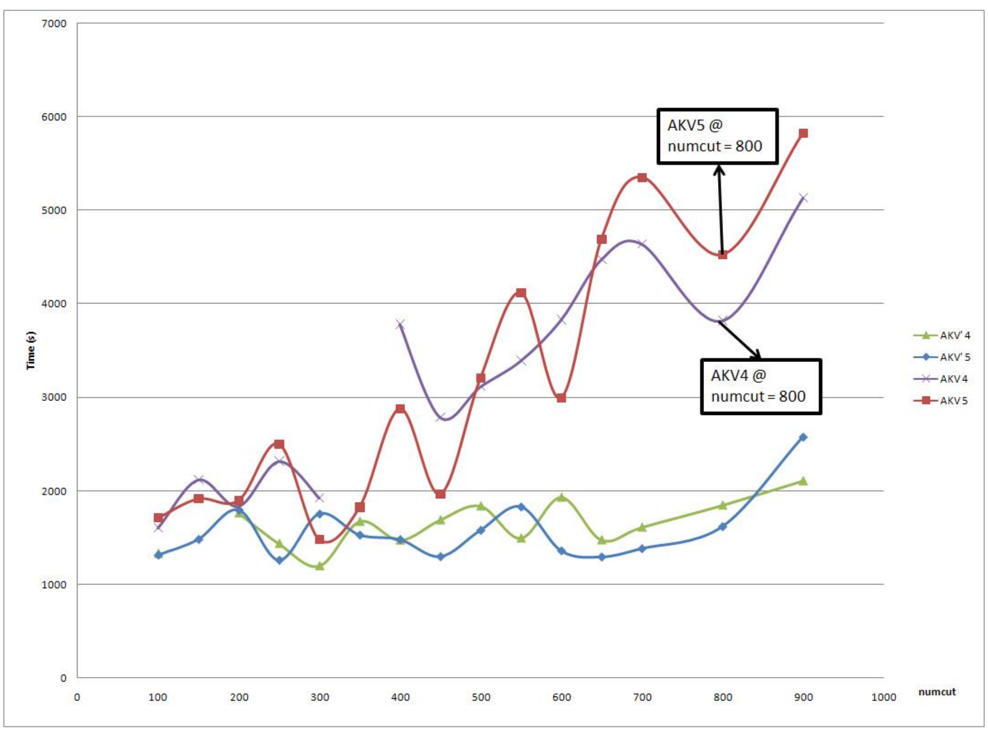

Figure 4.20: Comparison of Strategy 4 and Strategy 5 for HeKu20

AKV 4		numcut	800						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			0.523						
		15.027	15.550	15,286.0	15,549.0				
	-0.4000	89.772	105.321						
		100.302	205.624	15,380.0	15,549.0	94.0	0.615%	169.0	1.087%
1.01	-0.4040	18.513	224.137			27			
		235.323	459.459	15,432.0	15,549.0	52.0	0.338%	117.0	0.752%
1.01	-0.4080	18.494	477.953			22			
	2	236.752	714.706	15,439.0	15,549.0	7.0	0.045%	110.0	0.707%
0.51	-0.2080	92.472	807.178			0			
		411.271	1,218.448	15,498.0	15,549.0	59.0	0.382%	51.0	0.328%
1.01	-0.2101	18.756	1,237.204			145			
		685.503	1,922.708	15,543.0	15,549.0	45.0	0.290%	6.0	0.039%
1.00	-0.2101	18.719	1,941.427			137			
	5	631.768	2,573.195	15,544.0	15,549.0	1.0	0.006%	5.0	0.032%
0.25	-0.0525	238.052	2,811.246			9			
		990.668	3,801.915	15,549.0	15,549.0	5.0	0.032%	0.0	0.000%
		16.617	3,818.532						

Table 4.15: Computing HeKu20 using AKV4 with `numcut` = 800

AKV 5		numcut	800						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			0.896						
		14.310	15.206	15,286.0	15,549.0				
	-0.4000	86.238	101.444						
		98.109	199.553	15,380.0	15,549.0	94.0	0.615%	169.0	1.087%
1.01	-0.4040	18.486	218.039			27			
	2	231.453	449.492	15,432.0	15,549.0	52.0	0.338%	117.0	0.752%
0.51	-0.2080	90.951	540.443			22			
		415.700	956.143	15,503.5	15,549.0	71.5	0.463%	45.5	0.293%
1.01	-0.2101	18.824	974.967			41			
	125	694.006	1,668.973	15,541.0	15,549.0	37.5	0.242%	8.0	0.051%
0.25	-0.0525	144.031	1,813.004			64			
		1,162.207	2,975.212	15,548.0	15,549.0	7.0	0.045%	1.0	0.006%
1.00	-0.0525	19.993	2,995.205			1			
		1,511.139	4,506.344	15,549.0	15,549.0	1.0	0.006%	0.0	0.000%
		18.591	4,524.935						

Table 4.16: Computing HeKu20 using AKV5 with `numcut` = 800

4.2.5 From Strategy 5 to Strategy 6

In Strategy 6, the magnitude of any major and minor reduction to `vioRHS` cannot exceed 0.1, unless when the algorithm runs out of cuts and therefore it needs to lower `vioRHS` further to find cuts. This new change helps to calm the process of `vioRHS` reduction, and it successfully brings down the overall computation time as seen in Figure 4.21. Furthermore, this new approach results in the lowest run time thus far at `numcut` $= 300$, which is labeled in Figure 4.21 and detailed in Tables 4.17 and 4.18.

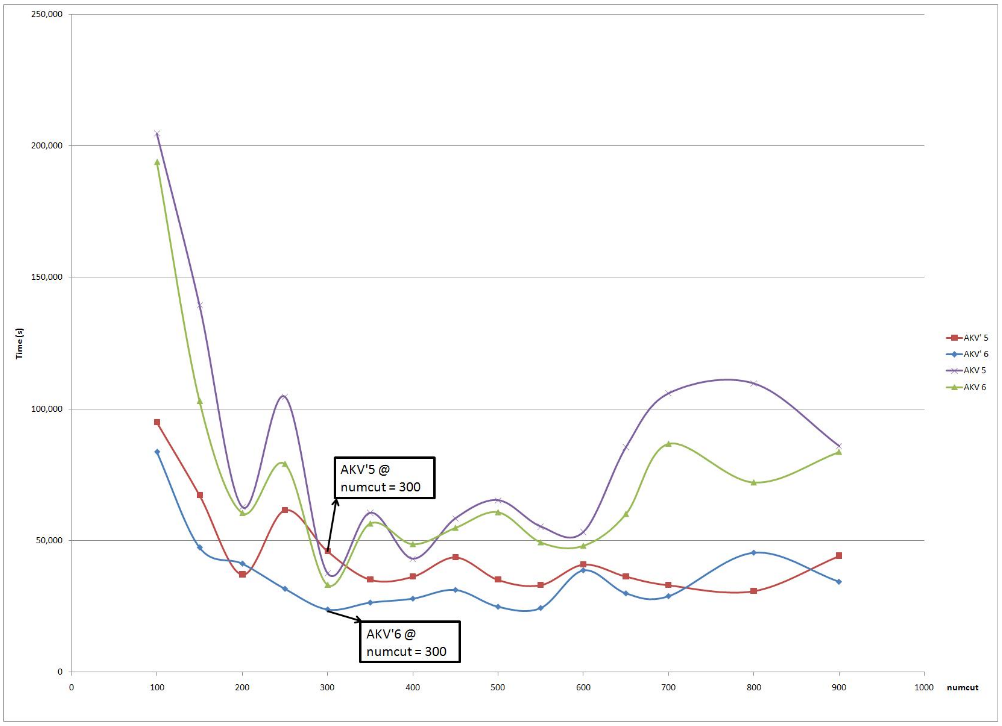

Figure 4.21: Comparison of Strategy 5 and Strategy 6 for AV25-2

The two circled durations in Table 4.17 are the total time required for finding and sorting the cuts after the algorithm made a major reduction in `vioRHS` due to shortages of cuts. The drop in `vioRHS` in Strategy 5 is much more aggressive than in the updated Strategy 6. Consequently, Table 4.18 shows that although Strategy 6 requires more frequent major reductions in `vioRHS`, each drop to `vioRHS` does not exceed 0.1, and therefore the required cut-searching time is substantially decreased. Furthermore, This new change corrects the overly aggressive `vioRHS` reduction in Strategy 5, such as the case shown in Table 4.14.

When comparing AKV and AKV' using the instance AV25-1 as presented in Figure 4.22, it is still difficult to judge which model performs better, as the four curves lie closely to each other with many overlaps. However, when comparing Strategies 5 and 6, Figure 4.22 shows that Strategy 6 generally outperforms Strategy 5, with an exception when `numcut` $= 100$, which is analyzed in Tables 4.19 and 4.20. The circled duration in Table 4.19 represents the time taken in finding and sorting

AKV' 5		numcut	300						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.977						
		108.761	111.738	35,960.5	37,172.5				
	-0.4000	87.298	199.035						
		194.408	393.443	36,247.5	37,130.5	287.0	0.798%	883	2.378%
1.01	-0.4040	85.470	478.914			44			
	156	209.344	688.257	36,326.5	37,130.5	79.0	0.218%	804	2.165%
0.50	-0.2040	789.722	1,477.979			3			
		248.224	1,726.203	36,510.0	37,130.5	183.5	0.505%	620.5	1.671%
1.01	-0.2060	143.927	1,870.130			33			
		294.052	2,164.181	36,598.5	37,130.5	88.5	0.242%	532	1.433%
1.01	-0.2081	87.574	2,251.756			49			
		514.933	2,766.688	36,667.0	37,130.5	68.5	0.187%	463.5	1.248%
1.01	-0.2102	85.416	2,852.105			113			
		569.478	3,421.583	36,733.0	37,130.5	66.0	0.180%	397.5	1.071%
1.01	-0.2123	85.960	3,507.543			84			
		604.329	4,111.872	36,820.0	37,130.5	87.0	0.237%	310.5	0.836%
1.01	-0.2144	84.528	4,196.401			95			
		641.330	4,837.730	36,890.5	37,130.5	70.5	0.191%	240	0.646%
1.01	-0.2166	84.742	4,922.473			91			
	195	619.322	5,541.795	36,917.5	37,130.5	27.0	0.073%	213	0.574%
0.25	-0.0541	3,965.081	9,506.876			38			
		734.180	10,241.056	36,936.5	37,130.5	19.0	0.051%	194	0.522%
0.80	-0.0433	4,639.905	14,880.961			107			
		762.784	15,643.745	36,974.0	37,116.5	37.5	0.102%	142.5	0.384%
0.80	-0.0346	3,372.214	19,015.959			72			
		823.964	19,839.923	37,005.5	37,116.5	31.5	0.085%	111	0.299%
0.87	-0.0300	3,846.381	23,686.304			56			
		956.707	24,643.011	37,030.0	37,116.5	24.5	0.066%	86.5	0.233%
1.00	-0.0300	3,152.158	27,795.169			54			
		1,018.619	28,813.788	37,052.0	37,116.5	22.0	0.059%	64.5	0.174%
1.00	-0.0300	2,321.017	31,134.804			37			
		1,133.034	32,267.838	37,068.0	37,116.5	16.0	0.043%	48.5	0.131%
1.00	-0.0300	1,155.061	33,422.899			22			
		1,312.592	34,735.492	37,084.5	37,116.5	16.5	0.045%	32	0.086%
1.00	-0.0300	805.234	35,540.726			9			
		1,462.252	37,002.977	37,094.5	37,116.5	10.0	0.027%	22	0.059%
1.00	-0.0300	345.608	37,348.585			12			
		1,656.571	39,005.157	37,106.0	37,116.5	11.5	0.031%	10.5	0.028%
1.00	-0.0300	112.230	39,117.387			7			
		1,877.923	40,995.310	37,112.5	37,116.5	6.5	0.018%	4	0.011%
1.00	-0.0300	87.342	41,082.652			0			
		2,213.655	43,296.308	37,114.5	37,116.5	2.0	0.005%	2	0.005%
1.00	-0.0300	85.045	43,381.352			0			
		2,552.013	45,933.366	37,116.5	37,116.5	2.0	0.005%	0	0.000%
		90.153	46,023.519						

Table 4.17: Computing AV25-2 using AKV'5 with `numcut` = 300

the cuts after the algorithm experiences a shortage in the number of cuts. This major reduction to `vioRHS` is smoothed out in Strategy 6 as the two circled times in Table 4.20 is much less than the cut-searching duration in Table 4.19. However, the reason why AKV'6 takes longer to close the gap is due to the continuous minor reduction to `vioRHS` as circled in the first column of Table 4.19. The algorithm continuosly decreases `vioRHS` because it finds that Z_{lb} is not improving enough. However, the modification to `vioRHS` is not the only factor that can affect the rate of lower bound improvement. The number of cuts added to the sub-problem in every iteration can also influence how the lower bound increases. In this case of `numcut` = 100, only roughly 100 cuts are added to the sub-problem at each iteration, so the impact of cuts is not great enough to cause quick improvement

AKV' 6		numcut	300						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.031						
		96.739	98.770	35,960.5	37,172.5				
	-0.4000	86.453	185.223						
		180.885	366.108	36,247.5	37,130.5	287.0	0.798%	883	2.378%
1.01	-0.4040	84.399	450.507			44			
	156	189.428	639.935	36,326.5	37,130.5	79.0	0.218%	804	2.165%
0.75	-0.3040	85.723	725.658			3			
		227.981	953.639	36,510.0	37,130.5	183.5	0.505%	620.5	1.671%
1.01	-0.3070	84.121	1,037.760			33			
	82	277.657	1,315.416	36,598.5	37,130.5	88.5	0.242%	532	1.433%
0.68	-0.2101	100.414	1,415.830			49			
		481.732	1,897.562	36,667.0	37,130.5	68.5	0.187%	463.5	1.248%
1.01	-0.2122	84.144	1,981.706			113			
		519.491	2,501.197	36,733.0	37,130.5	66.0	0.180%	397.5	1.071%
1.01	-0.2143	85.042	2,586.238			84			
		559.526	3,145.764	36,820.0	37,130.5	87.0	0.237%	310.5	0.836%
1.01	-0.2165	83.727	3,229.491			95			
	377	593.338	3,822.829	36,889.0	37,130.5	69.0	0.187%	241.5	0.650%
0.54	-0.1165	135.756	3,958.585			85			
		665.694	4,624.279	36,934.5	37,130.5	45.5	0.123%	196	0.528%
0.80	-0.0932	342.390	4,966.669			56			
		703.505	5,670.175	36,966.0	97,119.5	31.5	0.085%	60153.5	61.938%
0.80	-0.0745	543.363	6,213.537			51			
		775.767	6,989.304	36,995.0	97,119.5	29.0	0.078%	60124.5	61.908%
0.80	-0.0596	497.408	7,486.712			38			
		882.730	8,369.442	37,013.5	97,119.5	18.5	0.050%	60106	61.889%
0.80	-0.0477	1,143.247	9,512.689			24			
		968.919	10,481.609	37,048.0	97,119.5	34.5	0.093%	60071.5	61.853%
1.00	-0.0477	614.478	11,096.086			22			
		1,103.645	12,199.731	37,065.0	97,119.5	17.0	0.046%	60054.5	61.836%
1.00	-0.0477	332.150	12,531.881			5			
		1,241.196	13,773.077	37,079.5	97,119.5	14.5	0.039%	60040	61.821%
1.00	-0.0477	162.315	13,935.393			33			
		1,411.326	15,346.719	37,093.0	97,119.5	13.5	0.036%	60026.5	61.807%
1.00	-0.0477	87.550	15,434.269			4			
		1,571.538	17,005.807	37,101.0	97,119.5	8.0	0.022%	60018.5	61.799%
1.00	-0.0477	85.911	17,091.719			2			
		1,773.855	18,865.573	37,109.5	37,116.5	8.5	0.023%	7	0.019%
1.00	-0.0477	85.719	18,951.292			7			
	15	2,023.516	20,974.808	37,115.0	37,116.5	5.5	0.015%	1.5	0.004%
0.25	-0.0119	375.719	21,350.528			0			
		2,364.183	23,714.710	37,116.5	37,116.5	1.5	0.004%	0	0.000%
		88.253	23,802.964						

Table 4.18: Computing AV25-2 using AKV’6 with `numcut` = 300

in Z_{lb}, even after several attempts to adjust the number of cuts that can be found by changing the `vioRHS`. In summary, this exception case where Strategy 6 yields a longer computing time is mainly due to a low `numcut` parameter, which is not sufficient to conjecture Strategy 6 is worse than Strategy 5. Furthermore, since low `numcut` generally yields long computing time, the exception case as presented in Table 4.20 is not concerning for future development.

Finally, Figure 4.23 shows the comarison of AKV5, AKV6, AKV’5, and AKV’6 when solving the smaller instance HeKu20. It is straightforward to see that AKV’ has clearly outperformed AKV especially when `numcut` increases. Furthermore, the figure also demonstrates that Strategy 6 has consistently improved the overall computing time.

AKV' 5		numcut	100						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.336						
		98.704	101.039	4,463.5	4,618.0				
	-0.4000	86.959	187.998						
		148.512	336.510	4,488.0	4,618.0	24.5	0.549%	130	2.815%
1.01	-0.4040	82.854	419.364			4			
	46	184.354	603.718	4,511.5	4,618.0	23.5	0.524%	106.5	2.306%
0.51	-0.2080	692.324	1,296.041			8			
		205.451	1,501.492	4,531.0	4,618.0	19.5	0.432%	87	1.884%
1.01	-0.2101	289.284	1,790.776			18			
		213.170	2,003.946	4,543.5	4,618.0	12.5	0.276%	74.5	1.613%
1.01	-0.2122	109.702	2,113.648			4			
		235.027	2,348.675	4,552.5	4,618.0	9.0	0.198%	65.5	1.418%
1.01	-0.2143	85.041	2,433.716			6			
		240.737	2,674.453	4,562.0	4,618.0	9.5	0.209%	56	1.213%
1.01	-0.2165	83.385	2,757.838			5			
		259.609	3,017.447	4,568.5	4,618.0	6.5	0.142%	49.5	1.072%
1.01	-0.2187	82.903	3,100.350			8			
		269.762	3,370.112	4,575.0	4,618.0	6.5	0.142%	43	0.931%
1.01	-0.2208	82.919	3,453.031			12			
		289.571	3,742.602	4,581.0	4,618.0	6.0	0.131%	37	0.801%
1.01	-0.2230	82.984	3,825.586			19			
		325.439	4,151.025	4,586.5	4,618.0	5.5	0.120%	31.5	0.682%
0.80	-0.1784	83.201	4,234.226			25			
		330.144	4,564.370	4,592.5	4,618.0	6.0	0.131%	25.5	0.552%
1.01	-0.1802	82.917	4,647.287			12			
		318.671	4,965.958	4,596.5	4,618.0	4.0	0.087%	21.5	0.466%
0.80	-0.1442	83.505	5,049.463			7			
		351.926	5,401.389	4,599.5	4,618.0	3.0	0.065%	18.5	0.401%
0.80	-0.1153	91.260	5,492.649			12			
		342.936	5,835.586	4,602.5	4,618.0	3.0	0.065%	15.5	0.336%
0.80	-0.0923	133.921	5,969.507			7			
		351.236	6,320.743	4,605.0	4,618.0	2.5	0.054%	13	0.282%
0.80	-0.0738	254.290	6,575.034			10			
		387.531	6,962.564	4,607.0	4,618.0	2.0	0.043%	11	0.238%
0.80	-0.0591	829.479	7,792.044			9			
		402.779	8,194.823	4,609.5	4,618.0	2.5	0.054%	8.5	0.184%
1.00	-0.0591	385.634	8,580.457			2			
		428.124	9,008.581	4,611.5	4,618.0	2.0	0.043%	6.5	0.141%
1.00	-0.0591	246.271	9,254.853			11			
		447.279	9,702.132	4,612.5	4,618.0	1.0	0.022%	5.5	0.119%
1.00	-0.0591	198.640	9,900.772			4			
		444.047	10,344.819	4,614.0	4,618.0	1.5	0.033%	4	0.087%
1.00	-0.0591	118.023	10,462.842			5			
		475.219	10,938.062	4,615.0	4,618.0	1.0	0.022%	3	0.065%
1.00	-0.0591	103.195	11,041.256			9			
		514.936	11,556.192	4,616.0	4,618.0	1.0	0.022%	2	0.043%
1.00	-0.0591	83.204	11,639.396			2			
		552.778	12,192.174	4,617.0	4,618.0	1.0	0.022%	1	0.022%
1.00	-0.0591	83.316	12,275.490			1			
		570.219	12,845.709	4,617.0	4,618.0	0.0	0.000%	1	0.022%
1.00	-0.0591	83.253	12,928.961			14			
		609.167	13,538.129	4,617.5	4,618.0	0.5	0.011%	0.5	0.011%
1.00	-0.0591	83.266	13,621.395			3			
	15	669.927	14,291.321	4,617.5	4,618.0	0.0	0.000%	0.5	0.011%
0.25	-0.0148	1,240.288	15,531.610						
		728.321	16,259.931	4,618.0	4,618.0	0.5	0.011%	0	0.000%
		91.186	16,351.117						

Table 4.19: Computing AV25-1 using AKV'5 with `numcut` = 100

AKV' 6		numcut	100						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.916						
		98.213	101.129	4,463.5	4,618.0				
	-0.4000	88.137	189.266						
		148.842	338.109	4,488.0	4,618.0	24.5	0.549%	130	2.815%
1.01	-0.4040	83.429	421.538			4			
	46	183.953	605.491	4,511.5	4,618.0	23.5	0.524%	106.5	2.306%
0.76	-0.3080	99.632	705.123			8			
		206.305	911.428	4,531.0	4,618.0	19.5	0.432%	87	1.884%
1.01	-0.3111	83.544	994.972			18			
		211.995	1,206.967	4,543.5	4,618.0	12.5	0.276%	74.5	1.613%
1.01	-0.3142	83.372	1,290.339			4			
		234.897	1,525.236	4,552.5	4,618.0	9.0	0.198%	65.5	1.418%
1.01	-0.3174	84.248	1,609.484			6			
	98	250.679	1,860.162	4,561.5	4,618.0	9.0	0.198%	56.5	1.223%
0.68	-0.2174	84.027	1,944.189			9			
		268.206	2,212.396	4,569.0	4,618.0	7.5	0.164%	49	1.061%
1.01	-0.2195	83.484	2,295.879			8			
		280.139	2,576.018	4,575.5	4,618.0	6.5	0.142%	42.5	0.920%
1.01	-0.2217	83.647	2,659.665			9			
		296.965	2,956.631	4,580.0	4,618.0	4.5	0.098%	38	0.823%
0.80	-0.1774	84.049	3,040.680			18			
		308.459	3,349.138	4,585.0	4,618.0	5.0	0.109%	33	0.715%
0.80	-0.1419	101.880	3,451.019			12			
		328.625	3,779.644	4,589.0	4,618.0	4.0	0.087%	29	0.628%
0.80	-0.1135	198.640	3,978.283			19			
		339.592	4,317.876	4,593.5	4,618.0	4.5	0.098%	24.5	0.531%
0.80	-0.0908	305.175	4,623.050			17			
		354.309	4,977.360	4,597.0	4,618.0	3.5	0.076%	21	0.455%
0.80	-0.0727	727.447	5,704.807			15			
		350.105	6,054.912	4,600.0	4,618.0	3.0	0.065%	18	0.390%
0.80	-0.0581	1,279.292	7,334.204			17			
		355.283	7,689.487	4,603.0	4,618.0	3.0	0.065%	15	0.325%
0.80	-0.0465	2,095.796	9,785.283			11			
		373.322	10,158.604	4,605.5	4,618.0	2.5	0.054%	12.5	0.271%
0.80	-0.0372	2,784.474	12,943.078			10			
		389.218	13,332.296	4,608.5	4,618.0	3.0	0.065%	9.5	0.206%
0.81	-0.0300	3,752.878	17,085.174			3			
		402.409	17,487.583	4,610.0	4,618.0	1.5	0.033%	8	0.173%
1.00	-0.0300	3,471.585	20,959.169			9			
		415.414	21,374.582	4,612.5	4,618.0	2.5	0.054%	5.5	0.119%
1.00	-0.0300	2,228.326	23,602.908			8			
		428.579	24,031.487	4,613.5	4,618.0	1.0	0.022%	4.5	0.097%
1.00	-0.0300	2,326.351	26,357.839			10			
		455.020	26,812.859	4,615.0	4,618.0	1.5	0.033%	3	0.065%
1.00	-0.0300	799.094	27,611.952			3			
		481.222	28,093.174	4,616.0	4,618.0	1.0	0.022%	2	0.043%
1.00	-0.0300	631.606	28,724.780			2			
		499.668	29,224.448	4,617.0	4,618.0	1.0	0.022%	1	0.022%
1.00	-0.0300	189.106	29,413.555			3			
		529.333	29,942.888	4,617.5	4,618.0	0.5	0.011%	0.5	0.011%
1.00	-0.0300	126.323	30,069.212			0			
		546.506	30,615.718	4,617.5	4,618.0	0.0	0.000%	0.5	0.011%
1.00	-0.0300	86.994	30,702.711			1			
		577.217	31,279.929	4,618.0	4,618.0	0.5	0.011%	0	0.000%
		83.133	31,363.061						

Table 4.20: Computing AV25-1 using AKV'6 with `numcut` = 100

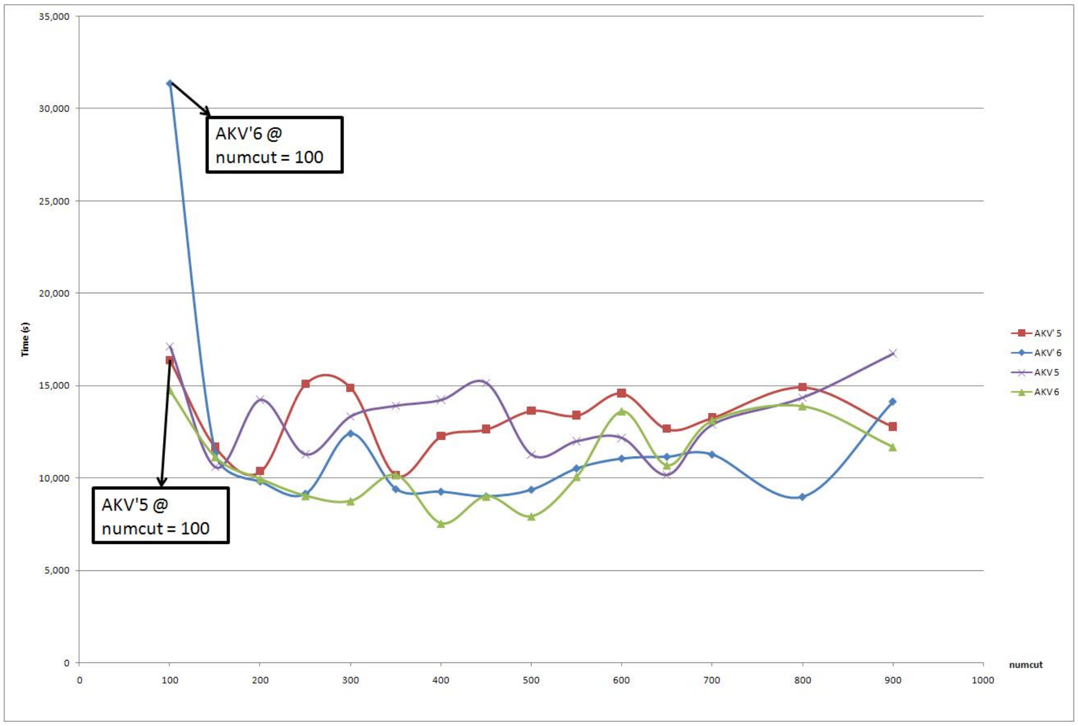

Figure 4.22: Comparison of Strategy 5 and Strategy 6 for AV25-1

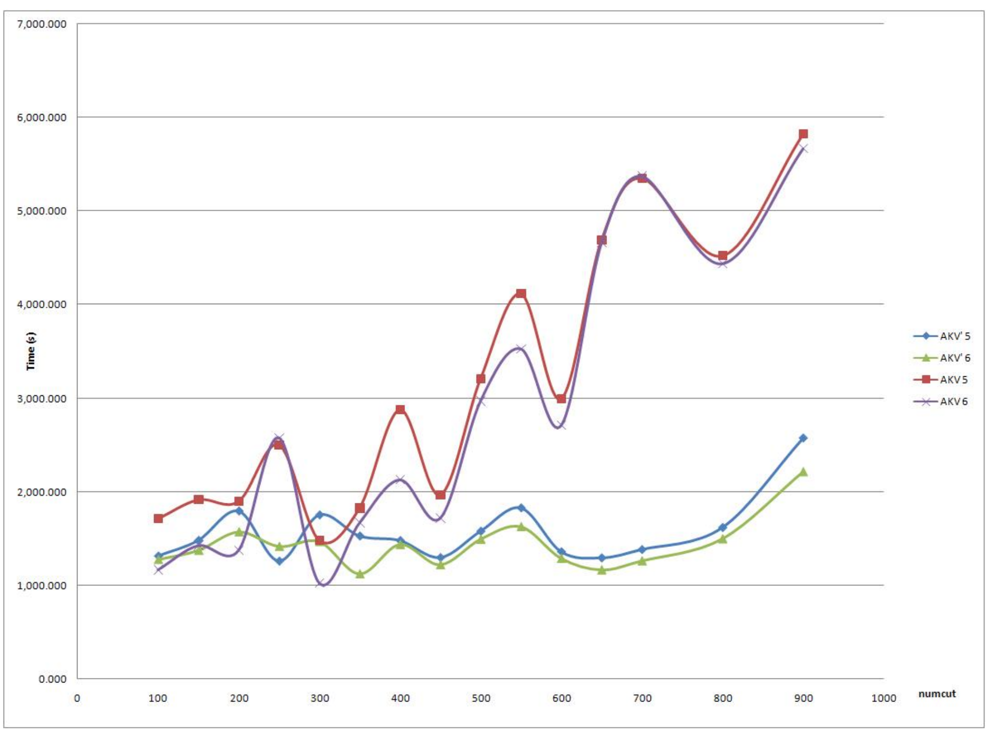

Figure 4.23: Comparison of Strategy 5 and Strategy 6 for HeKu20

4.3 Summary of Experiments with Medium-Sized Instances

In this chapter, three medium-sized instances are analyzed to help us understand the effect of the parameters `vioRHS` and `numcut` on the computing time to solve the SRFLP. The conclusion to the medium-sized instances is likely to shed some light in solving large instances. This section should also provide some advice to the readers who wish to solve medium-sized SRFLPs using SDP and a cutting plane approach. It should be noted that the computer setup can also affect the parameter setting and the computing time. All the computational results were obtained on a 2.0GHz Dual Opteron with 16Gb of RAM.

4.3.1 HeKu20

HeKu20 is the smallest medium-sized instance studied in this thesis. As discussed in Section 4.2.2, the cutting plane algorithm runs faster at lower `numcut` due to smaller optimization problem size and simpler computation requirement. Therefore, the conclusion for HeKu20 does not extend to the solving of larger instances. Nonetheless, it is interesting to notice how fast the problem complexity grows with the number of departments.

The lowest run time (986 seconds) was achieved by AKV2 at `numcut` = 100. Although every new strategy with new features results in better overall performance and AKV' has consistently outperformed AKV, AKV2 at `numcut` = 100 becomes an exception with a very low computing time, see Figure 4.14. However, this combination may be a special case to HeKu20. Therefore, when solving a smaller medium-sized instance like HeKu20, it is recommended to use AKV' and Strategy 6 at low `numcut` in the range of 50 to 300. Strategy 6 is chosen for it is the least aggressive method which also prevents premature termination. AKV' is preferred since it almost always outperforms AKV, and consequently AKV' should have better success rate.

4.3.2 AV25-1

AV25-1 is an example of the linear ordering problem, which is also a special case of the general SRFLP. Therefore, the conclusion for AV25-1 may not apply to solving general instances. Nevertheless, it is worthy of note to see that a different problem class can lead to a slightly different conclusion.

For AV25-1, although it is consistent to see newer strategies outperform the earlier versions, it is generally difficult to judge whether AKV or AKV' runs faster. Based on the graphs for AV25-1 in Section 4.2, the frequency of AKV outperforming AKV' seems slightly higher. In fact, the lowest computing time (7,532 seconds) comes from AKV6 at `numcut` = 400.

In conclusion, when solving a medium-sized linear ordering problem, it is suggested to use Strategy 6 since it has the smoothest approach to `vioRHS` while ensuring the algorithm will not terminate prematurely. Furthermore, medium-range `numcut` such as 350 to 500 generally yields lower computing time. However, the distinction between AKV and AKV' is not big enough to conclude which relaxation is better for this problem class. It is advised for the readers to carry out more detailed analysis on the linear ordering problem class using the cutting plane approach with AKV and AKV'. The readers can also refer to [16] for other cutting plane algorithms for the linear ordering problem.

4.3.3 AV25-2

AV25-2 is the most complicated and difficult SRFLP instance out of the three, and therefore the conclusion is likely to predict the computation for large instances. As studied in Section 4.2, AKV' has consistently outperformed AKV, and the new strategy has almost always improved its previous version. In fact, the best run time (23,803 seconds) is given by the combination of AKV'6 at `numcut` = 300. Therefore, when solving a medium-sized instance like AV25-2 with a similar computer setup, one should use the combination of AKV' and Strategy 6 while applying to the medium-range `numcut`. The medium-range `numcut` between 300 to 550 is observed to yield low computing time. However, high-range `numcut` after 550 also results in reasonably low run time which does not deviate much from the middle range. Therefore, it is also recommended to explore the performance of higher `numcut` when one wishes to solve the medium-sized instances like AV25-2 or larger instances. The study for large instances is presented in the next chapter.

4.4 Conclusion

The conclusion provides the highlight of Chapter 4.

To solve a small medium-sized instance like HeKu20, it is recommended to use:

- AKV' relaxation combined with Strategy 6,
- Low `numcut` in the range of 50 to 300.

For a medium-sized linear ordering problem like AV25-1, the distinction between AKV and AKV' relaxations is not prominent. Therefore, it is advised to explore both relaxations when approaching a problem class similar to AV25-1. The readers can also refer to [16] for other cutting plane algorithms targetted to linear ordering problem. Nevertheless, when solving a medium-sized instance like AV25-1 using the proposed cutting plane strategy, it is recommended to use:

- Strategy 6,
- Medium-range `numcut` such as between 350 to 500.

For the general medium-sized SRFLPs like AV25-2, it is recommended to use:

- AKV' relaxation combined with Strategy 6,
- Medium-range `numcut` such as between 300 to 550,
- Higher-range `numcut` is also recommended.

Chapter 5

Global Solutions for Large Instances

The SRFLP is strongly NP-hard [1] and remains a difficult problem class. By utilizing AKV with a simple cutting plane scheme, Anjos and Vannelli obtained global optimal solutions for a few large SRFLPs up to 30 departments that had remained unsolved since 1988 [5]. This achievement was considered a breakthrough in the field. Most recently, Amaral presented a new lower bound that solved instances of size up to $n = 35$ in [2]. In this thesis, six new large instances with 36 departments were successfully solved to optimality using AKV' and the new cutting plane methodology. We also briefly point out how the computing time can vary greatly between different sets of data of the same size.

5.1 New Strategies for Large Instances

The combination of Strategy 6 with AKV' was considered the best approach in solving the medium-sized instances. However, after a few attempts to solve some larger instances, the weaknesses of Strategy 6 began to reveal themselves. Consequently, two new strategies are proposed to handle large instances. This section presents Strategies 7 and 8, and Section 5.2 reports the experimental results and analysis.

5.1.1 Strategy 7

Strategy 7 changes the approach in the major adjustment of `vioRHS` when the problem is running low on the violations found. Instead of decreasing `vioRHS` by 75%, the new approach lowers `vioRHS` by half. This new change also calms the `vioRHS` reduction process. Figures 5.1 and 5.2 illustrate the logic of the cutting plane algorithm and the modification process of `vioRHS` in Strategy 7. The impact of these new changes is illustrated in Section 5.2.1.

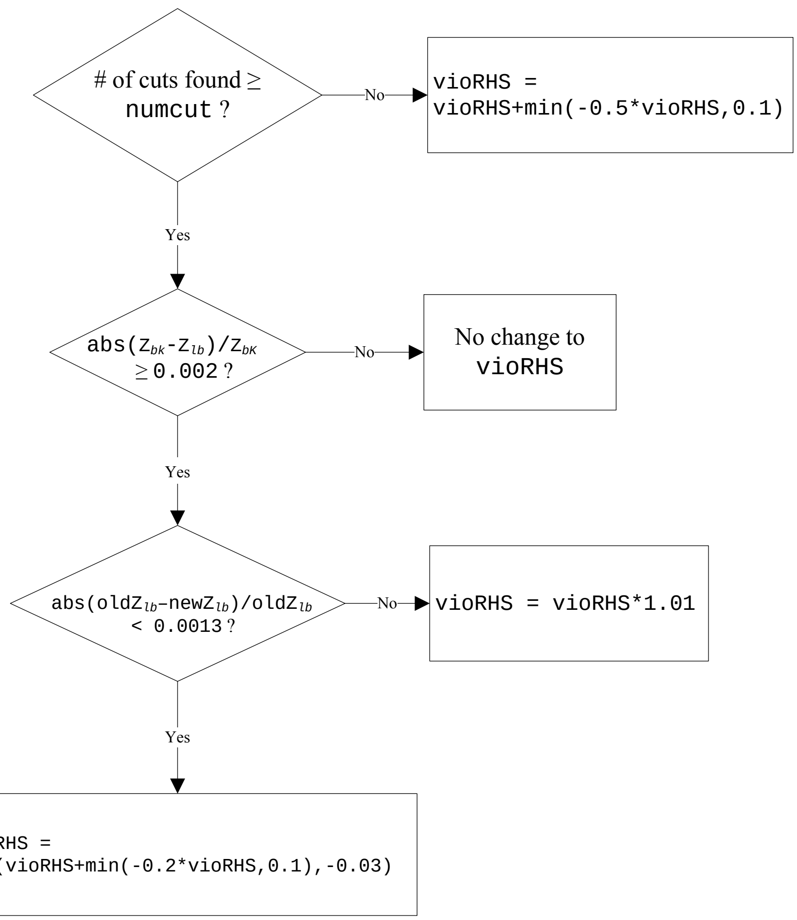

Figure 5.1: Strategy 7 on modification of `vioRHS`

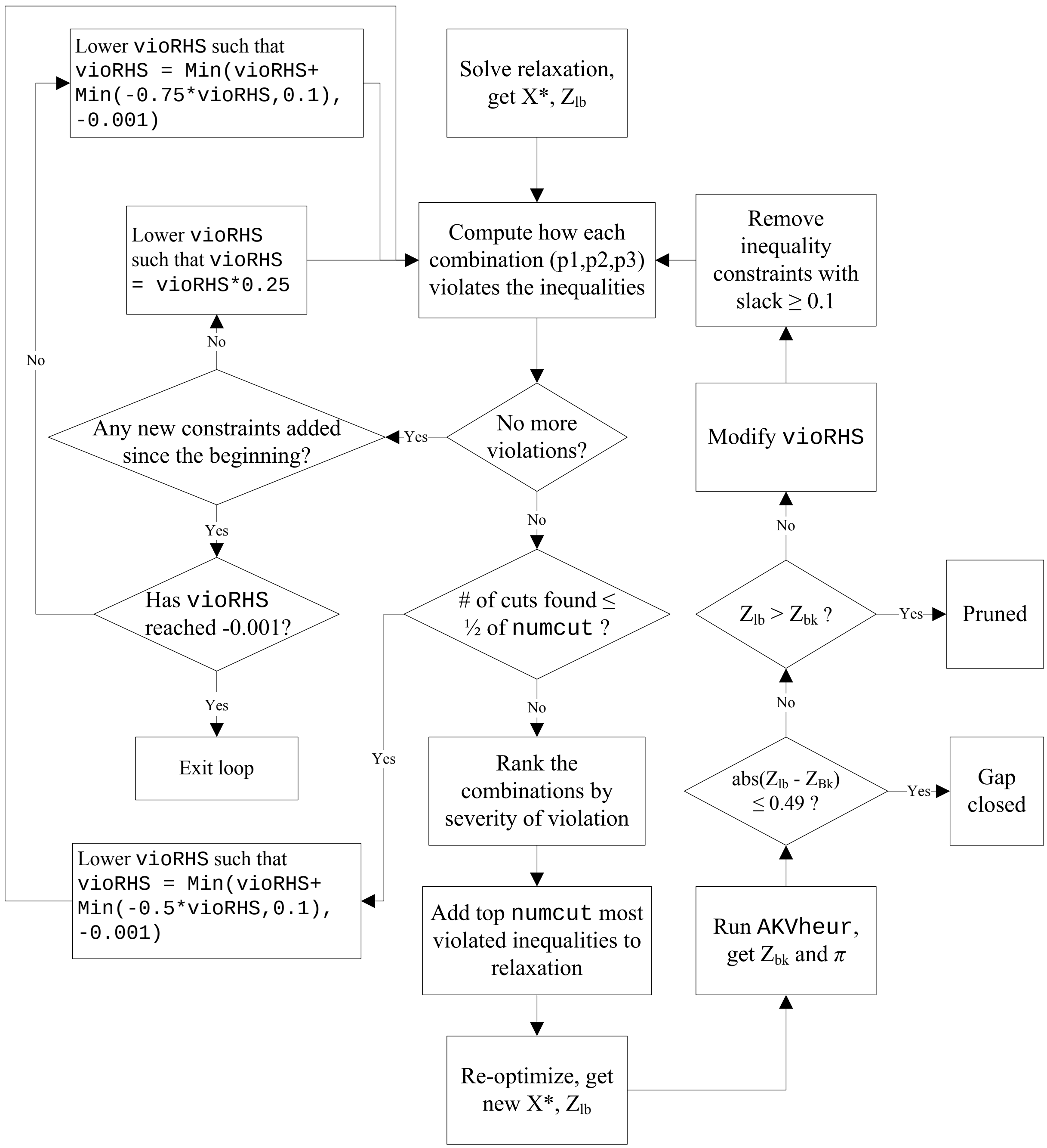

Figure 5.2: Cutting plane algorithm for Strategy 7

5.1.2 Strategy 8

When the problem runs low on the violations found, Strategy 8 also handles the major adjustment of `vioRHS` differently. As demonstrated in Figures 5.3 and 5.4, instead of decreasing `vioRHS` by half as in Strategy 7, the new approach lowers `vioRHS` by 40%. This new change further smoothes the `vioRHS` reduction process. The effect of these new modifications is studied in Section 5.3.

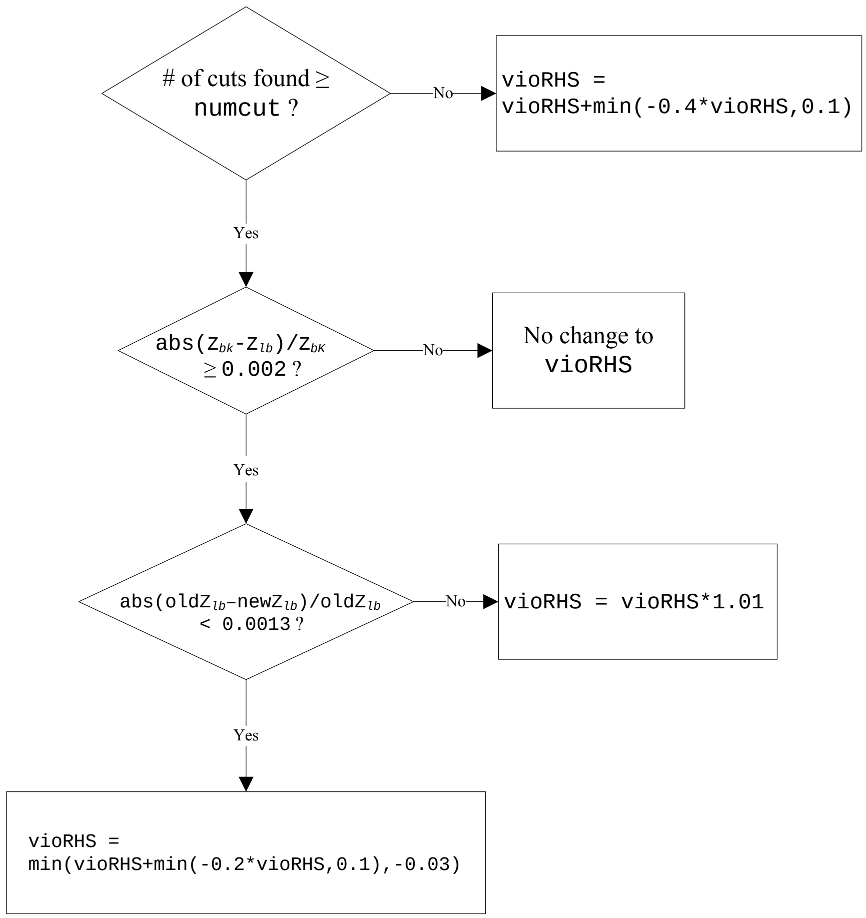

Figure 5.3: Strategy 8 on modification of `vioRHS`

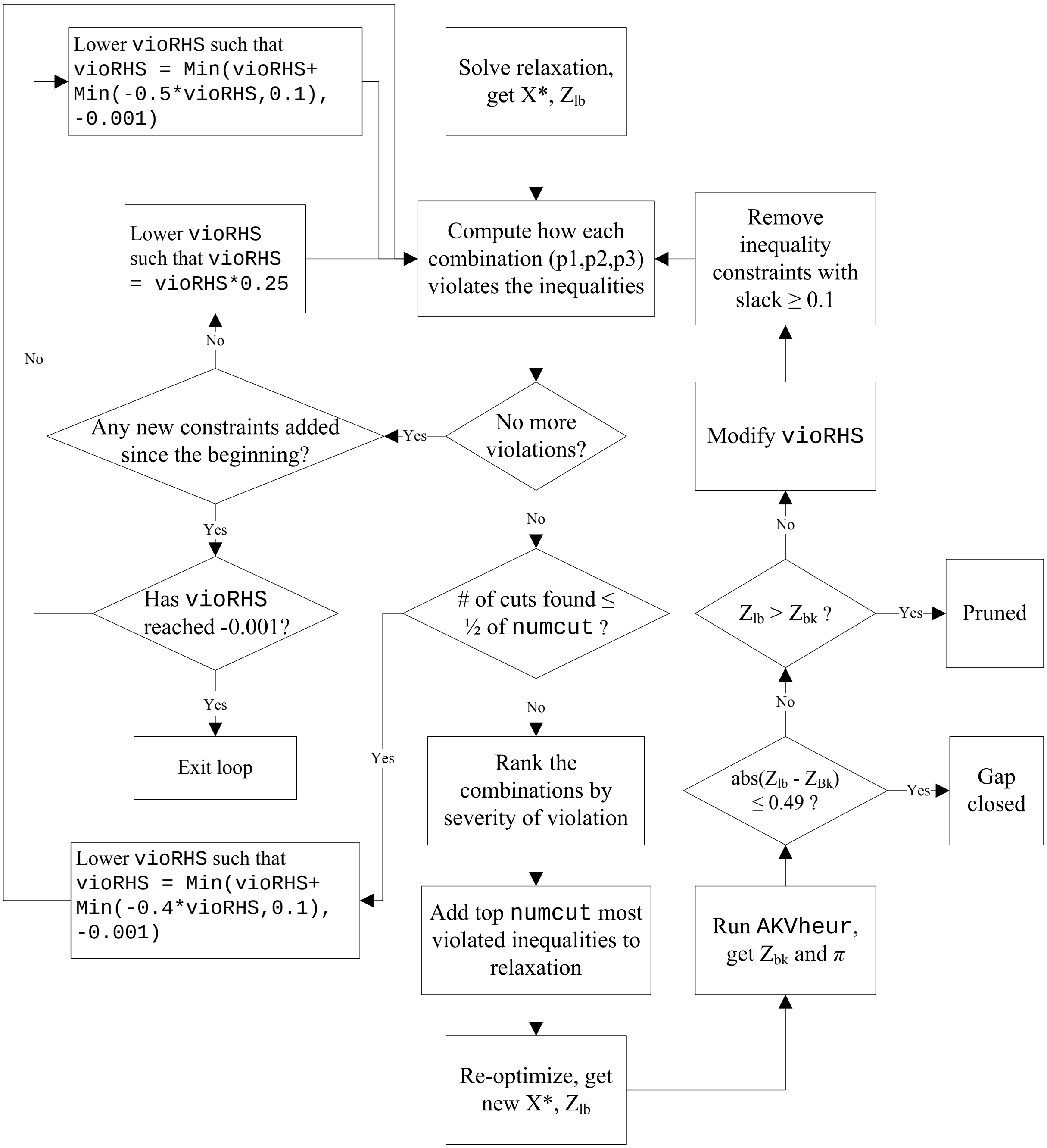

Figure 5.4: Cutting plane algorithm for Strategy 8

5.2 Experimental Analysis

This section reports the preliminary findings in developing Strategies 7 and 8. These results were obtained by SDPT3 version 4.0 [40] on a 2.0GHz Dual Opteron with 16Gb of RAM. However, it was later found out that as the problem size increases, the optimization time becomes too long. As a result, newer computations were generated on a Sun Fire V890 8*1.2GHz with 64Gb of RAM, while the SDP problems were solved using the interior-point solver CSDP (version 5.0) of [8] in conjunction with the ATLAS library of routines [41]. A simple comparative analysis between these two computing setups is also documented in this section to provide the best combination in solving the new large instances.

Several large instances were used in this section. HeKu30 is from Heragu and Kusiak in [20], while STE36-1 is created by Anjos and Yen in [6] and is originally based on the QAP instance from Steinberg in [38]. It should be noted that while STE36-1 is a linear ordering problem instance, HeKu30 is a SRFLP instance with varying lengths. All of the instances used in this thesis are listed in Appendix C.

5.2.1 Preliminary Results by SDPT3

Upon obtaining Strategies 4, 5, and 6, they were used to solve a few larger instances by SDPT3 version 4.0 [40] on a Sun Fire V890 8*1.2 GHz with 64 Gb of RAM. But after several attempts, it was observed that even the best strategy for the medium-sized problem is still not good enough for large instances. The main problem is that the `vioRHS` reduction process is still too aggressive, which leads to substantial CPU time to find and sort the cuts. Tables 5.1 and 5.5 illustrate the impact of the minor changes made in Strategies 7 and 8 to smooth out the `vioRHS` reduction process.

Table 5.1 shows that although the number of iterations has increased slightly as we updated the strategies, the computation time has greatly decreased. The percentage differences in the total CPU time between AKV'4 and AKV'8 are 18.9% and 24.1% for `numcut` = 700 and 800 respectively. Even for AKV'6, the best strategy for the medium-sized problem, the percentage difference to AKV'8 is as high as 20.6% at `numcut` = 800. The number of iterations may increase slightly for the newer strategies because as the changes made to `vioRHS` becomes more

numcut	AKV'4		AKV'5		AKV'6		AKV'7		AKV'8	
	CPU time (sec)	Number of iterations	CPU time (sec)	Number of iterations	CPU time (sec)	Number of iterations	CPU time (sec)	Number of iterations	CPU time (sec)	Number of iterations
700	87,602	17	93,232	16	72,781	18	71,222	18	71,045	18
800	82,513	16	94,072	14	78,967	15	74,139	15	62,663	14

Table 5.1: Comparison of Strategies 4, 5, 6, 7, and 8 Using HeKu30

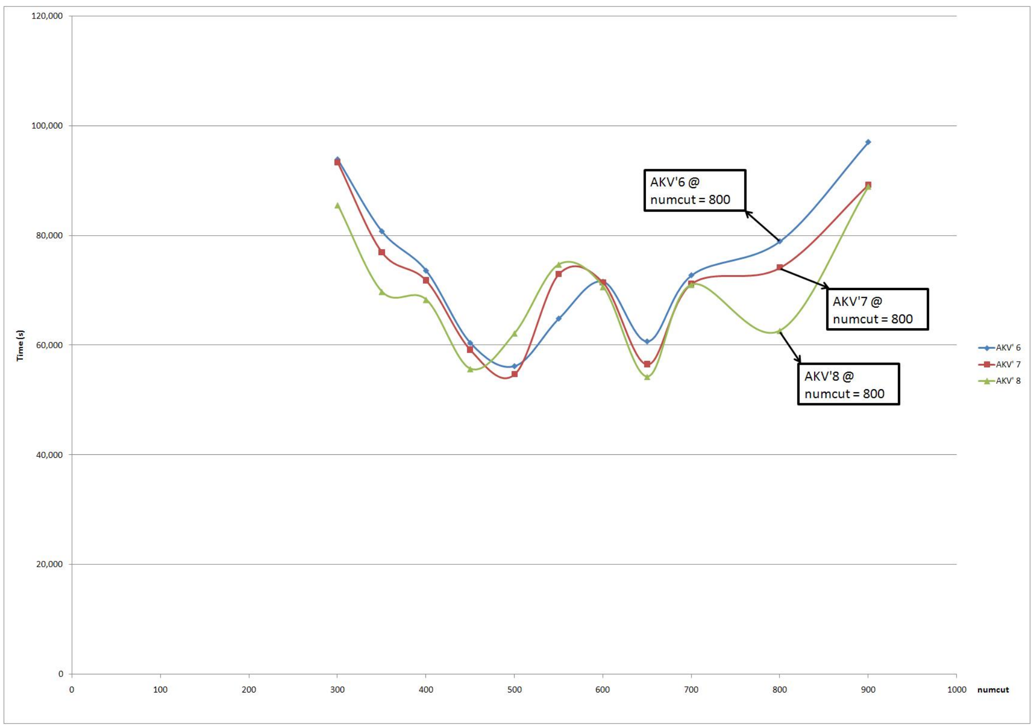

Figure 5.5: Comparison of Strategies 6, 7, and 8 for HeKu30

gentle, the rate of improvement to Z_{lb} may also become lower. As a result, it may sometimes take a few more iterations to close the gap. However, since the time duration for each iteration becomes much shorter, the overall effect is usually positive.

Figure 5.5 shows that Strategy 8 generally outperforms Strategies 6 and 7. In fact, Strategy 8 produces the shortest run time (54,266 seconds) for HeKu30 at `numcut` = 650. The three labeled data points are explained in Tables 5.2, 5.3, and 5.4. The two circled time duration values in Table 5.2 illustrate the result of an

AKV' 6		numcut	800						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.951						
		177.808	180.759	43,448.5	45,073.0				
	-0.4000	324.136	504.894						
	307	572.130	1,077.025	43,884.0	45,014.0	435.5	1.002%	1130	2.510%
0.76	-0.3040	341.223	1,418.248			219			
		1,070.126	2,488.374	44,149.5	45,014.0	265.5	0.605%	864.5	1.921%
1.01	-0.3070	287.439	2,775.813			211			
	204	1,276.229	4,052.043	44,324.0	45,014.0	174.5	0.395%	690	1.533%
0.68	-0.2101	339.922	4,391.965			319			
		1,698.199	6,090.164	44,463.0	45,014.0	139.0	0.314%	551	1.224%
1.01	-0.2122	288.472	6,378.635			244			
	302	2,128.733	8,507.368	44,596.5	45,014.0	133.5	0.300%	417.5	0.927%
0.54	-0.1143	544.995	9,052.362			284			
		2,506.958	11,559.321	44,705.5	44,983.0	109.0	0.244%	277.5	0.617%
1.01	-0.1155	307.824	11,867.144			236			
		3,210.261	15,077.406	44,791.0	44,983.0	85.5	0.191%	192	0.427%
1.01	-0.1166	290.415	15,367.820			121			
		4,057.708	19,425.528	44,844.5	44,975.0	53.5	0.119%	130.5	0.290%
0.80	-0.0933	289.975	19,715.503			66			
		5,099.051	24,814.554	44,888.0	44,974.0	43.5	0.097%	86	0.191%
1.00	-0.0933	288.402	25,102.956			51			
		6,213.989	31,316.944	44,911.0	44,974.0	23.0	0.051%	63	0.140%
1.00	-0.0933	289.381	31,606.325			26			
	445	7,198.061	38,804.386	44,930.5	44,965.0	19.5	0.043%	34.5	0.077%
0.25	-0.0233	4,229.378	43,033.764			17			
		8,827.271	51,861.034	44,948.5	44,965.0	18.0	0.040%	16.5	0.037%
1.00	-0.0233	1,372.404	53,233.438			11			
		11,111.414	64,344.852	44,963.0	44,965.0	14.5	0.032%	2	0.004%
1.00	-0.0233	292.307	64,637.159			6			
		14,061.976	78,699.135	44,965.0	44,965.0	2.0	0.004%	0	0.000%
		268.024	78,967.159						

Table 5.2: Computing HeKu30 using AKV'6 with `numcut` = 800

overly aggressive drop to `vioRHS`. The time required to finding and sorting the cuts suddenly surged up when `vioRHS` was reduced by 75% due to a shortage of cuts found. However, when we change the reduction percentage from 75% to 50% in Strategy 7, the resultant cut-searching time period is much smaller (Table 5.3). As the reduction percentage to `vioRHS` is further reduced to 40% in Strategy 8, the overall computing time is also lessened as shown in Table 5.4. Furthermore, because of this change in the approach to lower `vioRHS`, the major reduction occurs earlier in the process, which helps the algorithm to quickly improve Z_{lb} and close the gap. Consequently, AKV'8 at `numcut` = 800 requires one less iteration than both Strategies 6 and 7.

The impact of the new approach amplifies as the instance size increases. Table 5.5 presents the results of a few trials to solve STE36-1. Although STE36-1 is a linear ordering problem instance, the computing time still rises substantially. HeKu30 requires 93,232 seconds to reach optimality by AKV'5 at `numcut` = 700

AKV' 7		numcut	800						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.237						
		173.511	175.747	43,448.5	45,073.0				
	-0.4000	290.198	465.946						
	307	582.008	1,047.953	43,884.0	45,014.0	435.5	1.002%	1130	2.510%
0.76	-0.3040	296.638	1,344.591			219			
		1,074.930	2,419.521	44,149.5	45,014.0	265.5	0.605%	864.5	1.921%
1.01	-0.3070	265.892	2,685.413			211			
	204	1,274.721	3,960.134	44,324.0	45,014.0	174.5	0.395%	690	1.533%
0.68	-0.2101	296.328	4,256.462			319			
		1,707.916	5,964.378	44,463.0	45,014.0	139.0	0.314%	551	1.224%
1.01	-0.2122	266.853	6,231.231			244			
	302	2,121.240	8,352.470	44,596.5	45,014.0	133.5	0.300%	417.5	0.927%
0.54	-0.1143	494.338	8,846.808			284			
		2,528.623	11,375.431	44,705.5	44,983.0	109.0	0.244%	277.5	0.617%
1.01	-0.1155	279.316	11,654.747			236			
		3,219.060	14,873.807	44,791.0	44,983.0	85.5	0.191%	192	0.427%
1.01	-0.1166	269.208	15,143.015			121			
		4,099.711	19,242.727	44,844.5	44,975.0	53.5	0.119%	130.5	0.290%
0.80	-0.0933	269.217	19,511.944			66			
		5,126.070	24,638.014	44,888.0	44,974.0	43.5	0.097%	86	0.191%
1.00	-0.0933	268.473	24,906.487			51			
		6,238.827	31,145.314	44,911.0	44,974.0	23.0	0.051%	63	0.140%
1.00	-0.0933	268.728	31,414.042			26			
	445	7,237.491	38,651.534	44,930.5	44,965.0	19.5	0.043%	34.5	0.077%
0.50	-0.0467	303.176	38,954.709			17			
		8,984.893	47,939.602	44,948.5	44,965.0	18.0	0.040%	16.5	0.037%
1.00	-0.0467	270.740	48,210.342			11			
	171	11,246.848	59,457.190	44,963.0	44,965.0	14.5	0.032%	2	0.004%
0.50	-0.0233	294.454	59,751.644			6			
		14,144.560	73,896.204	44,965.0	44,965.0	2.0	0.004%	0	0.000%
		242.583	74,138.786						

Table 5.3: Computing HeKu30 using AKV'7 with `numcut` = 800

(see Table 5.1). The total CPU time required rises up to 1,205,688 seconds (approximately 14 days) for the same strategy combination to solve STE36-1. This long computing time is 12.9 times of the total CPU time for the smaller instance. The total run time has significantly decreased for Strategy 6 at `numcut` = 350. However, it is still quite substantial to solve on a routine basis.

Tables 5.6 and 5.7 break down the entire cutting plane processes of the two abovementioned cases that result in extensive computing time. The circled time periods in both tables indicate that the cause of this significant growth in computing time is the aggressive reduction to `vioRHS` after the algorithm runs short of cuts. After the reduction rate becomes lower in Strategies 7 and 8, the required cut-searching time becomes much smaller as seen in Tables 5.10 and 5.11.

However, we can also observe Strategy 8 yielding a slightly longer computing time than Strategy 7 for STE36-1. As shown in Tables 5.10 and 5.11, there is not any major difference between the two strategies in terms of the time duration in

AKV'8		numcut	800						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			2.227						
		177.863	180.090	43,448.5	45,073.0				
	-0.4000	290.532	470.621						
	307	580.595	1,051.216	43,884.0	45,014.0	435.5	1.002%	1130	2.510%
0.76	-0.3040	296.357	1,347.573			219			
		1,061.708	2,409.281	44,149.5	45,014.0	265.5	0.605%	864.5	1.921%
1.01	-0.3070	265.184	2,674.465			211			
	204	1,272.158	3,946.623	44,324.0	45,014.0	174.5	0.395%	690	1.533%
0.68	-0.2101	295.682	4,242.304			319			
		1,699.331	5,941.636	44,463.0	45,014.0	139.0	0.314%	551	1.224%
1.01	-0.2122	266.197	6,207.833			244			
	302	2,145.489	8,353.322	44,596.5	45,014.0	133.5	0.300%	417.5	0.927%
0.61	-0.1286	328.316	8,681.639			284			
		2,529.514	11,211.152	44,705.5	44,983.0	109.0	0.244%	277.5	0.617%
1.01	-0.1299	268.898	11,480.050			236			
		3,221.350	14,701.400	44,791.0	44,983.0	85.5	0.191%	192	0.427%
1.01	-0.1312	266.887	14,968.287			121			
		4,078.184	19,046.471	44,844.5	44,975.0	53.5	0.119%	130.5	0.290%
0.80	-0.1049	267.238	19,313.709			66			
		5,166.329	24,480.038	44,888.0	44,974.0	43.5	0.097%	86	0.191%
1.00	-0.1049	267.653	24,747.692			51			
	140	6,235.927	30,983.619	44,911.0	44,974.0	23.0	0.051%	63	0.140%
0.60	-0.0630	293.481	31,277.100			26			
		7,804.665	39,081.764	44,941.0	44,965.0	30.0	0.067%	24	0.053%
1.00	-0.0630	269.477	39,351.241			23			
	152	9,906.302	49,257.543	44,955.0	44,965.0	14.0	0.031%	10	0.022%
0.60	-0.0378	295.255	49,552.798			8			
		12,868.354	62,421.152	44,965.0	44,965.0	10.0	0.022%	0	0.000%
		241.482	62,662.635						

Table 5.4: Computing HeKu30 using AKV'8 with `numcut` = 800

every iteration. The only major distinction is that the trial by Strategy 8 requires one more iteration than Strategy 7, and consequently, the total time requirement is higher. As explained earlier in the section, newer strategies may sometimes need more iterations as a result of a smoother `vioRHS` reduction approach. However, the time difference for this cause is usually not significant.

Table 5.5 also presents a case where AKV'8 outperforms AKV'7, which is illustrated in Tables 5.8 and 5.9 explain this comparison set in detail. The circled cut-searching time in Table 5.8 depicts a typical example when the reduction to `vioRHS` is too aggressive. On the contrary, Table 5.9 shows that Strategy 8 avoids this surge in computation time. Therefore, although Strategy 8 does not always outperforms Strategy 7, Strategy 8 is still preferred because it is overall a better approach to larger instances. Tables 5.16 and 5.17 from the next section show another comparison set that was generated by another computing setup, which is explained in detail in the next section. This comparison set shows a drastic improvement of Strategy 8 over Strategy 7.

`numcut`	AKV'5		AKV'6		AKV'7		AKV'8	
	CPU time (sec)	Number of iterations	CPU time (sec)	Number of iterations	CPU time (sec)	Number of iterations	CPU time (sec)	Number of iterations
350	N/A	N/A	318,978	14	54,681	14	61,046	15
500	N/A	N/A	N/A	N/A	60,006	12	69,219	13
700	1,205,688	11	N/A	N/A	82,341	12	78,803	12

Table 5.5: Comparison of Strategies 5, 6, 7, and 8 using STE36-1

AKV' 5		numcut	700						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			5.411						
		658.849	664.260	9,851.0	10,375.0				
	-0.4000	2,477.805	3,142.065						
		1,841.765	4,983.830	9,978.5	10,337.0	127.5	1.294%	358.5	3.468%
1.01	-0.4040	1,008.504	5,992.334			130			
	56	2,310.783	8,303.117	10,112.0	10,337.0	133.5	1.338%	225	2.177%
0.51	-0.2080	4,517.934	12,821.050			100			
		2,911.915	15,732.965	10,185.0	10,289.0	73.0	0.722%	104	1.011%
1.01	-0.2101	3,205.641	18,938.606			115			
		3,639.163	22,577.769	10,223.0	10,289.0	38.0	0.373%	66	0.641%
1.01	-0.2122	1,075.563	23,653.332			115			
		4,684.676	28,338.008	10,242.5	10,289.0	19.5	0.191%	46.5	0.452%
1.01	-0.2143	1,009.102	29,347.111			317			
	72	5,129.806	34,476.917	10,262.0	10,289.0	19.5	0.190%	27	0.262%
0.25	-0.0541	704,286.273	738,763.190			118			
		5,945.456	744,708.646	10,275.0	10,289.0	13.0	0.127%	14	0.136%
1.00	-0.0541	430,724.893	1,175,433.539			94			
		7,411.893	1,182,845.433	10,283.5	10,287.0	8.5	0.083%	3.5	0.034%
1.00	-0.0541	1,935.060	1,184,780.493			7			
		8,630.150	1,193,410.642	10,285.5	10,287.0	2.0	0.019%	1.5	0.015%
1.00	-0.0541	1,015.036	1,194,425.678			5			
		10,331.404	1,204,757.082	10,287.0	10,287.0	1.5	0.015%	0	0.000%
		930.995	1,205,688.077						

Table 5.6: Computing STE36-1 using AKV'5 with `numcut` = 700

Although the two large instances show some major impacts of the new approach in Strategies 7 and 8, one may wonder whether this new approach can improve the performance of the medium-sized problems. Therefore, a comparison of Strategies 6, 7, and 8 was made for AV25-2 and the result is presented in Figure 5.6. It is observed that the newer strategy, for the most part, outperforms the earlier version, and therefore Strategy 8 has the lowest running time overall. In fact, the lowest ever computing time for AV25-2 is 23,128 seconds, which is generated by AKV'8 at `numcut` = 300. However, the graph also shows that the difference between the three strategies is very small. Nevertheless, this graph also shows that AKV'

AKV' 6		numcut	350						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			4.698						
		681.582	686.280	9,851.0	10,375.0				
	-0.4000	2,310.935	2,997.215						
		1,428.858	4,426.073	9,954.5	10,289.0	103.5	1.051%	334.5	3.251%
1.01	-0.4040	991.023	5,417.096			57			
		1,611.775	7,028.871	10,094.0	10,289.0	139.5	1.401%	195	1.895%
1.01	-0.4080	990.416	8,019.286			47			
	192	2,074.771	10,094.058	10,119.5	10,289.0	25.5	0.253%	169.5	1.647%
0.75	-0.3080	990.719	11,084.777			30			
		2,233.789	13,318.566	10,160.5	10,289.0	41.0	0.405%	128.5	1.249%
1.01	-0.3111	989.812	14,308.378			66			
		2,435.471	16,743.848	10,183.5	10,289.0	23.0	0.226%	105.5	1.025%
1.01	-0.3142	990.233	17,734.082			28			
	360	2,849.435	20,583.517	10,197.0	10,289.0	13.5	0.133%	92	0.894%
0.68	-0.2142	1,184.993	21,768.510			63			
		2,890.870	24,659.380	10,228.5	10,289.0	31.5	0.309%	60.5	0.588%
1.01	-0.2164	990.573	25,649.953			66			
		3,203.916	28,853.869	10,248.5	10,289.0	20.0	0.196%	40.5	0.394%
1.01	-0.2185	989.526	29,843.395			115			
	237	3,476.046	33,319.441	10,261.0	10,289.0	12.5	0.122%	28	0.272%
0.54	-0.1185	1,523.276	34,842.717			93			
		3,514.358	38,357.075	10,275.5	10,289.0	14.5	0.141%	13.5	0.131%
1.00	-0.1185	991.756	39,348.831			24			
		4,013.645	43,362.476	10,281.5	10,287.0	6.0	0.058%	5.5	0.053%
1.00	-0.1185	992.179	44,354.655			20			
	246	4,225.748	48,580.402	10,284.0	10,287.0	2.5	0.024%	3	0.029%
0.25	-0.0296	264,955.427	313,535.830			2			
		4,522.976	318,058.806	10,287.0	10,287.0	3.0	0.029%	0	0.000%
		918.709	318,977.515						

Table 5.7: Computing STE36-1 using AKV'6 with `numcut` = 350

clearly outperforms AKV. Therefore it can be concluded that the changes made to Strategies 7 and 8 to achieve a smoother `vioRHS` reduction process are important for larger instances, but these new changes make little difference for the medium-sized instances. We can also conclude that Strategy 8 is the best-performing strategy for the most part with some exception that Strategy 7 may run with fewer iterations and hence result in shorter run time.

AKV' 7		numcut	700						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			4.957						
		689.957	694.914	9,851.0	10,375.0				
	-0.4000	2,299.315	2,994.229						
		1,829.160	4,823.389	9,978.5	10,337.0	127.5	1.294%	358.5	3.468%
1.01	-0.4040	933.937	5,757.326			130			
	56	2,315.416	8,072.742	10,112.0	10,337.0	133.5	1.338%	225	2.177%
0.76	-0.3080	1,007.119	9,079.862			100			
		2,885.483	11,965.345	10,185.0	10,289.0	73.0	0.722%	104	1.011%
1.01	-0.3111	933.722	12,899.067			115			
	757	3,722.951	16,622.017	10,222.5	10,289.0	37.5	0.368%	66.5	0.646%
0.68	-0.2111	998.604	17,620.621			117			
		4,457.917	22,078.538	10,245.0	10,289.0	22.5	0.220%	44	0.428%
1.01	-0.2132	934.226	23,012.764			267			
	162	4,960.805	27,973.569	10,259.5	10,289.0	14.5	0.142%	29.5	0.287%
0.54	-0.1154	5,846.909	33,820.478			93			
		6,085.285	39,905.763	10,274.0	10,289.0	14.5	0.141%	15	0.146%
1.00	-0.1154	940.764	40,846.527			122			
	155	7,199.823	48,046.350	10,283.5	10,289.0	9.5	0.092%	5.5	0.053%
0.50	-0.0577	1,184.223	49,230.573			38			
		8,235.175	57,465.747	10,285.5	10,289.0	2.0	0.019%	3.5	0.034%
1.00	-0.0577	937.913	58,403.661			3			
	6	9,563.459	67,967.120	10,286.5	10,287.0	1.0	0.010%	0.5	0.005%
0.50	-0.0288	1,009.165	68,976.285			0			
		12,503.663	81,479.948	10,287.0	10,287.0	0.5	0.005%	0	0.000%
		861.375	82,341.323						

Table 5.8: Computing STE36-1 using AKV'7 with `numcut` = 700

AKV' 8		numcut	700						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			5.121						
		682.488	687.609	9,851.0	10,375.0				
	-0.4000	2,301.585	2,989.194						
		1,803.760	4,792.954	9,978.5	10,337.0	127.5	1.294%	358.5	3.468%
1.01	-0.4040	931.970	5,724.925			130			
	56	2,243.814	7,968.739	10,112.0	10,337.0	133.5	1.338%	225	2.177%
0.76	-0.3080	1,004.464	8,973.203			100			
		2,850.601	11,823.804	10,185.0	10,289.0	73.0	0.722%	104	1.011%
1.01	-0.3111	932.001	12,755.806			115			
	757	3,666.407	16,422.212	10,222.5	10,289.0	37.5	0.368%	66.5	0.646%
0.68	-0.2122	995.139	17,417.352			115			
		4,446.420	21,863.772	10,245.0	10,289.0	22.5	0.220%	44	0.428%
1.00	-0.2132	931.241	22,795.013			267			
	162	4,927.869	27,722.882	10,259.5	10,289.0	14.5	0.142%	29.5	0.287%
0.61	-0.1292	1,632.809	29,355.691			93			
		6,001.647	35,357.338	10,274.0	10,289.0	14.5	0.141%	15	0.146%
1.00	-0.1292	933.422	36,290.760			94			
	58	7,109.356	43,400.116	10,283.5	10,289.0	9.5	0.092%	5.5	0.053%
0.60	-0.0775	1,004.931	44,405.047			7			
	15	8,143.423	52,548.470	10,285.5	10,289.0	2.0	0.019%	3.5	0.034%
0.60	-0.0465	2,503.475	55,051.945			3			
	61	9,425.184	64,477.129	10,286.5	10,287.0	1.0	0.010%	0.5	0.005%
0.60	-0.0279	1,008.171	65,485.299			0			
		12,457.810	77,943.109	10,287.0	10,287.0	0.5	0.005%	0	0.000%
		859.649	78,802.758						

Table 5.9: Computing STE36-1 using AKV'8 with `numcut` = 700

AKV' 7		numcut	350						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			5.421						
		685.469	690.891	9,851.0	10,375.0				
	-0.4000	2,237.915	2,928.806						
		1,500.656	4,429.462	9,954.5	10,289.0	103.5	1.051%	334.5	3.251%
1.01	-0.4040	934.476	5,363.938			57			
		1,606.885	6,970.823	10,094.0	10,289.0	139.5	1.401%	195	1.895%
1.01	-0.4080	933.453	7,904.277			47			
	192	2,102.356	10,006.633	10,119.5	10,289.0	25.5	0.253%	169.5	1.647%
0.75	-0.3080	934.248	10,940.881			30			
		2,279.010	13,219.891	10,160.5	10,289.0	41.0	0.405%	128.5	1.249%
1.01	-0.3111	933.263	14,153.154			66			
		2,437.189	16,590.343	10,183.5	10,289.0	23.0	0.226%	105.5	1.025%
1.01	-0.3142	933.546	17,523.890			28			
	360	2,845.882	20,369.772	10,197.0	10,289.0	13.5	0.133%	92	0.894%
0.68	-0.2142	1,119.608	21,489.380			63			
		2,894.791	24,384.171	10,228.5	10,289.0	31.5	0.309%	60.5	0.588%
1.01	-0.2164	934.101	25,318.271			66			
		3,235.339	28,553.611	10,248.5	10,289.0	20.0	0.196%	40.5	0.394%
1.01	-0.2185	933.352	29,486.963			115			
	237	3,493.538	32,980.501	10,261.0	10,289.0	12.5	0.122%	28	0.272%
0.54	-0.1185	1,461.996	34,442.497			93			
		3,568.726	38,011.223	10,275.5	10,289.0	14.5	0.141%	13.5	0.131%
1.00	-0.1185	933.781	38,945.004			24			
		4,024.675	42,969.680	10,281.5	10,287.0	6.0	0.058%	5.5	0.053%
1.00	-0.1185	934.212	43,903.892			20			
	246	4,375.850	48,279.742	10,284.0	10,287.0	2.5	0.024%	3	0.029%
0.50	-0.0593	991.134	49,270.876			2			
		4,546.963	53,817.839	10,287.0	10,287.0	3.0	0.029%	0	0.000%
		863.165	54,681.004						

Table 5.10: Computing STE36-1 using AKV'7 with `numcut` = 350

AKV' 8		numcut	350						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			5.247						
		692.121	697.368	9,851.0	10,375.0				
	-0.4000	2,333.340	3,030.708						
		1,455.234	4,485.941	9,954.5	10,289.0	103.5	1.051%	334.5	3.251%
1.01	-0.4040	941.723	5,427.664			57			
		1,594.390	7,022.054	10,094.0	10,289.0	139.5	1.401%	195	1.895%
1.01	-0.4080	945.798	7,967.852			47			
	192	2,033.923	10,001.776	10,119.5	10,289.0	25.5	0.253%	169.5	1.647%
0.75	-0.3080	940.591	10,942.367			30			
		2,274.035	13,216.402	10,160.5	10,289.0	41.0	0.405%	128.5	1.249%
1.01	-0.3111	939.828	14,156.230			66			
		2,490.125	16,646.354	10,183.5	10,289.0	23.0	0.226%	105.5	1.025%
1.01	-0.3142	939.452	17,585.806			28			
	360	2,854.408	20,440.214	10,197.0	10,289.0	13.5	0.133%	92	0.894%
0.68	-0.2142	1,130.308	21,570.522			63			
		2,939.065	24,509.586	10,228.5	10,289.0	31.5	0.309%	60.5	0.588%
1.01	-0.2164	940.114	25,449.700			66			
		3,226.929	28,676.629	10,248.5	10,289.0	20.0	0.196%	40.5	0.394%
1.01	-0.2185	939.641	29,616.270			115			
	237	3,480.754	33,097.023	10,261.0	10,289.0	12.5	0.122%	28	0.272%
0.60	-0.1311	1,041.800	34,138.824			93			
		3,604.208	37,743.032	10,275.5	10,289.0	14.5	0.141%	13.5	0.131%
1.00	-0.1311	946.822	38,689.854			24			
	51	4,106.633	42,796.488	10,281.5	10,287.0	6.0	0.058%	5.5	0.053%
0.60	-0.0787	1,057.901	43,854.388			20			
		4,394.598	48,248.986	10,284.5	10,287.0	3.0	0.029%	2.5	0.024%
1.00	-0.0787	946.813	49,195.799			2			
	1	4,768.510	53,964.309	10,286.5	10,287.0	2.0	0.019%	0.5	0.005%
0.60	-0.0472	1,019.415	54,983.725			5			
		5,188.191	60,171.915	10,287.0	10,287.0	0.5	0.005%	0	0.000%
		873.862	61,045.778						

Table 5.11: Computing STE36-1 using AKV'8 with `numcut` = 350

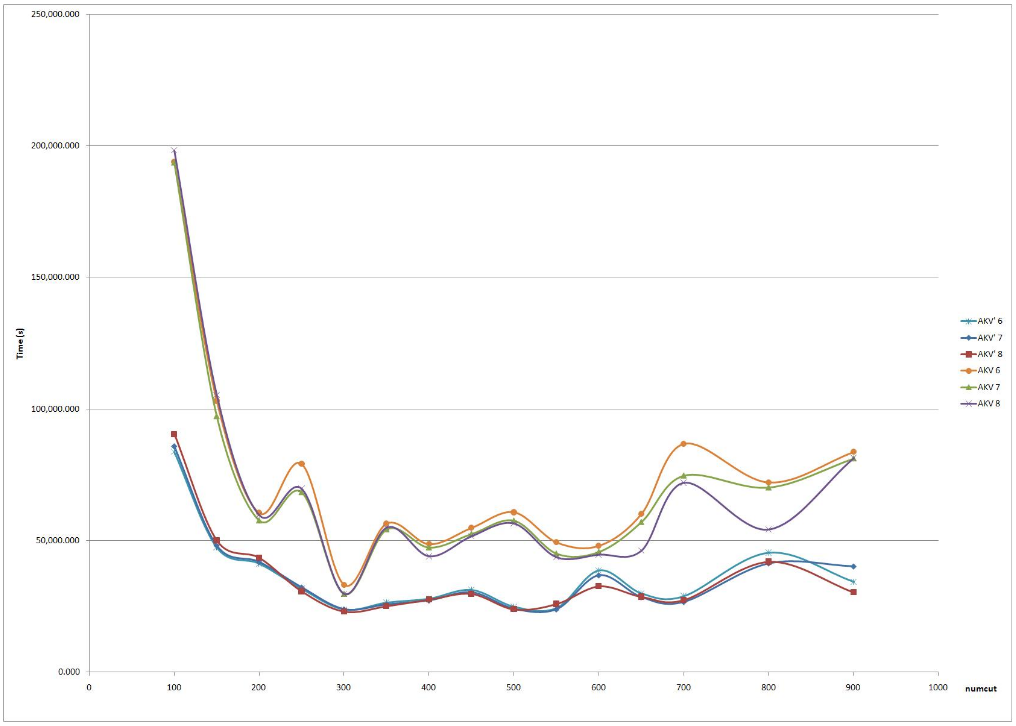

Figure 5.6: Comparison of Strategies 6, 7, and 8 for AV25-2

5.2.2 Results from CSDP in Parallel Computing

As the size of instances increased, it was observed that the optimization time also increased quickly (e.g. HeKu30 in Table 5.4). Therefore, other options were explored to facilitate solving large instances. So far, other than the lower bound experiment in Chapter 3, the experimental results were obtained by using SDPT3 version 4.0 [40] on a 2.0GHz Dual Opteron with 16Gb of RAM. The other option is to use CSDP version 5.0 [8] with the ATLAS library of routine [41] on a Sun Fire V890 8*1.2GHz with 64Gb of RAM. Running in parallel using 8 CPUs allows the optimization run to speed up. But since different computers were used to run these two different solvers, the configuration of Matlab may also affect the overall performance of each options.

Table 5.12 presents the overall computing time to solve HeKu20 by AKV'6 using the two solvers on different computers. In this example, the computing time actually increases as we switch the solver to CSDP. Table 5.13 details the first comparison set in Table 5.12 when `numcut` = 200. Table 5.13 shows that the new

`numcut`	CPU time (sec)	
	SDPT3	CSDP
200	1,574	2,155
300	1,470	1,884
500	1,495	1,759

Table 5.12: Quick comparison of SDPT3 and CSDP using AKV'6 solving HeKu20

AKV' 6	SDPT3	numcut	200			AKV' 6	CSDP	numcut	200		
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk
			0.000						0.000		
			1.416						0.979		
		25.951	27.367	15,203.0	15,549.0			44.969	45.949	15,203.0	15549
	-0.4	47.719	75.086				-0.4	58.434	104.383		
		48.272	123.358	15,297.0	15,549.0			87.573	191.955	15297.0	15549
1.01	-0.404	18.518	141.876			1.01	-0.404	40.501	232.456		
		56.347	198.223	15,366.0	15,549.0			97.311	329.767	15366.0	15549
1.01	-0.408	18.583	216.805			1.01	-0.408	40.963	370.730		
	125	57.639	274.445	15,401.0	15,549.0		125	101.322	472.052	15401.0	15549
0.76	-0.30804	18.886	293.331			0.76	-0.30804	41.202	513.253		
		85.428	378.759	15,443.5	15,549.0			113.921	627.174	15443.5	15549
1.01	-0.3111	18.504	397.263			1.01	-0.3111	40.452	667.626		
	26	95.323	492.586	15,479.5	15,549.0		26	125.655	793.281	15479.5	15549
0.69	-0.21423	22.321	514.907			0.69	-0.2142	45.278	838.559		
		102.449	617.357	15,521.5	15,549.0			133.408	971.967	15521.5	15549
1.00	-0.21423	18.285	635.642			1.00	-0.2142	40.376	1,012.343		
		130.684	766.326	15,529.5	15,549.0			153.312	1,165.655	15529.5	15549
1.00	-0.2143	18.282	784.609			1.00	-0.2142	40.198	1,205.853		
	104	140.124	924.732	15,537.0	15,549.0		104	192.949	1,398.803	15537.0	15549
0.53	-0.11423	18.903	943.635			0.53	-0.1142	40.763	1,439.566		
		162.144	1,105.779	15,544.5	15,549.0			171.589	1,611.154	15544.5	15549
1.00	-0.11423	18.432	1,124.211			1.00	-0.1142	40.278	1,651.433		
		185.046	1,309.257	15,548.5	15,549.0		206	201.274	1,852.707	15548.5	15549
0.25	-0.02856	22.279	1,331.536			0.25	-0.02856	45.171	1,897.878		
		220.313	1,551.849	15,549.0	15,549.0			222.006	2,119.884	15549.0	15549
		21.804	1,573.653					35.206	2,155.091		

Table 5.13: Comparing SDPT3 and CSDP by using AKV'6 to solve HeKu20

computing setup results in longer cut-searching time, which is due to the difference in the Matlab computing environment in different computers. Also, the optimization time is initially longer than the first computing option, but the difference in time slowly decreases as the sub-problem size increases.

Table 5.14 compares the two computing options by solving two bigger instances using Strategies 6, 7, and 8 at `numcut` = 500. We can observe that as the problem instance becomes larger and more complicated, the new computing option using CSDP actually pays off. Table 5.15 illustrates the time breakdown of the AKV'7 comparison set that solves STE36-1 at `numcut` = 500 in Table 5.14. This time breakdown shows that the cut-searching time in the CSDP option is approximately

Strategy	AV25-2		STE36-1	
	SDPT3	CSDP	SDPT3	CSDP
AKV'6	24,845	20,557	N/A	N/A
AKV'7	24,071	19,913	60,006	53,246
AKV'8	23,940	20,040	69,219	59,925

Table 5.14: Quick Comparison of SDPT3 and CSDP solving AV25-2 and STE36-1 at `numcut` = 500

AKV' 7	SDPT3	numcut	500			AKV' 7	CSDP	numcut	500		
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk
			0.000						0.000		
			4.069						7.503		
		637.006	641.075	9,851.0	10,375.0			486.193	493.695	9,851.0	10,375.0
	-0.4000	2,229.954	2,871.029				-0.4000	3,040.174	3,533.869		
		1,607.718	4,478.748	9,966.0	10,340.0			976.875	4,510.744	9,966.0	10,340.0
1.01	-0.4040	933.865	5,412.613			1.01	-0.4040	2,218.396	6,729.139		
	188	1,857.940	7,270.553	10,109.5	10,340.0		188	1,107.976	7,837.115	10,109.5	10,340.0
0.76	-0.3080	1,004.927	8,275.480			0.76	-0.3080	2,381.047	10,218.162		
		2,605.648	10,881.128	10,154.5	10,340.0			1,330.034	11,548.196	10,154.5	10,340.0
1.01	-0.3111	933.552	11,814.680			1.01	-0.3111	2,220.519	13,768.715		
	235	2,916.899	14,731.579	10,197.0	10,289.0		235	1,481.445	15,250.160	10,197.0	10,289.0
0.69	-0.2142	1,850.761	16,582.339			0.69	-0.2142	2,883.835	18,133.995		
		3,277.722	19,860.061	10,219.0	10,289.0			1,765.771	19,899.766	10,219.0	10,289.0
1.01	-0.2164	934.813	20,794.874			1.01	-0.2164	2,231.638	22,131.404		
		3,815.136	24,610.010	10,252.5	10,289.0			1,929.845	24,061.249	10,252.5	10,289.0
1.01	-0.2185	933.997	25,544.007			1.01	-0.2185	2,228.020	26,289.270		
		4,272.654	29,816.661	10,259.5	10,289.0			2,199.229	28,488.499	10,259.5	10,289.0
0.80	-0.1748	934.333	30,750.994			0.80	-0.1748	2,219.776	30,708.275		
	39	5,020.658	35,771.652	10,273.5	10,289.0		39	2,495.636	33,203.911	10,273.5	10,289.0
0.50	-0.0874	1,888.539	37,660.191			0.50	-0.0874	2,880.150	36,084.061		
		5,582.277	43,242.467	10,283.0	10,289.0			2,926.767	39,010.828	10,283.0	10,289.0
1.00	-0.0874	935.778	44,178.245			1.00	-0.0874	2,219.693	41,230.520		
	5	6,493.094	50,671.339	10,286.0	10,289.0		5	3,507.704	44,738.224	10,286.0	10,289.0
0.50	-0.0437	1,006.521	51,677.860			0.50	-0.0437	2,387.877	47,126.100		
		7,464.867	59,142.726	10,287.0	10,287.0			4,071.114	51,197.215	10,287.0	10,287.0
		863.190	60,005.917					2,048.290	53,245.505		

Table 5.15: Comparing SDPT3 and CSDP by using AKV'7 to solve STE36-1

more than double of the original setup that uses SDPT3. This discrepancy is again due to the difference of the Matlab computing environment in different computers. However, the optimization time in parallel computing is much smaller than the original setup. In fact, as more cuts are added and the sub-problem becomes bigger, the payoff becomes more significant.

Other than comparing the two computing options, Table 5.14 also shows that Strategies 6, 7, and 8 have similar performance in solving AV25-2 using the CSDP setup at `numcut` = 500. This again verifies that the fine-tuning changes made to Strategies 7 and 8 does not show any effect for medium-sized instances as concluded

AKV' 7		numcut	600						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			7.373						
		486.200	493.573	9,851.0	10,375.0				
	-0.4000	2,955.968	3,449.541						
		1,008.819	4,458.360	9,971.0	10,350.0	120.0	1.218%	379	3.662%
1.01	-0.4040	2,219.219	6,677.579			119			
	101	1,208.594	7,886.173	10,111.5	10,337.0	140.5	1.409%	225.5	2.181%
0.76	-0.3080	2,383.280	10,269.453			67			
		1,482.167	11,751.620	10,165.0	10,337.0	53.5	0.529%	172	1.664%
1.01	-0.3111	2,221.981	13,973.600			132			
	166	1,731.682	15,705.282	10,209.5	10,287.0	44.5	0.438%	77.5	0.753%
0.69	-0.2142	2,690.493	18,395.775			173			
		2,100.305	20,496.080	10,230.5	10,287.0	21.0	0.206%	56.5	0.549%
1.01	-0.2164	2,226.349	22,722.429			210			
		2,157.565	24,879.994	10,252.5	10,287.0	22.0	0.215%	34.5	0.335%
1.01	-0.2185	2,221.656	27,101.650			69			
		2,741.179	29,842.828	10,257.5	10,287.0	5.0	0.049%	29.5	0.287%
0.80	-0.1748	2,222.149	32,064.978			45			
	108	3,168.396	35,233.374	10,275.0	10,287.0	17.5	0.171%	12	0.117%
0.50	-0.0874	2,720.285	37,953.659			118			
		3,496.002	41,449.661	10,283.0	10,287.0	8.0	0.078%	4	0.039%
1.00	-0.0874	2,222.658	43,672.319			17			
	283	4,121.967	47,794.286	10,285.0	10,287.0	2.0	0.019%	2	0.019%
0.50	-0.0437	24,644.837	72,439.123			47			
		5,273.759	77,712.882	10,287.0	10,287.0	2.0	0.019%	0	0.000%
		2,054.767	79,767.649						

Table 5.16: Computing STE36-1 using AKV'7 with `numcut` = 600

from the previous chapter. However, as we move on to l arger instances such as STE36-1, Table 5.14 shows more performance deviation between strategies. In the case of `numcut` = 500 in Table 5.14, Strategy 8 has a slighly longer computation time than Strategy 7 in solving STE36-1, which is also due to the requirement of one more iteration in Strategy 8 as a result of a smoother `vioRHS` reduction process. Tables 5.16 and 5.17 present a contrary example when Strategy 8 outperforms Strategy 7. While AKV'7 requires 79,768 seconds to complete the run of solving STE36-1 at `numcut` = 600, AKV'8 only needs 57,437 seconds, whcich results in a percentage difference of 38.9%. As illustrated in the circled time period Table 5.16, this considerable difference is again due to an aggressive reduction in `vioRHS`. Therefore, this contrary example verifies the earlier finding in Section 5.2.1, which concludes that Strategy 8 is preferred over Strategy 7, even though it does not always outperforms Strategy 7. When Strategy 8 takes longer time than Strategy 7 to complete a run, the difference in time is usually relatively small, and it is usually due to the need of one more iteration in Strategy 8 as a result of a smoother `vioRHS` reduction process. However, when Strategy 8 outperforms Strategy 7, the difference is usually more significant. Moreover, it was already observed

AKV' 8		numcut	600						
Change in vioRHS	vioRHS	Duration (s)	t (s)	zlb	zbk	Gap (old-new zlb)	% diff.	Gap (zlb-zbk)	% diff.
			0.000						
			7.520						
		486.399	493.918	9,851.0	10,375.0				
	-0.4000	3,106.002	3,599.920						
		1,010.283	4,610.203	9,971.0	10,350.0	120.0	1.218%	379	3.662%
1.01	-0.4040	2,216.519	6,826.722			119			
	101	1,209.513	8,036.235	10,111.5	10,337.0	140.5	1.409%	225.5	2.181%
0.76	-0.3080	2,388.136	10,424.371			67			
		1,485.522	11,909.893	10,165.0	10,337.0	53.5	0.529%	172	1.664%
1.01	-0.3111	2,218.817	14,128.710			132			
	166	1,733.867	15,862.577	10,209.5	10,287.0	44.5	0.438%	77.5	0.753%
0.69	-0.2142	2,698.560	18,561.137			173			
		2,105.107	20,666.244	10,230.5	10,287.0	21.0	0.206%	56.5	0.549%
1.01	-0.2164	2,228.540	22,894.784			210			
		2,163.327	25,058.111	10,252.5	10,287.0	22.0	0.215%	34.5	0.335%
1.01	-0.2185	2,219.865	27,277.976			69			
		2,755.308	30,033.285	10,257.5	10,287.0	5.0	0.049%	29.5	0.287%
0.80	-0.1748	2,221.477	32,254.762			45			
	108	3,182.776	35,437.537	10,275.0	10,287.0	17.5	0.171%	12	0.117%
0.60	-0.1049	2,391.629	37,829.167			118			
		3,508.929	41,338.095	10,283.0	10,287.0	8.0	0.078%	4	0.039%
1.00	-0.1049	2,221.673	43,559.769			17			
	54	4,133.460	47,693.229	10,285.0	10,287.0	2.0	0.019%	2	0.019%
0.60	-0.0629	2,397.643	50,090.872			47			
		5,290.557	55,381.430	10,287.0	10,287.0	2.0	0.019%	0	0.000%
		2,055.509	57,436.938						

Table 5.17: Computing STE36-1 using AKV'8 with `numcut` = 600

that Strategy 8 has a better overall performance over Strategy 7 in HeKu30 and AV25-2 from Figures 5.5 and 5.6. This justifies the use of Strategy 8 to solve the new large instances in the next section.

Finally, since the quick comparison between the two computing options show that the CSDP setup runs faster as the instance size increases, it is decided that the new instances with $n = 36$ will be solved by the new CSDP computing setup. Furthermore, as was explained in Section 4.2.2, smaller instances run faster with smaller `numcut`. On the contrary, larger instances should be executed with higher `numcut` due to the fact that every additional iteration requires a great deal of computing time. Furthermore, by using high `numcut` we can exploit the advantage of parallel computing as the sub-problem size increases. Figure 5.7 also shows the trend of computing time with the effect of varying `numcut` for Strategy 7 to solve STE36-1. The computing time fluctuates a lot for low `numcut` from 300 to 640. The fluctuation seems to ease off and go down in higher `numcut`. In fact, the lowest computing time is generated by `numcut` = 900. Therefore, a high `numcut` such as 900 is used to pursue the large instances in the next section.

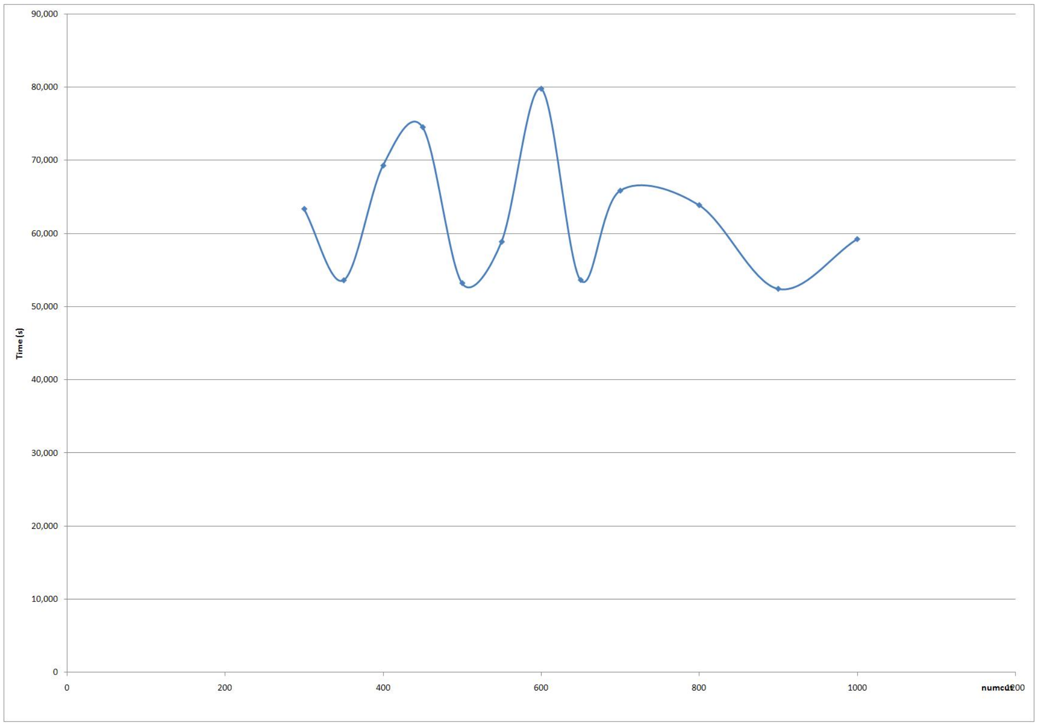

Figure 5.7: Effect of `numcut` on computing time of AKV'7 solving STE36-1

5.3 Solving New Large Instances

5.3.1 Results Analysis

The 36-facility instances are based on the QAP instance from Steinberg in [38] where the flow matrix is taken from [38] and the length vector is randomly generated. A total of nine STE instances were attempted, in which six of these were newly generated while three others (STE36-1, STE36-2, and STE36-3) already appeared in [6] where the new lower bounds were published. The complete listing of these instances can be found in Appendix C.

The instances STE36-2, STE36-3, and STE36-8 failed to reach optimality due to memory limitation in Matlab. Table 5.18 lists the six successful instances as well as the three failed instances, along with their optimal objective values and the total CPU run time when applicable. These results are obtained by running AKV'8 at `numcut` = 900 using the CSDP computing setup.

Instance	Generation Method	Optimal Solution	CPU time (sec)	Number of iterations
STE36-1	Vector of ones	10,287.0	62,754	11
STE36-2	U(1,37)	Not found	Out of memory	-
STE36-3	U(1,19)	Not found	Out of memory	-
STE36-6	U(1,3)	19,186.5	243,535	18
STE36-7	U(1,4)	25,055.0	124,852	14
STE36-8	U(1,5)	Not found	Out of memory	-
STE36-9	N(2,0.25)	20,203.5	56,675	11
STE36-10	N(3,0.25)	29,846.0	83,505	12
STE36-11	N(4,0.25)	41,240.0	104,091	13

Table 5.18: Results of the STE-series instances using AKV'8 and `numcut` = 900

5.3.2 Preliminary Analysis on Length Vector

Other than the optimization findings, Table 5.18 also shows how the lenght vector of these new 36-facility instances were created. The notation U(1,37) represents uniform distribution between 1 and 37, while N(2,0.25) denotes normal distribution with mean of 2 and variance of 0.25. The instances STE36-2, STE36-3, and STE36-8 failed to reach optimality due to memory limitation in Matlab, and they all have higher variations in the length elements. Therefore, it can be observed that as the degree of variance of the length elements increases, the problem structure becomes more complicated, and hence harder to solve. Furthermore, while STE36-6 and STE36-7 are solvable, STE36-8 has larger variance in the length vector and could not be solved. However, although STE36-6 is created by U(1,3), which is expected to be simpler than U(1,4), STE36-6 requires a longer run time than STE36-7. It is possibly due to the interaction between the length allocation to the given frequency of each department.

Another interesting fact is that the instance created by normal distribution exhibits shorter CPU run time. Since normal distribution has a characteristic bell shape with more elements falling in the range of the mean, it is expected to have less "jumps" between the length elements, and hence easier to solve. This prediction can be observed in Table 5.18. Using a fixed variance that controls the spread of the length elements enables the control of the difficulty level of the instances. Therefore we conjecture that the length vector plays an importaant role in the solvability of an instance. This interesting observation is recommended as the subject of future research. Meanwhile, it should be noted that Amaral's new instances of size $n = 35$ are not presented in his new paper [2],and thus we have not yet been able to experiment with them. However, it would be interesting to analyze the new 35-facility instances in order to fully understand the performance of his new lower bound in [2]

Chapter 6

Conclusions and Future Research

In this thesis, a new matrix-based model AKV' is presented. It is created based on AKV from [4], but it reduces the number of linear constraints from $O(n^3)$ to $O(n^2)$. AKV' relaxation can find a lower bound in a shorter computing time than AKV relaxation with only a minor penalty of slight deterioration in the lower bound. AKV' is observed to pay off as the instance size increases.

Six cutting plane strategies are proposed for the medium-sized SRFLP instances. The general approach is to smooth the `vioRHS` reduction process while preventing premature termination. Three instances of different characteristics are used to analyze the cutting plane strategies. To solve a small medium-sized instance like HeKu20 using a similar computing setup as described in this thesis, it is recommended to use the AKV' relaxation combined with Strategy 6 at low `numcut` in the range of 50 to 300. When approaching a medium-sized linear ordering problem like AV25-1, it is advised to explore both relaxations, since the distinction between them is not prominent. The readers can also refer to [16] for other cutting plane algorithms targetted to linear ordering problem. To solve a medium-sized instance like AV25-1 using the proposed cutting plane strategy, it is recommended to apply Strategy 6 with medium-range `numcut` such as between 350 to 500. For the general medium-sized SRFLPs like AV25-2, it is recommended to utilize AKV'6 with medium-range `numcut` such as between 300 to 550. However, higher-range `numcut` is strongly recommended to explore other medium-sized or larger SRFLPs.

Another two cutting plane strategies are proposed for large instances to achieve a smoother `vioRHS` reduction process. The combination of Strategy 8 with AKV' relaxation in high `numcut` is capable of solving six new instances of size $n = 36$, which is higher than the published results in literature. We also point out an interesting fact about the the length vector, where we conjecture that the length vector plays an important role in the solvability of an instance.

The interesting observation regarding the length vector analysis is recommended as the subject of future research. By investigating the effect of the length vector on the solvability and the computing time of the SRFLP instances, one can better understand how good different models are in literature. It can also facilitates a more thorough and fair comparison between optimization methods.

Another interesting topic for future research is to investigate a more in-depth comparison between the two SDP solvers, namely SDPT3 and CSDP. A fair comparison should be made within the same computing environment, and it should be able to help an analyst to make a better decision in choosing a suitable solver.

Just recently Amaral proposed a new lower bound approach, which is capable of solving SRFLPs of size $n = 35$. It would be interesting to compare the performance of Amaral's new model to the AKV and AKV' relaxations. Furthermore, it is also interesting to analyze the 35-facility instances that he used in [2] in order to fairly gauge the ability of his new lower bound approach.

Since the proposed methodology of combining Strategy 8 with AKV' relaxation reaches the memory limitation in Matlab, it would be very interesting to look into the possibility of translating the code to run in C in conjunction of CSDP as future research. By running on a different platform, the memory limit may be different, and consequently larger instances may be solved.

Finally, after a more thorough study of the SRFLP, the future research may extend from single-row to multi-row facility layout problem. It is likely that the result from the SRFLP may shed some light to the multi-row problems.

Appendix A

Matlab Code for 2-Opt

```
function [x] = twoopt(F, l, x)
n = length(l); % number of departments
xtemp = x;
bestCost = 0;
cost = 0;
xold = zeros(1,n);
bestCost = objfunction(F,l,x);
while x ~= xold
    xold = x;
    for a = 1:(n-1) % check swap b/w ith and jth positions
        for b = (a+1):n
            xtemp(a) = x(b);
            xtemp(b) = x(a);
            cost = objfunction(F,l,xtemp);
            if cost >= bestCost  % keep the same
                xtemp = x;
            else                 % swap and update best cost
                x = xtemp;
                bestCost = cost;
            end
        end
    end
end
```

Appendix B

Matlab Code for AKV Heuristic

```
function [x, xbk, zbk] = AKVheur(X, xbk, zbk, F, l);
n = length(l);
zub = 9999999;  %a large number as upper bound to begin with
for a = 1 : nchoosek(n,2),    %check thru each row of X*
    R = zeros(n);
    for i=1:n-1,    % Calculate Rij in matrix form, set Rii = 0
        for j=i+1:n,
            Xcol = (j-1)*(j-2)/2+i;
            R(i,j)=X(a,Xcol);
            R(j,i)=-1*X(a,Xcol);
        end
    end
    for i=1:n,  % calculate p
        P(i) = (sum(R(i,:))+n+1)/2; %Rii = 0 so no effect
    end
    [Y,x_temp] = sort(P,'descend');
    if objfunction(F,l,x_temp) < zub,   %zub = best obj value by comparing each row
        zub = objfunction(F,l,x_temp);  % zub not used
        x = x_temp;
    end
    [x_temp] = twoopt(F, l, x_temp);
    if objfunction(F,l,x_temp) < zbk,   %zbk = best global obj value
        xbk = x_temp;     % Update xbk
```

```
        zbk = objfunction(F,l,x_temp);      % Update zbk
        display('zbk updated at'); a
    end
end
```

Appendix C

Complete Listings of SRFLP Instances Used

C.1 HeKu20

```
l = [ 20 3 9 3 7 3 7 5 9 6 5 3 9 3 7 3 7 5 9 6]

F = [ 0  0  5  0  5  2 10  3  1  5  5  5  0  0  5  4  4  0  0  1; ...
      0  0  3 10  5  1  5  1  2  4  2  5  0 10 10  3  0  5 10  5; ...
      5  3  0  2  0  5  2  4  4  5  0  0  0  5  1  0  0  5  0  0; ...
      0 10  2  0  1  0  5  2  1  0 10  2  2  0  2  1  5  2  5  5; ...
      5  5  0  1  0  5  6  5  2  5  2  0  5  1  1  1  5  2  5  1; ...
      2  1  5  0  5  0  5  2  1  6  0  0 10  0  2  0  1  0  1  5; ...
     10  5  2  5  6  5  0  0  0  0  5 10  2  2  5  1  2  1  0 10; ...
      3  1  4  2  5  2  0  0  1  1 10 10  2  0 10  2  5  2  2 10; ...
      1  2  4  1  2  1  0  1  0  2  0  3  5  5  0  5  0  0  0  2; ...
      5  4  5  0  5  6  0  1  2  0  5  5  0  5  1  0  0  5  5  2; ...
      5  2  0 10  2  0  5 10  0  5  0  5  2  5  1 10  0  2  2  5; ...
      5  5  0  2  0  0 10 10  3  5  5  0  2 10  5  0  1  1  2  5; ...
      0  0  0  2  5 10  2  2  5  0  2  2  0  2  2  1  0  0  0  5; ...
      0 10  5  0  1  0  2  0  5  5  5 10  2  0  5  5  1  5  5  0; ...
      5 10  1  2  1  2  5 10  0  1  1  5  2  5  0  3  0  5 10 10; ...
      4  3  0  1  1  0  1  2  5  0 10  0  1  5  3  0  0  0  2  0; ...
      4  0  0  5  5  1  2  5  0  0  0  1  0  1  0  0  0  5  2  0; ...
      0  5  5  2  2  0  1  2  0  5  2  1  0  5  5  0  5  0  1  1; ...
      0 10  0  5  5  1  0  2  0  5  2  2  0  5 10  2  2  1  0  6; ...
      1  5  0  5  1  5 10 10  2  2  5  5  5  0 10  0  0  1  6  0]
```

C.2 AV25 Instances

The AV25 instances have the same flow matrix F as listed below.

```
F = [0   3   2   0   0  10   5   0   5   2   0   0   2   0   5 ...
     3   0   1  10   0   2   1   1   1   0; ...
     3   0   4   0  10   0   0   2   2   1   5   0   0   0   0 ...
     0   1   6   1   0   2   2   5   1  10; ...
     2   4   0   3   4   5   5   5   1   4   0   4   0   4   0 ...
     3   2   5   5   2   0   0   3   1   0; ...
     0   0   3   0   0   0   2   2   0   6   2   5   2   5   1 ...
     1   1   2   2   4   2   0   2   2   5; ...
     0  10   4   0   0   2   0   0   0   0   0   0   0   0   2 ...
     0   0   2   0   5   0   2   1   0   2; ...
    10   0   5   0   2   0  10  10   5  10   6   0   0  10   2 ...
    10   1   5   5   2   5   0   2   0   1; ...
     5   0   5   2   0  10   0   1   3   5   0   0   2   4   5 ...
    10   6   0   5   5   5   0   5   5   0; ...
     0   2   5   2   0  10   1   0  10   2   5   2   0   3   0 ...
     0   0   4   0   5   0   5   2   2   5; ...
     5   2   1   0   0   5   3  10   0   5   6   0   1   5   5 ...
     5   2   3   5   0   2  10  10   1   5; ...
     2   1   4   6   0  10   5   2   5   0   0   1   2   1   0 ...
     0   0   0   6   6   4   5   3   2   2; ...
     0   5   0   2   0   6   0   5   6   0   0   2   0   4   2 ...
     1   0   6   2   1   5   0   0   1   5; ...
     0   0   4   5   0   0   0   2   0   1   2   0   2   1   0 ...
     3  10   0   0   4   0   0   4   2   5; ...
     2   0   0   2   0   0   2   0   1   2   0   2   0   4   5 ...
     0   1   0   5   0   0   0   5   1   1; ...
     0   0   4   5   0  10   4   3   5   1   4   1   4   0   0 ...
     0   2   2   0   2   5   0   5   2   5; ...
     5   0   0   1   2   2   5   0   5   0   2   0   5   0   0 ...
     2   0   0   0   6   3   5   0   0   5; ...
     3   0   3   1   0  10  10   0   5   0   1   3   0   0   2 ...
```

```
    0   0   5   5   1   5   2   1   2  10; ...
    0   1   2   1   0   1   6   0   2   0   0  10   1   2   0 ...
    0   0   5   2   1   1   5   6   5   5; ...
    1   6   5   2   2   5   0   4   3   0   6   0   0   2   0 ...
    5   5   0   4   0   0   0   0   5   0; ...
   10   1   5   2   0   5   5   0   5   6   2   0   5   0   0 ...
    5   2   4   0   5   4   4   5   0   2; ...
    0   0   2   4   5   2   5   5   0   6   1   4   0   2   6 ...
    1   1   0   5   0   4   4   1   0   2; ...
    2   2   0   2   0   5   5   0   2   4   5   0   0   5   3 ...
    5   1   0   4   4   0   1   0  10   1; ...
    1   2   0   0   2   0   0   5  10   5   0   0   0   0   5 ...
    2   5   0   4   4   1   0   0   0   0; ...
    1   5   3   2   1   2   5   2  10   3   0   4   5   5   0 ...
    1   6   0   5   1   0   0   0   0   0; ...
    1   1   1   2   0   0   5   2   1   2   1   2   1   2   0 ...
    2   5   5   0   0  10   0   0   0   2; ...
    0  10   0   5   2   1   0   5   5   2   5   5   1   5   5 ...
   10   5   0   2   2   1   0   0   2   0]
```

C.2.1 AV25-1

```
l = ones(1,25)
```

C.2.2 AV25-2

```
l = [15  4  10  8  14  12  8  1  13  8  10  13  15  12  4  7  15  15  7  14 ...
       2  6  13  1  3]
```

C.3 HeKu30

```
l = [3   9   3   7   3   7   5   9   6   5   3   9   3   7   3 ...
     7   5   9   6   5   3   9   3   7   3   7   5   9   6   5]

F = [0   3   2   0   0   2  10   5   0   5   2   5   0   0   2 ...
     0   5   6   3   0   1  10   0  10   2   1   1   1   0   1; ...
     3   0   4   0  10   4   0   0   2   2   1   0   5   0   0 ...
     0   0   2   0   1   6   1   0   1   2   2   5   1  10   5; ...
     2   4   0   3   4   0   5   5   5   1   4   1   0   4   0 ...
     4   0   6   3   2   5   5   2   1   0   0   3   1   0   2; ...
     0   0   3   0   0   0   0   2   2   0   6   0   2   5   2 ...
     5   1   1   1   1   2   2   4   0   2   0   2   2   5   5; ...
     0  10   4   0   0   5   2   0   0   0   0   2   0   0   0 ...
     0   2   1   0   0   2   0   5   1   0   2   1   0   2   1; ...
     2   4   0   0   5   0   1   2   2   1   4  10  10   2   5 ...
     5   0   5   0   0   0  10   0   0   0   4   0  10   1   1; ...
    10   0   5   0   2   1   0  10  10   5  10  10   6   0   0 ...
    10   2   1  10   1   5   5   2   3   5   0   2   0   1   3; ...
     5   0   5   2   0   2  10   0   1   3   5   0   0   0   2 ...
     4   5   2  10   6   0   5   5   2   5   0   5   5   0   2; ...
     0   2   5   2   0   2  10   1   0  10   2   1   5   2   0 ...
     3   0   2   0   0   4   0   5   2   0   5   2   2   5   2; ...
     5   2   1   0   0   1   5   3  10   0   5   5   6   0   1 ...
     5   5   0   5   2   3   5   0   5   2  10  10   1   5   2; ...
     2   1   4   6   0   4  10   5   2   5   0   0   0   1   2 ...
     1   0   2   0   0   0   6   6   0   4   5   3   2   2  10; ...
     5   0   1   0   2  10  10   0   1   5   0   0   5   5   2 ...
     0   0   0   0   2   0   4   5  10   1   0   0   0   0   1; ...
     0   5   0   2   0  10   6   0   5   6   0   5   0   2   0 ...
     4   2   2   1   0   6   2   1   5   5   0   0   1   5   5; ...
     0   0   4   5   0   2   0   0   2   0   1   5   2   0   2 ...
     1   0   5   3  10   0   0   4   2   0   0   4   2   5   5; ...
     2   0   0   2   0   5   0   2   0   1   2   2   0   2   0 ...
     4   5   1   0   1   0   5   0   2   0   0   5   1   1   0; ...
```

```
 0   0   4   5   0   5  10   4   3   5   1   0   4   1   4 ...
 0   0   3   0   2   2   0   2   0   5   0   5   2   5  10; ...
 5   0   0   1   2   0   2   5   0   5   0   0   2   0   5 ...
 0   0   2   2   0   0   0   6   5   3   5   0   0   5   1; ...
 6   2   6   1   1   5   1   2   2   0   2   0   2   5   1 ...
 3   2   0   5   1   2  10  10   4   0   0   5   0   0   0; ...
 3   0   3   1   0   0  10  10   0   5   0   0   1   3   0 ...
 0   2   5   0   0   5   5   1   0   5   2   1   2  10  10; ...
 0   1   2   1   0   0   1   6   0   2   0   2   0  10   1 ...
 2   0   1   0   0   5   2   1   3   1   5   6   5   5   3; ...
 1   6   5   2   2   0   5   0   4   3   0   0   6   0   0 ...
 2   0   2   5   5   0   4   0   1   0   0   0   5   0   0; ...
10   1   5   2   0  10   5   5   0   5   6   4   2   0   5 ...
 0   0  10   5   2   4   0   5   0   4   4   5   0   2   5; ...
 0   0   2   4   5   0   2   5   5   0   6   5   1   4   0 ...
 2   6  10   1   1   0   5   0   0   4   4   1   0   2   2; ...
10   1   1   0   1   0   3   2   2   5   0  10   5   2   2 ...
 0   5   4   0   3   1   0   0   0   5   5   0   1   0   0; ...
 2   2   0   2   0   0   5   5   0   2   4   1   5   0   0 ...
 5   3   0   5   1   0   4   4   5   0   1   0  10   1   0; ...
 1   2   0   0   2   4   0   0   5  10   5   0   0   0   0 ...
 0   5   0   2   5   0   4   4   5   1   0   0   0   0   0; ...
 1   5   3   2   1   0   2   5   2  10   3   0   0   4   5 ...
 5   0   5   1   6   0   5   1   0   0   0   0   0   0  10; ...
 1   1   1   2   0  10   0   5   2   1   2   0   1   2   1 ...
 2   0   0   2   5   5   0   0   1  10   0   0   0   2   2; ...
 0  10   0   5   2   1   1   0   5   5   2   0   5   5   1 ...
 5   5   0  10   5   0   2   2   0   1   0   0   2   0   2; ...
 1   5   2   5   1   1   3   2   2   2  10   1   5   5   0 ...
10   1   0  10   3   0   5   2   0   0   0  10   2   2   0]
```

C.4 STE36 Instances

The STE36 instances have the same flow matrix F as listed below.

```
F = [0    0    0    2    1    7    9    0    4   75 ...
     7   12   22    7    1    0    0    0    0   23 ...
     0    0    0    0    0    0    0    0    0    0 ...
     0    0    0    0    0    0; ...
     0    0    0    0    0    0    4   16    0    8 ...
     0    0   16    0    0    0    0    6    0    4 ...
     0    0    0    0    0    0    0    0    0    0 ...
     0    0    0    0    0    0; ...
     0    0    0    0    0    4   16   20    0    0 ...
     0    0   20    0    0    0    0    0    0    4 ...
     0    0    0    0    0    0    0    0    0    0 ...
     0    0    0    0    0    0; ...
     2    0    0    0   29    5   18   47   23    2 ...
     4    0   48    0    4    0    0    0    0   25 ...
     0    0    0    0    0    0    0    0    0    0 ...
     0    0    0    0    0    0; ...
     1    0    0   29    0   18   12   25    0    0 ...
     4    0   25    0    3    0    0    0    0   18 ...
     0    3    0    0    0    0    0    0    0    0 ...
     0    0    0    0    0    0; ...
     7    0    4    5   18    0    4    2    0    1 ...
    23    2   19    0    0    0    0    0    2   19 ...
     0    0    0    0    0    0    0    0    0    0 ...
     0    0    0    0    0    0; ...
     9    4   16   18   12    4    0    0   14   72 ...
     7    8   39    8   40    8    0    8    4    7 ...
     0    0    0    0    0    0    0   28    8    0 ...
     0    0    0    0    0    0; ...
     0   16   20   47   25    2    0    0   10   71 ...
     2    0    0    0    0    0    0   41    0    0 ...
     0    0    0    0    0    0    7    8    0    0 ...
```

```
  0     0     0     0     0     0; ...
  4     0     0    23     0     0    14    10     0    14 ...
  0     0    18     0     0     0     0     0     0     0 ...
  0     0     0     0     0     0     0     0     0     0 ...
  0     0     0     0     0     0; ...
 75     8     0     2     0     1    72    71    14     0 ...
 11     1    17     0     1     0     0    17     0    15 ...
  0     0     0     0     0     0     0     0     0     0 ...
  0     0     0     0     0     0; ...
  7     0     0     4     4    23     7     2     0    11 ...
  0   316    33     8     2     0     0     0     8    34 ...
  0     0     6     0     0     0    10     0     0     6 ...
  0     0     0     0     0     0; ...
 12     0     0     0     0     2     8     0     0     1 ...
316     0   157    25     4     0     0     1     0     0 ...
  0     0     0     0    22     0     1     0     0     0 ...
  0     0     0     0     0     0; ...
 22    16    20    48    25    19    39     0    18    17 ...
 33   157     0    11     6     0     0     6     0     5 ...
  8     3    10     0     0     0     9    11     2     0 ...
  0     1     0     0     0     0; ...
  7     0     0     0     0     0     8     0     0     0 ...
  8    25    11     0     3     0     0     1     1    21 ...
  0     1     0     2     0     0     5     0     0     3 ...
  2     5     5     4     0     0; ...
  1     0     0     4     3     0    40     0     0     1 ...
  2     4     6     3     0    19     0     2     2    12 ...
  0     0     0     0     0     0     0     7     3     0 ...
  0     0     0     0     0     0; ...
  0     0     0     0     0     0     8     0     0     0 ...
  0     0     0     0    19     0     0     6     0     1 ...
  0     0     0     0     0     0     0     0     0     0 ...
  0     0     0     0     0     0; ...
  0     0     0     0     0     0     0     0     0     0 ...
```

```
 0   0   0   0   0   0   0  40   0   0 ...
 0   0   0   0   0   0   0   0   0   0 ...
 0   0   0   0   0   0; ...
 0   6   0   0   0   0   8  41   0  17 ...
 0   1   6   1   2   6  40   0   0  26 ...
 0   0   0   0   0   0   0   0   0   0 ...
 0   0   0   0   0   0; ...
 0   0   0   0   0   2   4   0   0   0 ...
 8   0   0   1   2   0   0   0   0  13 ...
 9   0   7   0   0   0   0  27  16   3 ...
 0  20   0   4   0   0; ...
23   4   4  25  18  19   7   0   0  15 ...
34   0   5  21  12   1   0  26  13   0 ...
11   4  36   0   0   0  16  18   9  10 ...
 1  28   6   2   0   0; ...
 0   0   0   0   0   0   0   0   0   0 ...
 0   0   8   0   0   0   0   0   9  11 ...
 0  36   6   0   8   0   2   0   0   0 ...
 0   0   0   0   0   0; ...
 0   0   0   0   3   0   0   0   0   0 ...
 0   0   3   1   0   0   0   0   0   4 ...
36   0   0   0   0   0   4   0   0   0 ...
 0   0   0   0   0   0; ...
 0   0   0   0   0   0   0   0   0   0 ...
 6   0  10   0   0   0   0   0   7  36 ...
 6   0   0   0   0  12   9   0   0   0 ...
 0   0   0   0   0   0; ...
 0   0   0   0   0   0   0   0   0   0 ...
 0   0   0   2   0   0   0   0   0   0 ...
 0   0   0   0  26   0   5   0   0   0 ...
 0   0   0   0   0   0; ...
 0   0   0   0   0   0   0   0   0   0 ...
 0  22   0   0   0   0   0   0   0   0 ...
 8   0   0  26   0  35   2   0   0   0 ...
```

```
 0     0     0     0     0     0; ...
 0     0     0     0     0     0     0     0     0     0 ...
 0     0     0     0     0     0     0     0     0     0 ...
 0     0    12     0    35     0     4     0     0     0 ...
 0     0     0     0     0     0; ...
 0     0     0     0     0     0     0     7     0     0 ...
10     1     9     5     0     0     0     0     0    16 ...
 2     4     9     5     2     4     0     0     0     0 ...
 0     0     0     0     0     0; ...
 0     0     0     0     0     0    28     8     0     0 ...
 0     0    11     0     7     0     0     0    27    18 ...
 0     0     0     0     0     0     0     0    10    22 ...
 4     6     4    12     0     0; ...
 0     0     0     0     0     0     8     0     0     0 ...
 0     0     2     0     3     0     0     0    16     9 ...
 0     0     0     0     0     0     0    10     0    19 ...
12     0     0     0     0     0; ...
 0     0     0     0     0     0     0     0     0     0 ...
 6     0     0     3     0     0     0     0     3    10 ...
 0     0     0     0     0     0     0    22    19     0 ...
19     4     5     8     0     0; ...
 0     0     0     0     0     0     0     0     0     0 ...
 0     0     0     2     0     0     0     0     0     1 ...
 0     0     0     0     0     0     0     4    12    19 ...
 0     0     3    13     0     0; ...
 0     0     0     0     0     0     0     0     0     0 ...
 0     0     1     5     0     0     0     0    20    28 ...
 0     0     0     0     0     0     0     6     0     4 ...
 0     0    18    24     0     0; ...
 0     0     0     0     0     0     0     0     0     0 ...
 0     0     0     5     0     0     0     0     0     6 ...
 0     0     0     0     0     0     0     4     0     5 ...
 3    18     0    20     0     0; ...
 0     0     0     0     0     0     0     0     0     0 ...
```

```
     0     0     0     4     0     0     0     0     4     2 ...
     0     0     0     0     0     0     0    12     0     8 ...
    13    24    20     0     0     0; ...
     0     0     0     0     0     0     0     0     0     0 ...
     0     0     0     0     0     0     0     0     0     0 ...
     0     0     0     0     0     0     0     0     0     0 ...
     0     0     0     0     0     0; ...
     0     0     0     0     0     0     0     0     0     0 ...
     0     0     0     0     0     0     0     0     0     0 ...
     0     0     0     0     0     0     0     0     0     0 ...
     0     0     0     0     0     0]
```

C.4.1 STE36-1

```
l = ones(1,36)
```

C.4.2 STE36-2

```
l = [17  10  26  16  22   5  34  11   1  37  29  19  23   6  24   7 ...
     17   4  22  11  23  11  16  23  18  17   3  28   4  11  32  14 ...
     28  31   2  35 ]
```

C.4.3 STE36-3

```
l = [11  13   5   8  15  13   9  11  15   2  12   2   8   6  17 ...
      1  15  18  19  15   9  10   5  13   7  18  14   8  14   6 ...
      9  18  13   5  16  12]
```

C.4.4 STE36-6

```
l = [2   1   2   3   3   3   1   2   3   1   1   1   1   1   1   1   2   2 ...
     2   2   2   2   3   2   2   2   3   3   3   2   3   1   2   3   3   2]
```

C.4.5 STE36-7

```
l = [4  3  4  1  3  3  1  4  4  3  4  3  1  1  1  2  1  2 ...
     2  2  2  2  2  4  3  3  2  2  4  1  3  3  4  3  4  3]
```

C.4.6 STE36-8

```
l = [4  2  3  2  3  1  3  1  3  4  5  2  2  3  3  3  5  3 ...
     3  5  2  5  3  3  5  4  4  3  4  3  2  2  3  2  5  3 ]
```

C.4.7 STE36-9

```
l = [2  1  2  1  2  3  2  2  2  2  2  2  3  2  3  3  2  2 ...
     2  2  2  2  1  3  2  2  2  1  2  2  2  2  2  1  2  2]
```

C.4.8 STE36-10

```
l = [3  2  3  3  2  4  3  2  3  4  3  3  3  2  3  3  4  3 ...
     4  3  4  3  2  3  2  3  2  3  3  3  3  3  2  3  2  3]
```

C.4.9 STE36-11

```
l = [4  4  4  4  4  3  4  3  4  4  5  4  4  4  4  4  5  4 ...
     4  5  4  5  4  4  5  4  4  4  4  4  4  3  4  4  5  4]
```

References

[1] A.R.S. Amaral. On the exact solution of a facility layout problem. *Journal of Operational Research*, 173:508–518, 2006.

[2] A.R.S. Amaral. A new lower bound for the single row facility layout problem. *Discrete Applied Mathematics*, 2008. doi:10.1016/j.dam.2008.06.002.

[3] A.R.S. Amaral and A.N. Letchford. A polyhedral approach to the single row facility layout problem. Working paper. Department of Management Science, Lancaster University, 2008.

[4] M.F. Anjos, A. Kennings, and A. Vannelli. A semidefinite optimization approach for the single-row layout problem with unequal dimensions. *Discrete Optimization*, 2:113–122, 2005. doi:10.1016/j.disopt.2005.03.001.

[5] M.F. Anjos and A. Vannelli. Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. *Informs Journal on Computing*, 2008. doi: 10.1287/ijoc.1080.0270.

[6] M.F. Anjos and G. Yen. Provably near-optimal solutions for very large single-row facility layout problems. Working paper. Department of Management Sciences, University of Waterloo, 2008.

[7] K. Anstreicher, N. Brixius, J.-P. Goux, and J. Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming Series B*, 91:563–588, 2002.

[8] B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11/12(14):613–623, 1999.

[9] Y. A. Bozer, R. D. Meller, and S. J. Erlebacher. An improvement-type layout algorithm for single and multiple-floor facilities. *Management Science*, 40:918–932, 1994.

[10] I. Castillo and T. Sim. A spring-embedding approach for the facility layout problem. *Journal of the Operational Research Society*, 55:73–81, 2004.

[11] E. Çela. *The Quadratic Assignment Problem*, volume 1 of *Combinatorial Optimization*. Kluwer Academic Publishers, Dordrecht, 1998.

[12] M.M. Deza and M. Laurent. *Geometry of Cuts and Metrics*, volume 15 of *Algorithms and Combinatorics*. Springer Verlag, Berlin, 1995.

[13] H. Djellab and M. Gourgand. A new heuristic procedure for the single-row facility layout problem. *International Journal of Computer Integrated Manufacturing*, 14:270–280, 2001.

[14] Z. Drezner. A heuristic procedure for the layout of a large number of facilities. *Management Sciences*, 33:907–915, 1987.

[15] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42(6):1115–1145, 1995.

[16] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.

[17] K.M. Hall. An r-dimensional quadratic placement algorithm. *Management Sciences*, 17, 1970.

[18] S. S. Heragu. *Facilities Design*. iUniverse, second edition, 2006. Section 5.4.2.

[19] S.S. Heragu and A.S. Alfa. Experimental analysis of simulated annealing based algorithms for the layout problem. *European Journal of Operational Research*, 57:190–202, 1992.

[20] S.S. Heragu and A. Kusiak. Machine layout problem in flexible manufacturing systems. *Operations Research*, 36:258–268, 1988.

[21] S.S. Heragu and A. Kusiak. Efficient models for the facility layout problem. *European Journal of Operational Research*, 53:1–13, 1991.

[22] R.M. Karp and M. Held. Finite-state processes and dynamic programming. *SIAM Journal of Applied Mathematics*, 15:693–718, 1967.

[23] T. C. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

[24] K.R. Kumar, G.C. Hadjinicola, and T. Lin. A heuristic procedure for the single-row facility layout problem. *European Journal of Operational Research*, 87:65–73, 1995.

[25] A. Kusiak and S. S. Heragu. The facility layout problem. *European Journal of Operational Research*, 29:229–251, 1987.

[26] W. Liu and A. Vannelli. Generating lower bounds for the linear arrangement problem. *Discrete Applied Mathematics*, 59(2):137–151, 1995.

[27] E.M. Loiola, N.M.M. de Abreu, P.O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176:657–690, 2007.

[28] R.F. Love and J.Y. Wong. On solving a one-dimensional allocation problem with integer programming. *Information Processing and Operations Research*, 14(2):139–143, 1976.

[29] B.A. McCarl. *McCarl GAMS User Guide Version 22.4*. Texas A&M University and GAMS Development Corporation, 2007.

[30] R. Meller and K.-Y. Gau. The facility layout problem: Recent and emerging trends and perspectives. *Journal of Manufacturing Systems*, 15:351–366, 1996.

[31] F. Neghabat. An efficient equipment layout algorithm. *Operations Research*, 22:622–628, 1974.

[32] C.E. Nugent, T.E. Vollman, and J. Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, 16:150–173, 1968.

[33] J. Picard and M. Queyranne. On the one-dimensional space allocation problem. *Operations Research*, 29(2):371–391, 1981.

[34] G Reinelt. *The linear ordering problem: Algorithms and applications*, volume 8 of *Research and Exposition in Mathematics*. Heldermann Verlag, Berlin, 1985.

[35] D. Romero and A. Sánchez-Flores. Methods for the one-dimensional space allocation problem. *Computers and Operations Research*, 17:465–473, 1990.

[36] D.M. Simmons. One-dimensional space allocation: An ordering algorithm. *Operations Research*, 17:812–826, 1969.

[37] M. Solimanpur, P. Vrat, and R. Shankar. An ant algorithm for the single row layout problem in flexible manufacturing systems. *Computers & Operations Research*, 32:583–598, 2005.

[38] L. Steinberg. The backboard wiring problem: a placement algorithm. *SIAM Review*, 3:37–50, 1961.

[39] J.K. Suryanarayanan, B.L. Golden, and Q. Wang. A new heuristic for the linear placement problem. *Computers & Operations Research*, 18(3):255–262, 1991.

[40] R.H. Tütüncü, K.C. Toh, and M.J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming*, 95(2):189–217, 2003.

[41] R. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimizations of software and the atlas project. *Parallel Computing*, 27(1-2):3–35, 2001.