2012

Parameter Tuning for Optimization Software

RadhaShilpa Koripalli
Virginia Commonwealth University

Parameter Tuning for Optimization Software

Radha S Koripalli

Thesis submitted to the Faculty of the

Virginia Commonwealth University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Mathematical Sciences, Statistics Concentration

J. Paul Brooks, David J. Edwards, Advisors

August, 2012

Richmond, Virginia

Keywords: optimal design, parameter tuning, optimization software

Parameter Tuning for Optimization Software

Radha S Koripalli

(ABSTRACT)

Mixed integer programming (MIP) problems are highly parameterized, and finding parameter settings that achieve high performance for specific types of MIP instances is challenging. This paper presents a method to find the information about how CPLEX solver parameter settings perform for the different classes of mixed integer linear programs by using designed experiments and statistical models. Fitting a model through design of experiments helps in finding the optimal region across all combinations of parameter settings. The study involves recognizing the best parameter settings that results in the best performance for a specific class of instances. Choosing good setting has a large effect in minimizing the solution time and optimality gap.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Literature Review

1.1 Introduction

An *optimization problem* is the problem of finding the best solution from all feasible solutions. Each individual unique optimization problem is termed as an *instance*. An *instance* of an optimization problem is a pair (F, c), where F is any set, the domain of feasible points; c is the cost function, a mapping

$$c : F \longrightarrow R^1. \tag{1.1}$$

The problem is to find an $f \in F$ for which $c(f) \leq c(y)$ for all $y \in F$

Such a point f is called *optimal solution*.

Instances with the same constraints and objective functions are grouped into *instance classes*. Each instance is associated with a set of solutions with a certain values. Solving the optimization problem is concerned with finding the best (or optimal) solution for each given

instance, which is either the minimum (or maximum) value depending on the description of optimization problem.

Different important optimization problems can be described as mixed integer linear programming (MILP). The user must solve many instances of a certain class of MILPs, in order to identify the best solver settings which results in considerable time savings. Within MILP solvers, there are several *parameters* which control the execution of the underlying algorithm and each parameter is assigned a specific value. A set of parameter values that is assigned with one for each parameter when the solver is executed, is a *setting*. Finding the best parameter settings can result in significant time savings. This paper presents the best different (or optimal) settings for parameter values and class of instances which results in time-savings. The MILP solver considered in this paper is CPLEX (CPL, 2009). The collection of settings tested on a class of instances are termed *designed experiments*, consisting of performing a predetermined set of *runs* (or settings) where the effects of a different combination of *parameters* on a response is observed, which is considered as *result* of the experiment. For each run, the factors are set by the experimenter to a predetermined quantity called a *level*. The experiments are sometimes *replicated* for identifying the sources of variation, to better estimate the true effects of treatments.

Design optimality involves choosing designs parameter settings that produce certain *nice* characteristics in the results. *Optimal designs* allow parameters to be estimated without bias and with minimum-variance. It also produces a design which provides a good fit, sta-

bility of the prediction variance, small variances of the coefficients, detection of lack-of-fit, guards against model misspecification and minimize the impact of small errors. Once the designed experiment has been performed, the data is analysed by forming a mathematical expression, or model, of the effects of the *factors* (or independent variables) on the *response* (or dependent variables). A model is a mathematical expression that relates a function of the factors to the response. To date, no rigorous analysis of integer programming parameter settings has been conducted that accounts for the large number of categorical variables, forms a model allowing for extrapolation, and does not require an unknown number of iterations to arrive at a recommendation.

This work starts with the description of the basic principles of design of experiments (DOE) (Section 1.2) which gives the brief overview about the concept of DOE and describes the concept of optimal designs and Alphabetic optimality criteria. Section 1.3 describes a special design type, the D-optimal design and the generation of the D-optimal design, which is accomplished by the use of an exchange algorithm and is explained under algorithms for the construction of D-Optimal design. Section 1.4 describes parameter tuning methods and benchmarking metrics of , which briefly explains the methods used by different papers and about the Selection Tool for Optimization Parameters (STOP) method (Baz et al., 2007) and then discussed the set-up of our experiments (Section 2.2). Next, we report results for both the limited experiment with subset of 6 parameters and the full experiment with 59 parameters methods and compare these results to the pairwise coverage heuristic results (Section 3.1) and conclude with some general observations (Section 4).

1.2 Design of Experiments

There are two types of variables when we perform experiments: responses and factors. The response gives us information about properties and general conditions of the studied system or process. For example, the taste of a cake, fuel consumption of a car, etc. The factors or the design variables, are our tools for manipulating the system. For example, amount of flour in a cake mix recipe, exhaust gas re-circulation in a car engine, etc. Factors could be either quantitative which may change according to a continuous scale or qualitative which is a categorical variable, that can only assume certain discrete values.

Prior to conducting any experiments, the experimenter has to specify some input condi-

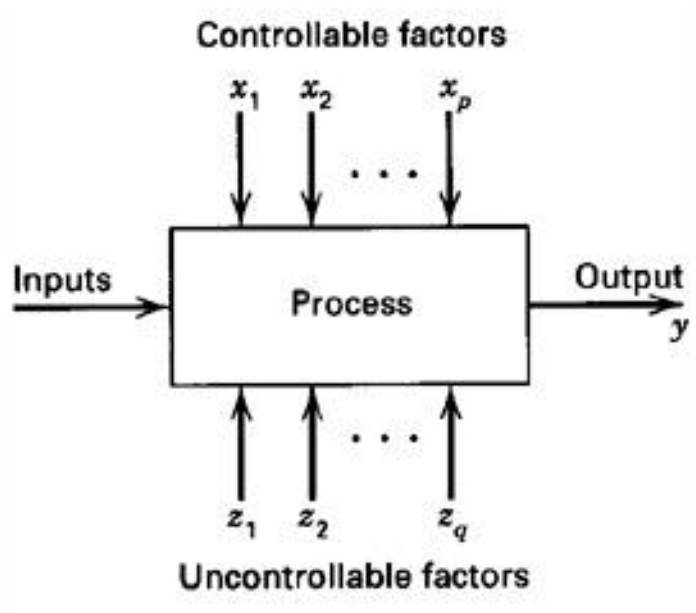


Figure 1.1: Design of Experiments System

tions: the number of factors, their range, number of responses and the experimental objective

(Eriksson, 2008). Considering the cake baking example we have our ingredients such as flour, sugar, milk, eggs, etc. Regardless of the quality of these ingredients we still want our cake to come out successfully. In every experiment there are inputs and in addition there are factors (such as time of baking, temperature, geometry of the cake pan, etc.), some of which we can control and others that we can't control. The experimenter must think about factors that affect the outcome. We also talk about the output and the yield or the response to our experiment. For the cake, the output might be measured as texture, flavour, height, size, or deliciousness, this process can be seen in Figure 1.1.

We generally want to center and scale (or code) our quantitative factors, or design variables, such that the low level is represented by -1 and a high level is represented by $+1$. This ensures they are scaled to lie in the $[-1, 1]$ interval. Specifically, if a factor, X, has maximum and minimum values $x_{i,max}$ and $x_{i,min}$ and takes values in a specified interval, as shown in equation (1.2) (Atkinson, 2007).

$$x_{i,min} \leq x_i \leq x_{i,max} (i = 1, 2, ..., k) \tag{1.2}$$

where k is the number of factors. Then for any given value x, the coded variables are defined by Montgomery (2008):

$$x'_i = \frac{(x_i - x_{i0})}{\triangle_i}, \tag{1.3}$$

where

$$x_{i0} = (x_{i,min} + x_{i,max})/2 \tag{1.4}$$

and

$$\triangle_i = x_{i,max} - x_{i0} = x_{i0} - x_{i,min}. \tag{1.5}$$

In order to understand how factors and responses relate to each other and to reveal which factors are influential for which responses, it is favourable to calculate a polynomial model of the form

$$y = f(x_1, ..., x_k) + \epsilon \tag{1.6}$$

where ϵ is a random error term, k is the number of factors and y is the response. Once a function $f()$ has been specified, the model matrix X that corresponds to the design matrix D can also be defined (Montgomery, 2008). For example, if the model proposed is a first-order polynomial with a constant (or intercept) term, then by defining β_i as the coefficient for the i_{th} variable the proposed model is

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon. \tag{1.7}$$

We can extend the equation (1.7) to one with n number of responses and arrive at:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + ... + \beta_p x_{pi} + \epsilon_i, i = 1, ..., n, \tag{1.8}$$

where y_i stand for the response of the i_{th} set of factors x_{1i},x_{2i}....x_{pi}. Equation (1.8) can also be written in matrix notation as:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{1.9}$$

where the $n \times (p+1)$ model matrix $\mathbf{X}$ contains all factors for responses, where n is the total number of runs, p is the number of factors, 1 is counted for the intercept coefficient. $\mathbf{Y}$ and $\boldsymbol{\epsilon}$ are $n \times 1$ vectors. $\boldsymbol{\beta}$ are the regression coefficients of all the factors in $\mathbf{X}$ (Wu and Hamada,

2000).

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}, \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}. \tag{1.10}$$

1.2.1 Optimal Designs

Optimal design means a design that is "best" with respect to some statistical criterion (Montgomery, 2008). In DOE for estimating statistical models, optimal designs allow parameters to be estimated without bias, with minimum-variance and least number of experimental runs to estimate the parameters (Atkinson, 2007). In practical terms, optimal experiments can reduce the costs of experimentation by decreasing the number of experimental runs.

The optimality of a design depends on the statistical model and optimal designs are selected based on maximizing (or minimizing) properties of transformations of the model matrix X (Montgomery, 2008). These designs are assessed with respect to a statistical criterion, which is related to the variance-covariance matrix $(X'X)^{-1}$ (Atkinson, 2007). Specifying an appropriate model and specifying a suitable criterion function both require an understanding of statistical theory and practical knowledge with designing experiments.

The minimizing or maximizing optimal criteria is expressed as a function of the moments matrix, $M(X) = \frac{X'X}{n}$. $\lambda_1, \lambda_2, ..., \lambda_p$ are the eigenvalues of $M(X)$ and are inversely propor-

tional to the square of the lengths of the axes of the confidence ellipsoid of the variable coefficients. That is, the smaller λ_i (i is $1, 2, ..., p$) is, the less confident we will be about the variable coefficient estimate of the linear combination $a_i'\hat{\beta}$, where a_i is the eigenvector corresponding to λ_i and $\hat{\beta}$ is the estimated coefficient(Atkinson (2007)).

There are many different optimal criteria, the mostly used criteria are mentioned below(Myers, 2009) :

Alphabetic optimality criteria

- A-optimality (average or trace)
  - A-optimality criteria, seeks to minimizes the trace of the inverse of the information matrix $(X'X)$. This criterion results in minimizing the average variance of the estimates of the regression coefficients i.e., $\min \sum_{i=1}^{p} \frac{1}{\lambda_i}$.
- D-optimality (determinant)
  - D-optimality which is one of the most popular criteria, seeks to minimize $|(X'X)^{-1}|$, or which maximizes the determinant of the information matrix $X'X$ of the design. This criterion results in minimizing the product of the generalized variances of the variable coefficient estimates; i.e., $\min \Pi_{i=1}^{p} \frac{1}{\lambda_i}$
- E-optimality (eigenvalue)
  - E-optimality criteria, maximizes the minimum eigenvalue of the information matrix. This criteria reduces the variance by minimizing the eigenvalues of $(X'X)^{-1}$; i.e., min $\max_i \frac{1}{\lambda_i}$.

Other optimality-criteria are concerned with the variance of predictions (Boyd, 2009):

- G-optimality (Global)
  - A popular criterion is G-optimality, which seeks to minimize the maximum entry in the diagonal of the hat matrix $X(X'X)^{-1}X'$. This has the effect of minimizing the maximum variance of the predicted values.
- I-optimality (integrated)
  - A second criterion on prediction variance is I-optimality, which seeks to minimize the average integrated prediction variance over the design space.
- V-optimality (variance)
  - A third criterion on prediction variance is V-optimality, which seeks to minimizes the average prediction variance over a set of m specific points

Since the optimality criterion of most optimal designs is based on some function of the information matrix, the "optimality" of a given design is model dependent, this means there can be first-order, second-order, etc. optimal designs. For example, an optimal design that used a D-optimal criterion and the first order model would be termed a first-order D-optimal. While an optimal design is best for a particular model, its performance may deteriorate on other models. On other models, an optimal design can be either better or worse than a non-optimal design. Therefore, it is important to benchmark the performance of designs under alternative models.

1.3 D-Optimal Designs

Out of different design optimality criterion, the most widely used is the D-optimality criterion. A design is said to be D-Optimal if $|(X'X)^{-1}|$ is minimized. A D-optimal design minimizes the volume of the joint confidence region on the vector of regression coefficients. A measure of the relative efficiency of design 1 to design 2 according to the D-criterion is given by:

$$D_e = \left(\frac{|(X_2'X_2)^{-1}|}{|(X_1'X_1)^{-1}|} \right)^{\frac{1}{p}} \tag{1.11}$$

where X_1 and X_2 are the X model matrices for the two designs and p is the number of model parameters. Many popular software packages, including JMP, Design-Expert, and minitab will construct D-Optimal designs.

1.3.1 Algorithms for constructing D-Optimal Designs

The usual way to construct optimal designs is to specify a model, determine the region of interest, select the number of runs to make, specify the optimality criterion and then choose the design points from a set of *candidate* points that the experimenter would consider using. Usually the candidate points are a grid of points spaced over the feasible design region.
The alphabetically optimal design can be constructed analytically. One good example is the 2^k design, which is D-, A-, G- and V-optimal for fitting a first order model in k variables or for fitting a first order model with interaction. However, in most cases the optimal design is not known, and a computer-based algorithm must be used to find the design.

Exchange Algorithms

Based on the complexity of the D-optimal designs and the huge number of possible combinations of experiments, computer algorithms are used for the selection process.

An exchange algorithm selects the optimal design matrix X^* by exchanging one or more points from a generated start design and repeats this exchanges until the best matrix seems to be found. The algorithms can be differentiated in two groups where a rank-1 algorithm adds and deletes the points sequentially, and a rank-2 algorithm realizes the exchange by a simultaneous adding and deleting process (Meyer and Nachtsheim, 1995).

Point Exchange algorithm is one of the design construction methods. The form of this algorithm is, a grid of candidate points where an initial design from these points is selected. Then the algorithm exchanges points that are in the grid but not in the design with points currently in the design in an effort to improve the selected optimality criterion. As not every possible design is evaluated, there is no guarantee that an optimal design has been found, but the exchange procedure ensures that a design that is "close" to optimal results. Some implementations repeat the design construction process several times, starting from different initial designs, to increase the likelihood that a final design that is very near the optimal will result (Montgomery, 2008).

Another way to construct optimal designs is with a *coordinate exchange* algorithm (Montgomery, 2008). This method searches over each coordinate of every point in the initial design recursively until no improvement in the optimality criterion is found. The procedure is usu-

ally repeated several times with each cycle starting with a randomly generated initial design. See Meyer and Nachtsheim (1995) and Fabian (2008) for further details and discussion of the Exchange algorithm.

1.4 Parameter Tuning and Benchmarking for Optimization Software

The performance of optimization software improves if a few critical parameter values are good. Which parameters are critical is decided by the class of instances. There are some papers which show the effort made to find the best parameter settings for a variety of algorithms.

Kohavi et al. (1995) describes the "Wrapper" method by considering the evaluation of the best parameters as a discrete function optimization problem. This method uses best-first search, which treats the parameter space as a set of connected nodes, and pursues the unvisited successor node of visited nodes that has the best score and cross-validation to wrap around the basic induction algorithm (Mitchell, 1982). The Wrapper method requires two components, a search component which repeatedly suggests parameter settings and the evaluation component which evaluates these setting by running the induction algorithm several times and getting an estimate of resulting accuracy. Also, this method is limited to numeric or binary parameters.

Imbault and Lebart (2004) consider the parameter tuning for support vector machines (SVMs),the results of which highly depends on the two parameters of the model: the scale of the kernel and the regularization parameter. Estimating these parameters is referred to as *tuning*.

Benchmark results from Imbault and Lebart (2004) suggests an improvement in the classification efficiency when compared to using no tuning at all and also this method is fairly fast for even very large datasets. However, it has been observed that the initialisation of the algorithm is crucial, i.e., bad initialization could lead to bad results.

The two techniques suggested by Imbault and Lebart (2004) to tune these parameters are genetic algorithms and simulated annealing. These methods require a large number of successive runs to be performed, each dependent on the results of previous runs and also, the second technique, simulated annealing is restricted to continuous parameters. Both the methods give similar results. Genetic algorithms tend to be faster. Simulated annealing requires less parameter settings when compared to genetic algorithms. The main drawback of these methods is their computational load.

Adenzo-Diaz (2004) developed the CALIBRA method for fine-tuning algorithms. This method uses experimental design (Taguchi's fractional factorial design) along with a local search procedure to find the best values of the search parameters, but these values are

not guaranteed to be optimal. CALIBRA can also be used for searching the best possible solution to a given problem instance, i.e., finding values for a set of parameters which provides the best possible answer to the problem of interest. This procedure consists of four phases:

1. Perform a 2^k full factorial experiment, with each parameter restricted to two extreme values, using the results to identify the most important parameter.

2. Determine the initial solution for the local search. The local search uses the $L_9(3^4)$ design, which can handle a maximum of 4 parameters, if the parameters are more than four, the remaining parameters other than 4 are fixed to a chosen value.

3. Perform local search to find the optimal values of the parameters by incrementally reducing the range of parameters until the range converges to a single point.

4.Check if more experiments can be performed within the allotted budget. If a new search is possible then it again starts the new search from step 2.

Audet et al. (2006) consider using mesh adaptive direct search for searching through the potential parameters, which iteratively searches over a decreasing area of the parameter space. It depends on the distance between parameters value, which does not apply to categorical

factors. This iterative search requires an unknown number of steps to find a minimum. Audet et al. (2006) used Black-Box approach for parameter estimation, which selects value for the parameter which seems to perform better which can be specified by the user. In a way it is beneficial as it allows user to use their full knowledge of the context to design appropriate test sets and performance measures.

Hutter et al. (2009) consider different procedures based on Gaussian process models for optimizing the performance of parametrised, randomised algorithms. Hutter et al. (2009) referred to two approaches, sequential parameter optimisation (SPO) (Bartz-Beielstein, 2006) and sequential kriging optimisation (SKO) (Huang, 2006); both approach were evaluated. A Latin hyper-cube (LHD) design for the parameters is generated for both the models, and a Gaussian process model is fit to the results. A new set of runs is then selected based on the expected improvement, and the process repeats until a specified number of iterations have completed, or negligible improvement is found in the Gaussian process model. This process requires a large number of runs to make a recommendation, and finally, the performance of SPO resulted in being more robust than SKO. The models used may not result in good extrapolation properties, as Gaussian process models does not attempt to model points which are not tested, and so the process may or may not estimate performance well at points which are not explicitly tested.

Benchmarking is an important tool in the evaluation and development of solvers for mathematical programming problems, in uncovering solver deficiencies, and in the quality assurance

process of mathematical programming software.

Hans D. Mittelmann (2006) developed an online server PAVER to help facilitate and automate performance analysis and visualization of benchmarking data taking into account various performance metrics like robustness, efficiency and quality of solution. PAVER provides simple online tools for automated performance analysis, visualization, and processing of benchmarking data. Benchmark data obtained by running several solvers over a set of models can be automatically analyzed via online submission to the PAVER server. A detailed performance analysis report is returned via e-mail in HTML format and is also initially available online.

PAVER accepts benchmarking data in the form of GAMS trace files or generic data files, where each data file contains data for a single solver over a model test set. Users submit their data files via the online (worldwide web) submission tool at:
$http://www.gamsworld.org/performance/paver/pprocess\_submit.htm.$

The tool accepts the submission of up to eight data files, automatically performing cross comparisons of each solver with respect to every other solver. After submission, the data files are analyzed for correctness and formatting. If a data file does not have the proper format, users receive an error message online. Provided the data files are valid, a scheduler is invoked, which schedules the performance analysis job for a particular workstation. The scheduling task takes the form of optimizing a mixed integer program, taking into account current workstation workloads. The data files are then sent from the server to the next available workstation specified by the schedule. The performance analysis is completed and

the results returned to the server. The server then sends the results via e-mail attachment to the user.

The tools available in PAVER allow either direct comparisons between two solvers or comparisons of more than two solvers simultaneously. The solver square and resource time utilities belong to the former and the performance profile utility to the latter. Because solvers use different optimality metrics (measurements of optimality) and feasibility, the comparison of benchmarking data from different solvers is inherently imprecise. It may be useful to verify the solution provided by each solver using the same optimality and feasibility criteria. This additional measure still does not guarantee a completely fair comparison.

1.4.1 MILP Parameter Tuning

Linear programming (LP or Linear optimization) is a mathematical method for getting the best results in a given mathematical model for some different set of requirements represented as linear relationship. If some of the unknown parameters are required to be integers, then the problem is called Mixed Integer Linear Programming problem. A common approach to modeling optimization problems with discrete decisions is to formulate them as mixed integer optimization problems. Such a problem is called a mixed integer linear optimization problem (MILP) (Nemhauser and Wolsey, 1988). Tuning different parameters concerned in solving MILP problems results in considerable reductions in solve time.

STOP Methods

Baz et al. (2007) implement a method Selection Tool for Optimization Parameters (STOP) for tuning software parameters using ideas from software testing and machine learning, which reports the best settings of a parameter by trying a number of settings on representative instances. The tests of methods are presented on three MILP solvers: CPLEX, CBC, and GLPK. Out of these tests, the STOP method finds parameter values that outperform the default values. Baz et al. (2007), also exhibit that significant reductions in solution times can be made by considering the combination of heuristic and machine learning techniques. With STOP 31% to 88% of improvement in total run time over default performance was generally observed. This method attempts to find good parameter values using a relatively small number of optimization trials, but as it tries many settings of a parameter this method requires a significant investment of computer time, though it requires very little of the users time. The general steps (Baz et al. ,2007) for STOP method are:

1. Select the initial settings of set of parameters using random, greedy heuristic or pairwise coverage.
2. Record different metrics by running the solver on the initial settings selected above.
3. Use machine learning(either regression trees and artificial neural networks) to identify additional settings based on the initial data obtained from the above two steps.
4. Record the results by running the solver on the additional settings obtained from step 3.
5. Output the best observed settings.

There are three methods used in step 1 (Baz et al., 2007) for selecting the initial settings: random, greedy heuristic and pairwise coverage. The random selection of initial settings is done by selecting the each value of a parameter randomly. This method is easy to execute but might miss some interactions.

For the greedy heuristic method (Baz et al., 2007), initial settings are selected randomly and then for each additional settings, all the possible combinations of settings are compared with the already selected initial settings based on minimizing the number of parameters in common. Then this method iteratively selects settings. Then again new settings are chosen by minimizing the maximum number of parameter values in common with any individual previously selected setting. Ties are broken based on the setting with the minimum sum of parameter values in common with all previously selected settings. This method could become complex for the large number of parameters as it requires extensive search of possible settings.

The pairwise coverage method used an algorithm from Cohen et al. (1997), which can generate a pairwise coverage test set where all pairs of parameter settings appear at least twice. This test is used in identifying programming errors that are revealed when two parameter values interact.

Step 3, i.e., machine learning, is an optional part of the algorithm. This method is used

to get the useful information from the initial settings to direct the selection of additional settings by constructing good probabilistic models. Baz et al. (2007) suggests using either of the two machine learning methods: regression trees or artificial neural networks. Regression trees are used to find the two settings that had the largest effect in minimizing the solution time. They are used as a non-parametric method regression. In linear regression, Regression trees are used to identify the relationship between a continuous dependent variable and independent variables. (See Breiman et al. (1984) for more details on Regression trees). Neural networks are used to predict the response at every single point in the parameter space to identify further points to test. This method is not suitable for extremely large parameter space.

The STOP software in Baz et al. (2007) used only one metric TIME-TO-OPTIMALITY, to optimize the parameter settings. TIME-TO-OPTIMALITY is the total time required to solve an MILP instance optimally within the tolerance of the solver. Baz et al. (2009) extended the work of Baz et al. (2007) by considering two additional metrics, PROVEN-GAP at time t and BEST-INTEGER-SOLUTION at time t. These two metrics are useful for the users who may not need the optimal solution, which can reduce extensive use of computer effort and may need a good quality solution within a reasonable period t. When settings obtained from STOP with the TIME-TO-OPTIMALITY metric were used, the average improvement is larger than 55.01% for MIPLIB instances and with PROVEN-GAP metric within 600 and 3600 seconds, average improvement is 58.35% and 65.53% respectively. With the BEST-INTEGER-SOLUTION metric within 600 and 3600 seconds, it is up to a 35.33%

and 38.24% improvement respectively. For more details on the computation part, see Baz et al. (2009).

Chapter 2

Methodology

2.1 Introduction

Mixed integer programming (MIP) solvers have a large set of parameters which give users control over a wide range of design choices. The MILP solver considered in this paper is CPLEX (CPL, 2009)–a widely used commercial MIP solver. We use design of experiments and statistical modelling to find the settings of the CPLEX solver parameters. Model fitting can also explain the effects of the parameters through which we can identify the desirable and bad settings. The two metrics considered in this paper are solution time and optimality gap. Solution time is the CPU time required to achieve provable optimality and it is measured in centi-seconds (cs). Gap is often given as a indicator of a solution to an MIP problem, and for minimization problems gap is calculated as

$$G = \frac{U - L}{U}, \tag{2.1}$$

where G is the optimality gap, for minimization, U represents the best-known upper bound on the optimal objective value and L represents the best-known lower bound on the optimal objective value. Gap is measured in percentage (%), so it is always positive. Better solution is found, when the gap is smaller.

The three instance classes considered are telecommunication network design problems (easy) with 20 instances with solution time as the response from Yasser Tanvir (2009), cellular metabolism instance class (medium) with 5 instances with solution time or optimality gap as the response from Brooks et al. (2012) and MIPLIB subset problem (hard) with 3 instances with optimality gap as response from Thorsten Koch et al. (2011). These instances were used in our research on the basis of the perl script used by Charles(2010).

2.2 Method

Using the CPLEX 12.0 "parameters reference manual", we identified 59 parameters (shown in Appendix A) that can be modified in order to improve performance. The 59 parameters we selected affect all aspects of CPLEX. They include 17 preprocessing presolve parameters (mostly categorical); 15 MIP best bound strategy parameters (mostly categorical); 16 categorical parameters deciding how aggressively to use which types of cuts, cuts are constraints added to a model to restrict (cut away) non-integer solutions that would otherwise be solutions of the continuous relaxation. The addition of cuts usually reduces the number of branches needed to solve a MIP, and 11 simplex parameters. Most parameters have an

"automatic" option as one of their values, which is a default value. We allowed this value, but also included other values (for categorical parameters).

After the solver, instance class and the set of parameters of interest has been chosen, two cases are considered:

1) a limited subset of 6 CPLEX parameters used by Baz et al. (2007), which can be seen in Table 2.1 and

2) the full set of 59 parameters available to CPLEX (shown in Appendix A).

Table 2.1: Six CPLEX parameters used for the limited case

Parameter	Levels	Description
MIP emphasis	[0,1,2,3,4]	Controls trade-offs between speed, feasibility, optimality and moving bounds in MIP
Node selection	[0,1,2,3]	Sets the rule for selecting the next node to process when backtracking
Branching var. sel.	[-1,0,1,2,3,4]	Sets the rule for selecting the branching variable at the node which has been selected for branching
Dive type	[0,1,2,3]	Controls the MIP dive strategy
Fractional cuts	[-1,0,1,2]	Decides whether or not Gomory fractional cuts should be generated for the problem
MIR cuts	[-1,0,1,2]	Decides whether or not to generate MIR cuts for the problem

The solution time and optimality gap is obtained by running the design runs for different

instances through jobs submission on a Beowulf Linux cluster, which uses Sun Grid Engine to submit the jobs with identical nodes,i.e., four 2.6GHz processors with 4GB RAM. The Sun Grid Engine (SGE) is a queue and scheduler that accepts jobs and runs them on the cluster for the user. SGE has a large set of programs that let the user submit/delete jobs, check job status, and have information about available queues and environments. SGE has the following advantages:

• Scheduling - allows us to schedule a virtually unlimited amount of work to be performed when resources become available. This means we can simply submit as many tasks (or jobs) as we like and let the queuing system handle executing them all.

• Load Balancing - automatically distributes tasks across the cluster such that any one node doesn't get overloaded compared to the rest.

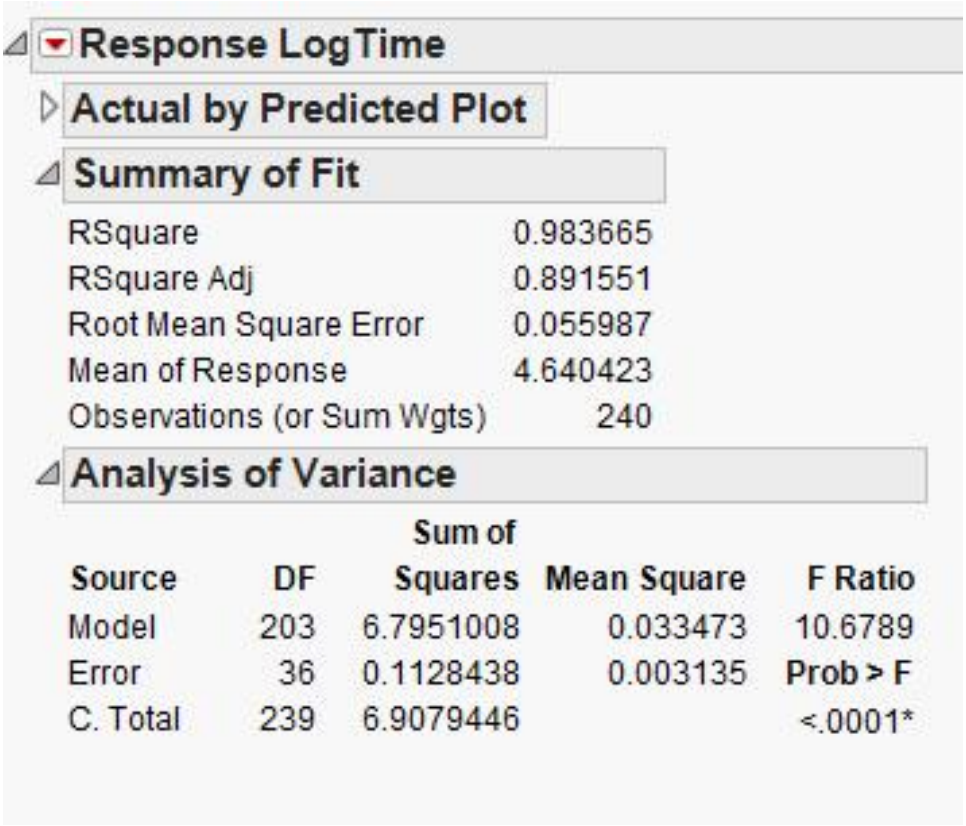
• Monitoring/Accounting - ability to monitor all submitted jobs and query which cluster nodes they are running on, whether they are finished, encountered an error, etc. Also allows querying job history to see which tasks were executed on a given date, by a given user, etc.

2.2.1 Building and Evaluating Model

To build and evaluate the models a statistical software package called JMP is used. JMP includes an array of statistical platforms that helps in building robust models for our data. With methods for revealing relationships among variables in a process, JMP allows us to not only make predictions but also to identify settings for factors that yield better performance. The steps below explains how the settings are selected through profiler:

1. For building a model in JMP, click *Analyze* in the task bar and then select *Fit Model* in the drop down menu and then add response and factors to the corresponding fields and click *Run*, this will generate the report as shown in Figure 2.1.

2. Select Prediction Profiler from the red triangle menu on the report title bar (Figure 2.1).



Response LogTime

Actual by Predicted Plot

Summary of Fit

RSquare	0.983665
RSquare Adj	0.891551
Root Mean Square Error	0.055987
Mean of Response	4.640423
Observations (or Sum Wgts)	240

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	203	6.7951008	0.033473	10.6789
Error	36	0.1128438	0.003135	Prob > F
C. Total	239	6.9079446		<.0001*

Figure 2.1: JMP Result Report

Scroll to the bottom of the report to open the Prediction Profiler. Open it by clicking the disclosure button beside the Prediction Profiler title bar. The Profiler (Figure 2.2) displays prediction traces for each parameter.

3. Click the red triangle icon in the Prediction Profiler title bar and select *Desirability Functions*, which is to select the desirable response, i.e., if we want to minimize the response, we

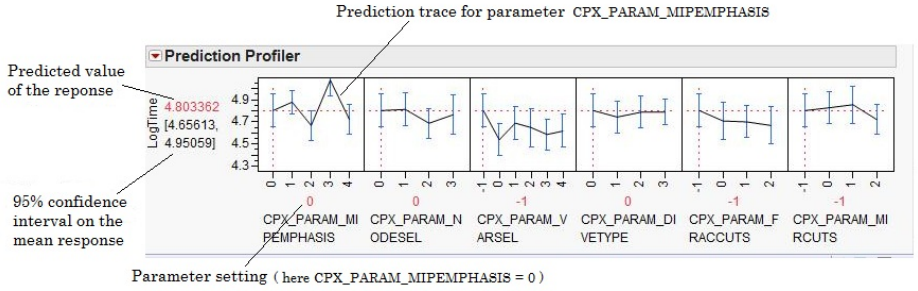


Figure 2.2: The Prediction Profiler

can select the desirability as minimum, otherwise by default it is set at maximum, which is to maximize the response. Now select *set desirability* from the same drop down list and Select the response goal as *minimize desirability*, as we seek our response solution time or optimality gap to be minimum.

4. Click the red triangle icon in the Prediction Profiler title bar and select *Maximize Desirability*. JMP automatically adjusts the graph to display the optimal settings (Figure 2.3).

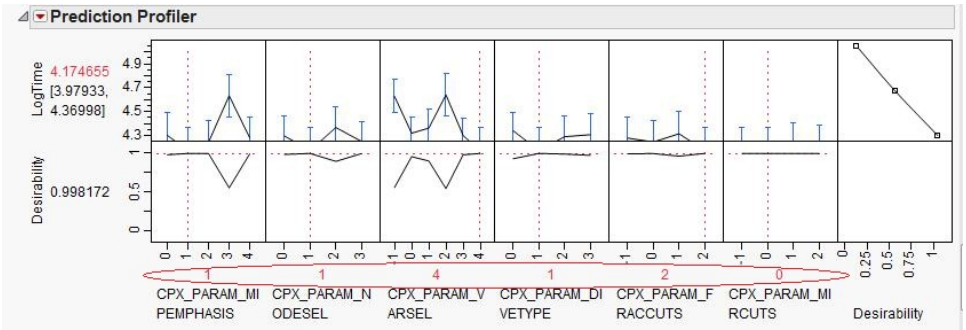


Figure 2.3: The most desirable settings

The number of runs for constructing a first order or second-order model is suggested by JMP - Design Generation panel. The Design Generation panel (Figure 2.4) shows the minimum

number of runs needed to perform the experiment based on the effects we have added to the model.

The Design Generation panel has the following options for selecting the number of runs we want (Figure 2.4) (JMP,2005):

- *Minimum* is the smallest number of terms that can create a design. When you use Mini-

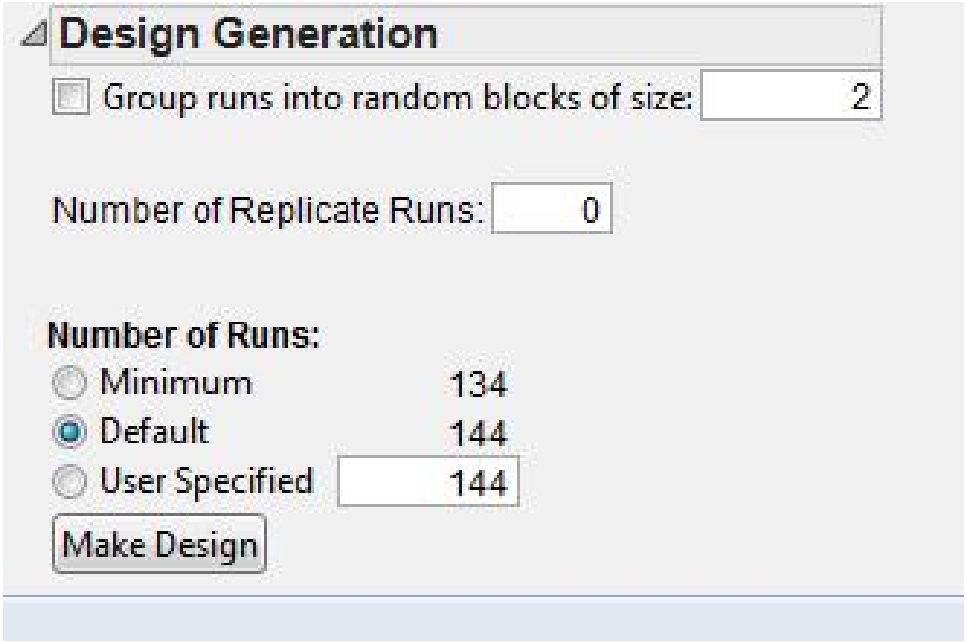


Figure 2.4: Options for Selecting the Number of Runs

mum, the resulting design is saturated (no degrees of freedom for error). This is an extreme choice that can be risky, and is appropriate only when the cost of extra runs is prohibitive.

- *Default* is a custom design suggestion for the number of runs. This value is based on heuristics for creating balanced designs with a few additional runs above the minimum.

- *User Specified* is a value that specifies the number of runs we want.

2.2.2 Method for Limited Subset CPLEX Parameters Experiment

A total of 7680 settings of full-factorial design, with all the 6 CPLEX important parameters used by Baz et al. (2007) are submitted as jobs using SGE to obtain the solution time and optimality gap, these settings are independent of each other. Then for each combination of settings of 7680 runs, the median of the solution time and average of optimality gap is calculated. Median is considered for the solution time as it is a good measure when some of the values for solution time are missing for the settings, as for the missing settings average is not a good measure and average is considered for the optimality gap, as the gap for the settings which achieves provable optimality for some settings is zero and when we log transform, zero is invalid and also average tend to stabilize the variance. These results for solution time and optimality gap are then used to find the possible outcomes for the optimal designs.

The below process is followed to identify best settings:

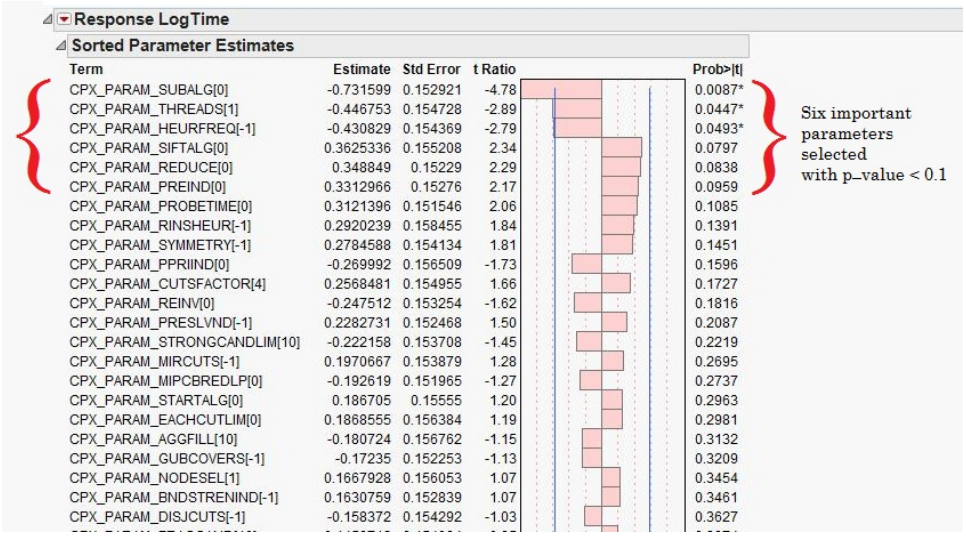
Optimal Design: Construct a second-order D-optimal and I-Optimal designs to fit a second-order regression model with the 6 CPLEX important parameters using JMP, and identify which settings give the best performance and get the actual values for the solution time and optimality gap for these settings from the full factorial results.

2.2.3 Method for Full Set CPLEX Parameters Experiment

Below process is followed to identify important parameters and best settings:

1. *Screening*: Construct a first-order D-optimal design with the two extreme levels of all the

59 CPLEX parameters using JMP and identify the most influential parameters, by selecting the parameters with p-value less than 0.1 from the JMP results report. For example, Figure 2.5 shows the screening design results report with sorted parameter estimates and the p-value for the easy class of instance. The first six parameters shows the p-value as less than 0.1, these first six parameters were selected as the important parameters for the follow-up design for easy instance class (section 3.1.2), as shown in the Figure 3.5.



Term	Estimate	Std Error	t Ratio	Prob>|t|
CPX_PARAM_SUBALG[0]	-0.731599	0.152921	-4.78	0.0087*
CPX_PARAM_THREADS[1]	-0.446753	0.154728	-2.89	0.0447*
CPX_PARAM_HEURFREQ[-1]	-0.430829	0.154369	-2.79	0.0493*
CPX_PARAM_SIFTALG[0]	0.3625336	0.155208	2.34	0.0797
CPX_PARAM_REDUCE[0]	0.348849	0.15229	2.29	0.0838
CPX_PARAM_PREIND[0]	0.3312966	0.15276	2.17	0.0959
CPX_PARAM_PROBETIME[0]	0.3121396	0.151546	2.06	0.1085
CPX_PARAM_RINSHEUR[-1]	0.2920239	0.158455	1.84	0.1391
CPX_PARAM_SYMMETRY[-1]	0.2784588	0.154134	1.81	0.1451
CPX_PARAM_PPRIIND[0]	-0.269992	0.156509	-1.73	0.1596
CPX_PARAM_CUTSFACTOR[4]	0.2568481	0.154955	1.66	0.1727
CPX_PARAM_REINV[0]	-0.247512	0.153254	-1.62	0.1816
CPX_PARAM_PRESLVND[-1]	0.2282731	0.152468	1.50	0.2087
CPX_PARAM_STRONGCANDLIM[10]	-0.222158	0.153708	-1.45	0.2219
CPX_PARAM_MIRCUTS[-1]	0.1970667	0.153879	1.28	0.2695
CPX_PARAM_MIPCBREDLP[0]	-0.192619	0.151965	-1.27	0.2737
CPX_PARAM_STARTALG[0]	0.186705	0.15555	1.20	0.2963
CPX_PARAM_EACHCUTLIM[0]	0.1868555	0.156384	1.19	0.2981
CPX_PARAM_AGGFILL[10]	-0.180724	0.156762	-1.15	0.3132
CPX_PARAM_GUBCOVERS[-1]	-0.17235	0.152253	-1.13	0.3209
CPX_PARAM_NODESEL[1]	0.1667928	0.156053	1.07	0.3454
CPX_PARAM_BNDSTRENIND[-1]	0.1630759	0.152839	1.07	0.3461
CPX_PARAM_DISJCUTS[-1]	-0.158372	0.154292	-1.03	0.3627

Figure 2.5: Selection of important parameters from screening design result report for easy class of instance

2. *Follow-up design*: Construct a second-order D-optimal design in JMP to fit a second-order regression model with the best influential parameters selected in the first step and identify which settings give the best performance and then find the actual values for the solution time and optimality gap for these settings by running the job using SGE.

The models fitted in the both the limited and full set experiments discussed above, are the

linear regression models. The response used in the models is the log of the median over the solution time of each instance of an instance class i.e., for easy median over the 20 instances of each run, for medium median over 5 instances and for hard median of solution time over 3 instances and average gap. Both the response are log transformed, in order to reduce the effects of increased variance in solution time and optimality gap as they both increases. For easy class of instance, solution time is the only response considered; for medium solution time and optimality gap are considered as two responses treated separately and for hard optimality gap is considered as the only response.

In the follow-up step of full set experiment, the second-order model consists of all the main effects and the second order interactions. For example, if we consider a second-order model with two parameters x_1 and x_2, the model looks like the equation (2.2).

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{1i} x_{2i} + \epsilon_i, i = 1, 2, ..., n. \tag{2.2}$$

where y is the response variable, β_0 is the intercept, β_1,β_2 are the coefficient of estimates for the two factors x_1 and x_2, β_3 is the coefficient of estimate for the interaction term $x_1 x_2$, which explains the interaction between the two factors x_1 and x_2 and if this term is significant, then the effects of one factor depends on the level of the other.

2.2.4 Method for Pairwise coverage Heuristic

The best settings recommended in the limited and full set experiments are compared to the pairwise heuristic method. pairwise heuristic method attempts a number of settings on each

class of instance and reports the best setting observed.

When pairwise heuristic is compared to limited set experiment, the settings for pairwise heuristic is generated by pairwise coverage algorithm with the six important parameters. In Pairwise coverage method, an algorithm from Cohen et al. (1997) is used to generate the pairwise coverage test sets. The objective of this algorithm is to have all pairs of parameter values appear at least twice in settings. If the number of settings needed is higher than the number the user allows, then we step down to the easier constraint that all pairs appear at least once.

To get the actual solution time and the optimality gap, the settings obtained from the pairwise heuristic are then extracted from the 7,680 settings from the full-factorial design discussed in section 2.2.2 for easy, medium and hard class of instances. Then the best settings for the pairwise heuristic can be obtained by getting the lowest value for the solution time from all set of pairwise settings for easy and medium instance classes and lowest value for optimality gap from all set of pairwise settings for medium and hard instance classes.

When pairwise heuristic is compared to full set experiment, the settings for pairwise heuristic is generated by pairwise coverage algorithm with the all the 59 parameters. In this case, for additional settings greedy heuristic can't be used to reach the specified number of initial settings, because this method is intractable for the large set of parameters. Greedy heuristic method selects the first settings randomly and then for each additional setting, all possible settings are compared with the already selected settings. For each possible setting, the number of parameter values in common with each of the already-selected settings

are considered. Then the new setting that minimizes the maximum number in common are chosen. This complete process is very difficult for large set of parameters, due to which only pairwise coverage algorithm is used to generate all the settings in the full set experiment.

Chapter 3

Results

3.1 CPLEX Results

The results for both limited and full set experiments are shown in this section and these results are compared to the pairwise selection heuristic method (Cohen et al., 1997), as it also attempts to find better parameter settings for MIP solvers, and does not require an open-ended number of iterations.

Optimal design can be a better choice for a user than the pairwise coverage, as with the pairwise coverage one may not be able to fit a statistical model. In the case of optimal design, the best runs from the design region are generated with a user specified model and an optimality-criterion i.e., we can enforce constraints. Optimal designs such as a D-optimal design minimize the volume of the joint confidence region of the regression coefficients. Optimal design often requires fewer runs than a traditional design.

3.1.1 Limited Set Experiment

The process from method for the limited set experiment from section 2.2.2 are performed. Considering the six important categorical CPLEX parameters- 4 parameters with 4 values each and 2 parameters with 5 and 6 values each, results in a total of 7,680 potential combinations of settings. All the 7,680 settings are then run on three instance classes easy, medium and hard with 20, 5 and 3 instances respectively.

For constructing a D-optimal design with the six critical parameters mentioned in Table 2.1, a minimum of 204 runs are needed to fit a second order model with the main effects and two factor interactions, we selected 240 runs, which is greater than the minimum runs required, suggested by JMP -Design Generation panel as the default runs. With the 240 runs a second-order regression model is fit to the log of the median solution times across the 20 and 5 instances of easy and medium class of instance respectively and also a second-order regression model is fit to the log of the average gap across the 5 and 3 instances of medium and hard class of instance respectively. These models includes all the parameter levels for each of the six CPLEX parameters considered, and with all main effects and two factor interactions.

Then the second-order model is used to identify the setting that would give the faster predicted solution time for easy and medium and smaller optimality gap for medium and hard class of instances though profiler and the recommended settings are shown in Table 3.1 below. Also, settings with the lowest response from the design are shown in Table 3.2. Then the CPLEX limited experiment recommendations are compared with pairwise method. In

Table 3.1: Recommended settings and values by the model in CPLEX limited set experiment

Instance(Value Type)	MIPEmphasis	NodeSels	VarSels	DIVtypes	FracCuts	MIRCuts	Value
Easy(time)	1	1	4	1	2	0	0.765s
Medium(time)	2	2	-1	0	1	-1	20.99s
Medium(gap)	2	1	0	0	2	2	7.73%
Hard(gap)	4	1	4	1	1	1	47.116%

Table 3.2: Settings for the lowest response from design and response values in CPLEX limited set experiment

Instance(Value Type)	MIPEmphasis	NodeSels	VarSels	DIVtypes	FracCuts	MIRCuts	Value
Easy(time)	1	1	3	1	-1	0	0.745s
Medium(time)	2	3	4	1	1	0	18.61s
Medium(gap)	2	1	0	0	2	1	7.07%
Hard(gap)	4	2	3	2	0	2	44.207%

pairwise method, 60 runs are selected using pairwise heuristic algorithm (Cohen et al., 1997), and the fastest run in each of the sets was identified for easy and medium class of instance and smallest average gap was selected for medium and hard class of instance. The results are shown in Table 3.3 below.

Table 3.3: Recommended setting and values in pairwise coverage heuristic method

Instance (Value Type)	MIPEmphasis	NodeSels	VarSels	DIVtypes	FracCuts	MIRCuts	Value
Easy(time)	1	0	3	0	-1	1	0.75s
Medium(time)	2	0	0	3	1	1	18.22s
Medium(gap)	2	1	4	3	-1	2	8.5807%
Hard(gap)	4	1	2	3	2	0	44.1409%

The solution time and optimality gap for the default settings obtained for the six CPLEX

parameters can be seen in Table 3.4

Table 3.4: Default setting and values

Instance (Value Type)	MIPEmphasis	NodeSels	VarSels	DIVtypes	FracCuts	MIRCuts	Value
Easy(time)	0	1	0	0	0	0	0.99s
Medium(time)	0	1	0	0	0	0	1216.60s
Medium(gap)	0	1	0	0	0	0	19.8855%
Hard(gap)	0	1	0	0	0	0	82.2146%

Comparison

By comparing the results for CPLEX limited, pairwise coverage and default settings for the six parameters, the results are shown in the plots, easy in Figure 3.1, medium with solution time as the response in Figure 3.2, medium with optimality gap as the response in Figure 3.3 and hard with optimality gap as the response in Figure 3.4 instances. The x-axis in the plots indicate the ordered runs and y-axis indicates the median solution time or average optimality gap.

By looking at all the plot results, it is clear that recommendations from the fitted model and pairwise coverage heuristic does much better than the defaults. Also, there is not much difference between the recommendations of the pairwise coverage heuristic and the use of optimal models.

In case of easy class of instances, from Figure 3.1, pairwise coverage method does slightly better than the CPLEX model recommended settings, but the lowest setting from the design runs does better than pairwise method; these settings are obtained by selecting the lowest

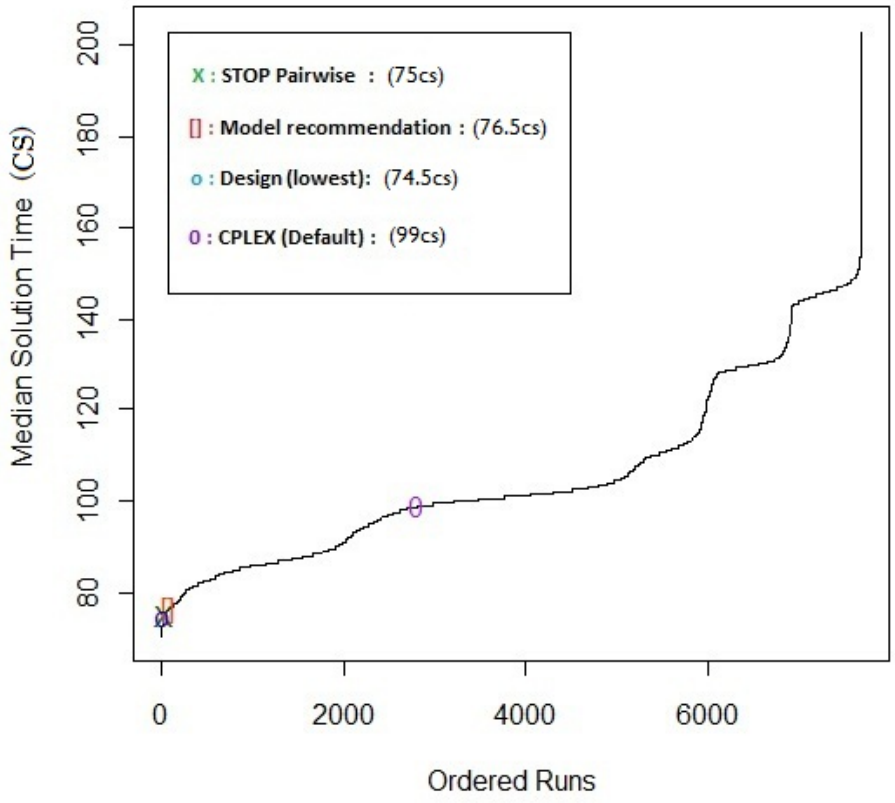


Figure 3.1: Median solution time of limited set CPLEX parameters experiment for easy instance

value for the response (i.e., minimum solution time) from the design runs.

For medium class of instance, with solution time as the response (Figure 3.2), pairwise method does better than both the model recommended and the settings selected from the design runs. Medium class, with optimality gap as the response, from Figure 3.3, model and the lowest settings from design does better than the pairwise method.

In case of hard class of instance, with optimality gap as the response (Figure 3.4), pairwise method recommendations is better than the model recommended settings and pairwise is

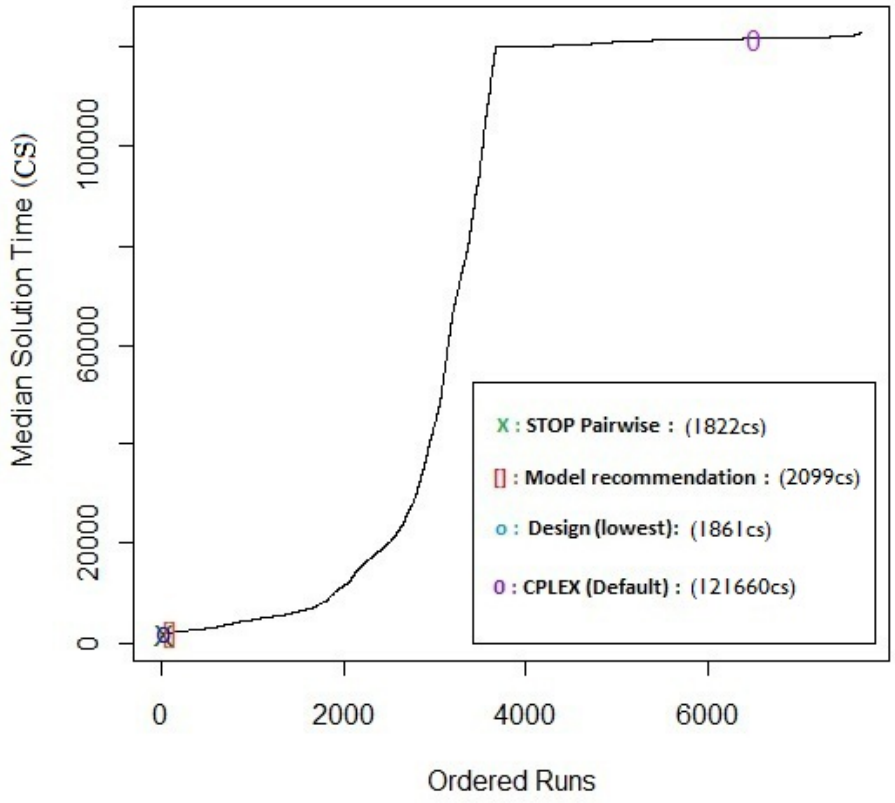


Figure 3.2: Median Solution time of limited set CPLEX parameters experiment for medium instance

comparable with settings from the design, both settings produce almost the same desirable response.

CPLEX model also has the added benefit of providing information about the effects of the parameters on solution time or gap across the instance class, for example, Figure 3.5 for easy class of instances is shown, which indicates the best settings as (1 1 4 1 2 0) for the corresponding six parameters, as this setting generates the best solution time, these setting are

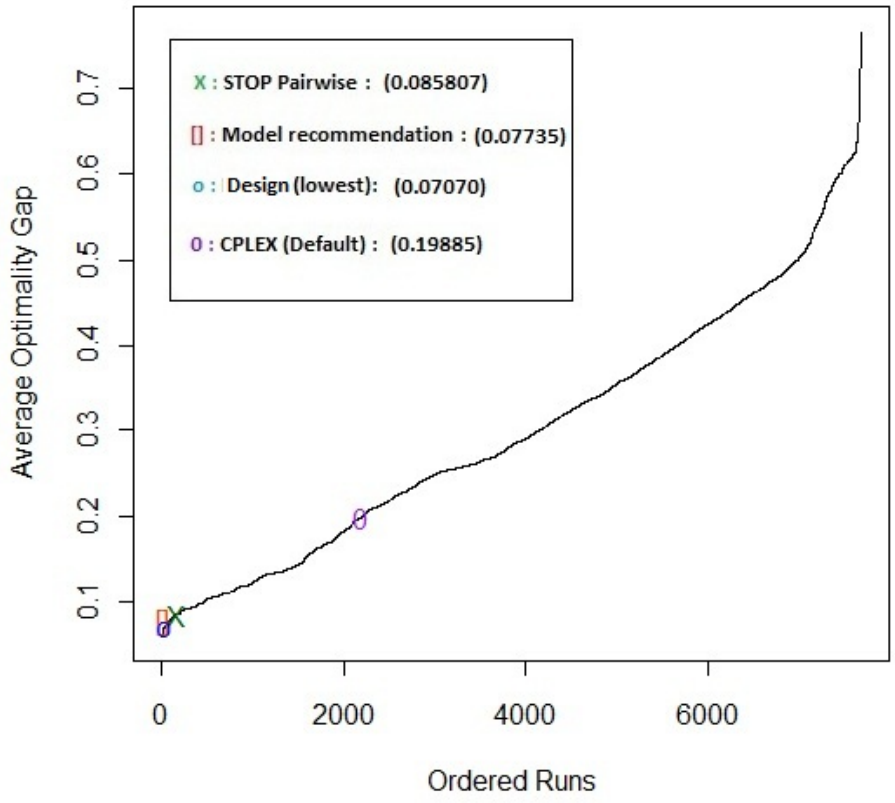


Figure 3.3: Average optimality gap of limited set CPLEX parameters experiment for medium instance

obtained by following the steps mentioned in section 2.2.1. The effect of the parameters can also be explained from the profiler in Figure 3.5; MIP emphasis switch parameter provides the best solution time at level 1 and when it is set at level 3, solution time is increasing, indicates that level 3 is not the good setting for the parameter MIP emphasis switch. Similarly, the parameter MIP variable selection strategy provides the best solution time when set at level 4 and solution is large at level 2, which indicates level 2 is not a good setting for the parameter MIP variable selection strategy. The effects of the remaining parameters can

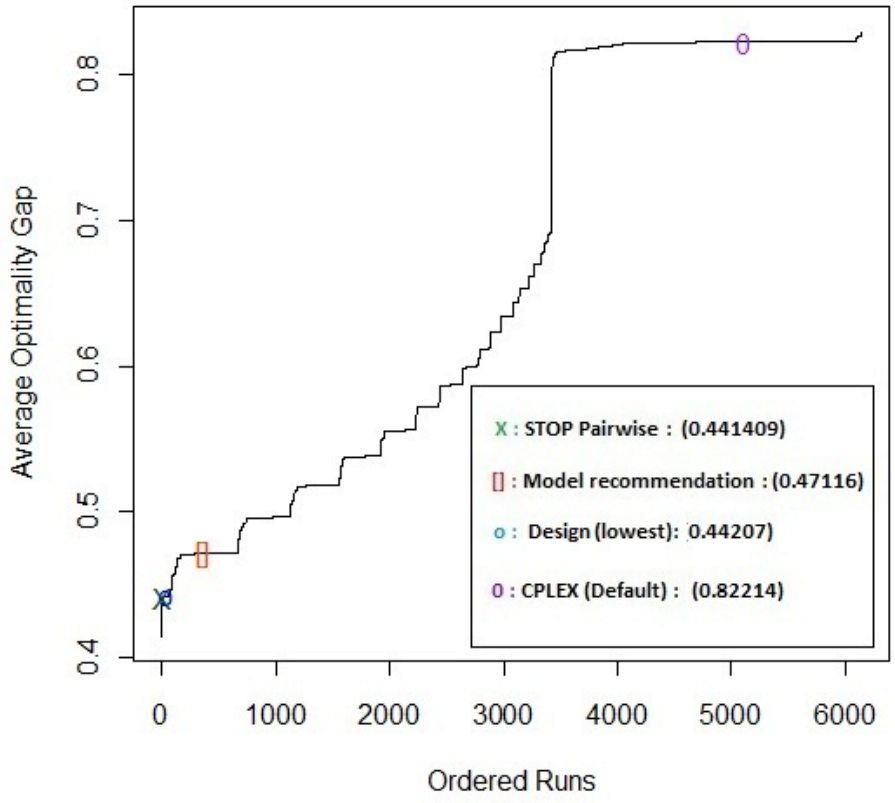


Figure 3.4: Average optimality gap of limited set CPLEX parameters experiment for hard instance

also be attained in the similar way by looking at the profiler.

The Prediction Profiler is a way to interactively change levels of a parameter and look at the effects on the predicted response. For example, from the Figure 3.6, the best settings are (2 1 0 0 2 2), as it gives the best optimality gap, these settings are obtained by following the steps from the section 2.2.1 and also we can say that the parameter MIP emphasis switch provides the desirable result when set at 2, as the log(Gap) for the settings (2 1 0 0 2 2)

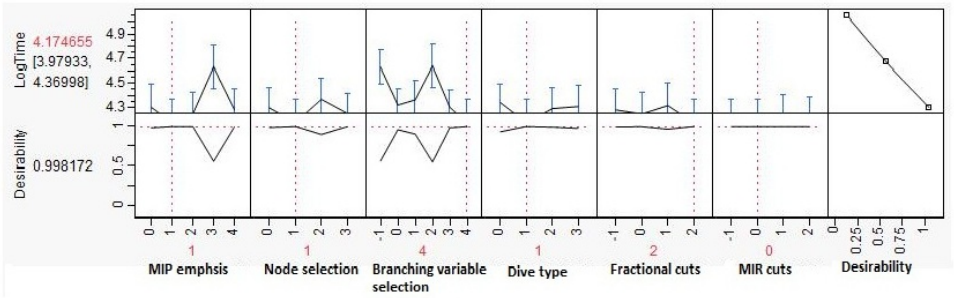


Figure 3.5: Parameter setting for the six parameters for Easy instance with solution time as response

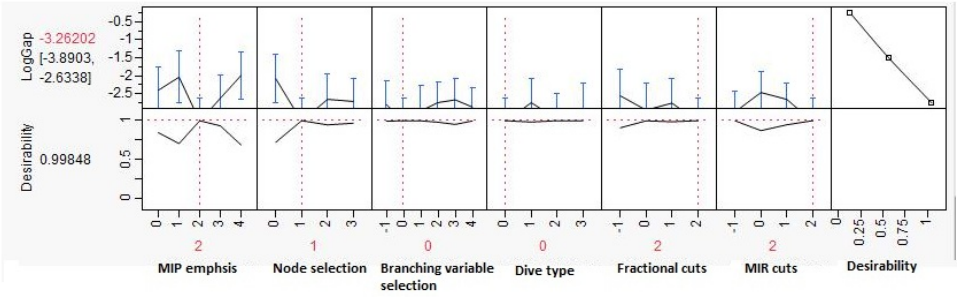


Figure 3.6: Parameter setting for the six parameters for medium instance with optimality gap as response

in the Figure 3.6 shows -3.26202 which is same as optimality gap of 3.83%. Move the red dotted lines to see the effect that changing the parameters value has on the response. Click the red line in the time graph and drag it right and left to see the effect. When the dotted red line for the parameter MIP emphasis switch is moved to 1, i.e., for setting (1 1 0 0 2 2) Figure 3.7 indicates the log(Gap) as -2.03526, which is same as optimality gap of 13.064%, which clearly shows the large percent of increase in optimality gap, this suggests that the level 1 is not a recommended setting for the parameter MIP emphasis switch. In the similar way, we can also check the effects for the other parameters. For all the remaining class of instances, the profiler results are explained in Appendix C.

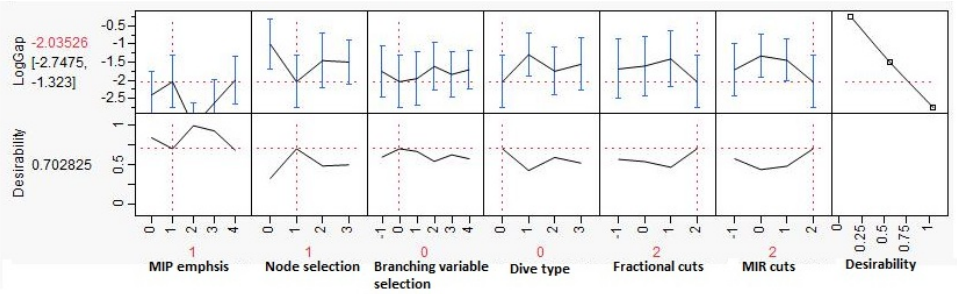


Figure 3.7: Parameter setting for the six parameters for medium instance at different level with optimality gap as response

3.1.2 Full Set Experiment

All the 59 CPLEX parameters in Appendix A are considered in the full case and the method for full case mentioned in the section 2.2.3 is applied. In the screening design, for constructing a D-optimal design by considering the two extreme levels of all 59 parameters, a minimum of 60 runs are needed to fit a model with all the main effects. We selected 64 runs, which is greater than the minimum runs required, suggested by JMP -Design Generation panel as the default runs.

The important parameters selected from the screening design are different for each class of instance, due to which the number of runs for constructing the follow-up design varies for all the three class of instances.

Table 3.5: Settings identified in full set experiment- easy class of instance with solution time as the response

Easy(Time)	Best settings
$CPX\_PARAM\_SIFTALG$	0
$CPX\_PARAM\_HEURFREQ$	-1
CPX_PARAM_SUBALG	0
CPX_PARAM_PREIND	1
CPX_PARAM_REDUCE	1
$CPX\_PARAM\_THREADS$	1
Solution time (s)	0.6

Solution Time

Easy Class of Instance

For the easy instance, the important parameters selected from the screening design are shown in Table 3.5, parameter selection process is shown in section 2.2.3. With these parameters a follow-up second-order D-optimal design is constructed. For constructing the design, a minimum of 134 runs are needed to fit a second order model with all the main effects and two factor interactions, we selected 144 runs, suggested by JMP -Design Generation panel as the default runs. A second-order regression model is fit to the log of the median solution times across the 20 instances.

Then the second-order model is used to identify the setting that would give the faster predicted solution time and the recommended settings for easy class of instance are shown in Table 3.5 The value obtained for this setting is 0.6s.

Table 3.6: Settings identified in full set experiment- medium class of instance with solution time as response

Medium(Time)	Best settings
$CPX\_PARAM\_AGGCUTLIM$	100
CPX_PARAM_REINV	10000
$CPX\_PARAM\_HEURFREQ$	100
CPX_PARAM_DEPIND	0
CPX_PARAM_PREIND	0
$CPX\_PARAM\_THREADS$	2
Solution time (s)	16

Medium Class of Instance

For the medium instance, the parameters selected are shown in Figure 3.6, for constructing a second-order D-optimal model, a minimum of 201 runs are needed, we used 240 runs, suggested by JMP -Design Generation panel as the default runs and a second-order regression model is fit to the log of the median solution times across the 5 instances. The model includes all the main effects and the second order interaction terms of the six CPLEX parameters considered.

Then the second-order model is used to identify the setting that would give the faster predicted solution time and the recommended settings for medium in Table 3.6. The value obtained for this setting is 16s.

Table 3.7: Settings identified in full set- medium class of instance with average gap as the response

Medium(Gap)	Best settings
$CPX\_PARAM\_AGGCUTLIM$	10000
CPX_PARAM_BRDIR	-1
$CPX\_PARAM\_HEURFREQ$	10
$CPX\_PARAM\_BNDSTRENIND$	0
$CPX\_PARAM\_MIPCBREDLP$	0
$CPX\_PARAM\_THREADS$	4
Optimality gap	4.86%

Optimality Gap

Medium Class of Instance

Medium class of instance is again considered for optimality gap. A follow-up second-order D-optimal design is constructed with the important parameters selected from the screening design, shown in Table 3.7. A minimum of 133 runs are required to construct a second-order model with main effects and the interaction terms of the six parameters selected in screening design. we selected 180 runs, which is more than the minimum required runs, suggested by JMP - Design Generation panel and a second-order regression model is fit to the log of the average optimality gap across the 5 instances.

Then the second-order model is used to identify the setting that would give the smaller optimality gap and the recommended settings for medium class of instance are shown in Table 3.7. The values given for this settings is 4.86%.

Table 3.8: Settings identified in full set experiment- hard class of instance with average gap as the response

Hard(Gap)	Best settings
$CPX\_PARAM\_MIPEMPHASIS$	1
$CPX\_PARAM\_NODESEL$	1
$CPX\_PARAM\_AGGCUTLIM$	100
$CPX\_PARAM\_FRACCAND$	10
$CPX\_PARAM\_HEURFREQ$	10
CPX_PARAM_REDUCE	3
Optimality gap	74.86%

Hard Class of Instance

A follow-up second-order D-optimal design is constructed with the important parameters selected from the screening design, shown in Table 3.8. A minimum of 183 runs are required to construct a second-order model with main effects and the interaction terms. we selected 240 runs, which is more than the required runs, suggested by JMP - Design Generation panel and a second-order regression model is fit to the log of the average optimality gap across the 3 instances.

Then the second-order model is used to identify the setting that would give the smaller optimality gap and the recommended settings for hard class of instance is shown in Table 3.8. The values given for this settings is 74.86%.

Comparison

Then the CPLEX Full case results are compared with pairwise method. For this, 143 runs generated by pairwise selection algorithm (Cohen et al., 1997) with all the 59 parameters, and the fastest run in each of the sets was identified for easy and medium class of instance and smallest average gap was selected for medium and hard class of instance. The faster predicted solution time and smaller gap for the default settings are also noted for the 59 CPLEX parameters.

The values for pairwise method, CPLEX Full case model and default values for all the three classes of instance can be seen in Table 3.9.

Table 3.9: Best values for pairwise method and default settings

Class of Instance	Pairwise method values	CPLEX Full case model values	default settings values
Easy(time)	0.55s	0.60s	1.02s
Medium(time)	23.90s	16.00s	801.58s
Medium(gap)	0%	4.86%	15.16%
Hard(gap)	45.66%	74.86%	121.4%

By looking at the Table 3.9, the recommendation identified by the model still beats the default average time. When model is compared with the pairwise heuristic results, for easy class of instance, pairwise heuristic does better than the model, but for the medium class of instance with solution time as the response, model does better than pairwise heuristic. In case of optimality gap, pairwise heuristic does better than the model. However, the model does better than the default settings and has the added benefit of providing information

about the effects of the each parameter value on solution time or optimality gap across the instance classes, which can be explained through Figure 3.8 for medium class of instance

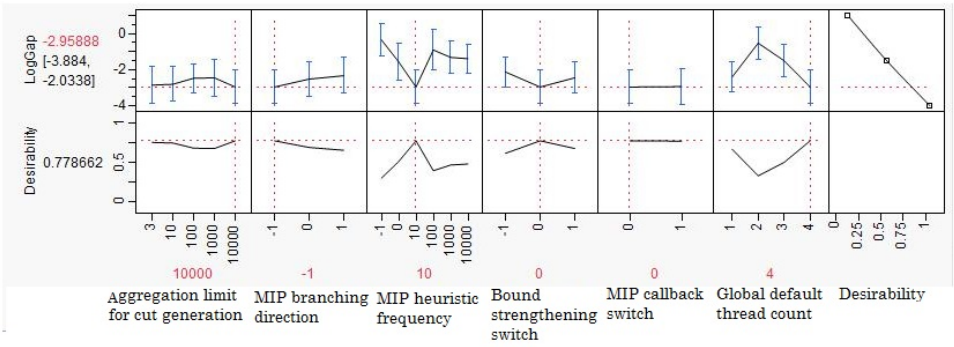


Figure 3.8: Parameter setting for the six parameters for medium instance with optimality gap as response

with optimality gap as the response, which indicates the best settings as (10000 -1 10 0 0 4) for the corresponding six parameters, as these settings results in the lowest optimality gap which is desirable and these setting are obtained by following the steps mentioned in section 2.2.1. The effect of the parameters can also be explained from the profiler in Figure 3.8. For example, parameter constraint aggregation limit provides the best optimality gap at level 10000 and when it is set at 1000, optimality gap is increasing which is not desirable, which indicates that 1000 is not the good setting for the parameter constraint aggregation limit, the parameter MIP branching direction provides the best optimality gap when set at level -1 and the optimality gap is high when set at level 1, which indicates that the level 1 is not a good setting for MIP branching direction. The effects of the remaining parameters can also be explained in the similar way by looking at the profiler.

Chapter 4

Conclusions

The paper presented a method to find the best parameter settings using design of experiments. The method is based on the key observation that models with different classes of instances, can result in good settings and as well as it also provides the effects and significance of the parameters.

Two experiments were conducted, one with a small subset of CPLEX parameters, and another with 59 CPLEX parameters. In the small subset experiment, six important parameters were considered and a D-optimal second-order model was constructed to identify the best settings. Both the optimal small subset model and pairwise heuristic method gave the results which are much better than default settings and when the pairwise heuristic method is compared with the optimal small subset model, both the methods, are pretty much equally good.

In the full set experiment, all the 59 parameters were considered and a screening design was constructed using JMP with two extreme levels of all the 59 parameters and then follows with a follow-up design with the best selected parameters from the screening design for all the three instance classes and the best setting were identified. We see some differences in the full set experiment when compared with pairwise heuristic method, but there is no clear winner and both the methods are better than the default.

Pairwise heuristic method is not desirable when the parameter effects are needed by the user, this method is useful when the large number of parameters are tuned at once, to get the best settings, but when the parameter effects is not desired.

Model provides an explicable effects of each parameter levels, which may be useful in understanding how instance classes interact with the various parameters. The effects of all the parameters is easily visualized by examining the parameters interactively with the Prediction Profiler.

Building different models with different modelling techniques and using different design procedures could result in even better results. For the limited set CPLEX parameters experiment, identifying some more important parameters which could effect the response and applying the method could improve the results. Also, the method can be extended with the other different metrics, such as instead of time difference the maximum time for different instances.

Acknowledgements

I would like to thank my advisor's, Dr. J. Paul Brooks and Dr. David J. Edwards, who were abundantly helpful and offered invaluable assistance, support and guidance. Also, special thanks to my family for their understanding through out my duration of studies.

Bibliography

[1] Abraham,B.,Chipman,H. and Vijayan,K.(1999) *Some Risks in the Construction and Analysis of Supersaturated Designs.* University of Waterloo,Canada.

[2] Adenzo-Díaz, B. and Laguna, M. (2004) *Fine-tuning of Algorithms Using Fractional Experimental Designs and Local Search.*

[3] Atamtürk, A. and Savelsbergh, M. W. P. (2005) *Integer-Programming Software Systems.* Annals of Operations Research, 140, 67Ű124.

[4] Atkinson, A. C., Donev, A. N., and Tobias, R. D. (2007) *Optimum Experimental Designs with SAS.* vol. 34 of Oxford Statistical Science Series, Oxford University Press.

[5] Audet, C. and Orban, D. (2006) *Finding Optimal Algorithmic Parameters Using Derivative Free Optimization.* SIAM Journal on Optimization, 17, 642–664.

[6] Bartz-Beielstein T.(2006) *Experimental Research in Evolutionary Computation: The New Experimentalism.* Springer Verlag.

[7] Baz, M., Brady Hunsaker, J. Paul Brooks and Abhijit Gosavi(2007) *Automated Tuning of Optimization Software Parameters.* University of Pittsburgh.

[8] Baz, M., Hunsaker, B., and Prokopyev, O. (2009) *How much do we "pay" for using default parameters?*. Computational Optimization and Applications.

[9] Bixby, R. E. (2002) *Solving Real-World Linear Programs: A Decade and More of Progress*. Operations Research, 50, 3–15.

[10] Booth, K.H.V., and Cox, D.R. (1962) *Some systematic supersaturated designs*. Technometrics4,489-495.

[11] Boyd, Stephen P.,Lieven Vandenberghe (2009) *Convex Optimization*. Cambridge University Press.

[12] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. (1984) *Classification and Regression Trees*. Trees, Boulder: Westview.

[13] J. Paul Brooks, William P. Burns, Stephen S. Fong, Chris M. Gowen, Seth B. Roberts. (2012) *Gap Detection for Genome-Scale Constraint-Based Models*, accepted to Advances in Bioinformatics.

[14] Charles R. Stewart (2010) *Automated Selection of Mixed Integer Program Solver Parameter Settings*.

[15] Cohen, D. M., Dalal, S. R., Fredman, M. L., and Patton, G. C. (1997) *The AETG System: An Approach to Testing Based on Combinatorial Design*. IEEE Transactions on Software Engineering, 23, 437–444.

[16] CPL Ibmilog CPLEX,(2009) `http://www-01.ibm.com/software/integration/optimization/cplex`

[17] Cristianini, N. and Shawe-Taylor, J. (2000) *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge, UK: Cambridge University Press.

[18] DuMouchel,W. and Jones, B. (1994) *A Simple Bayesian Modification of D-Optimal Designs to Reduce Dependence on an Assumed Model.* Technometrics, 36, 37–47.

[19] Eriksson, L. (2008) *Design of experiments: principles and applications.* Umetrics Academy publisher

[20] Fabian Triefenbach (2008) *Design of Experiments:The D-Optimal Approach and Its Implementation As a Computer Algorithm.* Umea University,sweden.

[21] Hamadam, S. and Wu, C. F. J.(1992) *Analysis of designed experiments with complex aliasing.* J. Qual. Technometrics,24, 130-7.

[22] Hans D. Mittelmann and Armin Pruessner(2006) *Optimization Methods and Software.* Taylor and Francis ,Vol. 21, No. 1.

[23] Huang, D. T. T. Allen, W. I. Notz, and N. Zeng(2006) *Global optimization of stochastic black-box systems via sequential kriging meta-models.* Journal of Global Optimization, 34(3):441–466.

[24] Hutter, F., Hoos, H. H., Leyton-Brown, K., and Murphy, K. P. (2009) *An Experimental Investigation of Model-Based Parameter Optimisation: SPO and Beyond.* in GECCO 09,Genetic and Evolutionary Computation.

[25] Imbault, F. and Lebart, K(2004) *A stochastic optimization approach for parameter tuning of Support Vector Machines.* International Conference on Pattern Recognition.

[26] JMP,(2005) *JMP Design of Experiments, Release 6.* SAS Institute Inc., Cary, NC, USA.

[27] Kohavi, R. and John, G. H. (1995) *Automatic Parameter Selection by Minimizing Estimated Error,in Machine Learning.* Proceedings of the Twelfth International Conference.

[28] Lin, D. K. J. (1993) *A new class of supersaturated designs.* Technometrics 35, 28-3 1.

[29] Lin, D. K. J. (1995) *Generating supersaturatedd systematic designs.* Technometrics,37, 213-225.

[30] Laundy, R., Perregaard, M., Tavares, G., Tipi, H., and Vazacopoulos, A. (2007) *Solving hard Mixed Integer Programming Problems With Xpress-MP: A MIPLAB 2003 Case Study.* Tech. Rep. 2, Rutgers Center for Operations Research.

[31] Meyer, R. K. , Nachtsheim, C. J. (1995) *The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs.* Technometrics 37(1), 60-69.

[32] Miller, A. J.(1990) *Subset Selection in Regression.* London: Chapman and Hall.

[33] Mitchell,T. M.(1982) *Generalization as search.* Artificial Intelligence 18,203-266.

[34] Montgomery, D. C. (2008) *Design and analysis of experiments.* Hoboken, NJ: Wiley.

[35] Raymond H. Myers,Douglas C. Montgomery, Anderson-Cook (2009) *Response Surface Methodology: Process and Product Optimization Using Designed Experiments.* Third Edition, NJ: Wiley.

[36] Nemhauser, G. L., L. A. Wolsey(1988) *Integer and Combinatorial Optimization.* John Wiley and Sons, Inc., New York.

[37] Satterthwaite, F. (1959) *Random Balance Experimentation.* Technometrics, 1, 111-137

[38] Tang, B., and Wu, C.F.J. (1993) *A method for constructing supersaturatedd esigns and its $E(s^2)$ optimality.*

[39] Thorsten Koch, T. Berthold, G. Gamrath, A. M. Gleixner, S. Heinz, D. E. Steffy, K. Wolter, T. Achterberg, E. Andersen, O. Bastert, R. E. Bixby, E. Danna, A. Lodi, H. Mittelmann, T. Ralphs and D. Salvagnin (2011) *Mixed Integer Programming Library version 5.* Math. Prog. Comp. (2011) 3:103–163.

[40] Wu, C.F.J. (1993) *Construction of supersaturated designs through partially aliased interactions.* Biometrika,80, 661-669.

[41] Wu, C. F. J. and Hamada, M. (2000) *Design of Experiments: Planing, Analysis and Parameter Design Optimization.* John Wiley and Sons, INC., USA.

[42] Yasser Tanvir (2009) *A System for Optimal Telecommunication Network Installation.*

Appendix A

List of CPLEX parameters

The 59 CPLEX parameters used in the full case are mentioned below, with all the levels of each parameter and the default setting using information from CPL (2009).

Description of CPLEX parameters

The description of the 59 CPLEX parameters from the above table are described here, using information from CPL (2009).

1) $CPX\_PARAM\_MIPEMPHASIS$: MIP emphasis switch Controls trade-offs between speed, feasibility, optimality, and moving bounds in MIP.

2) $CPX\_PARAM\_NODESEL$: MIP node selection strategy is used to set the rule for selecting the next node to process when backtracking.

3) CPX_PARAM_VARSEL : MIP variable selection strategy sets the rule for selecting the branching variable at the node which has been selected for branching.

4) $CPX\_PARAM\_DIVETYPE$: MIP dive strategy controls the MIP dive strategy. The MIP traversal strategy occasionally performs probing dives, where it looks ahead at both children nodes before deciding which node to choose.

5) $CPX\_PARAM\_FRACCUTS$: MIP Gomory fractional cuts switch decides whether or not Gomory fractional cuts should be generated for the problem.

6) $CPX\_PARAM\_MIRCUTS$: MIP MIR (mixed integer rounding) cut switch decides whether or not to generate MIR cuts (mixed integer rounding cuts) for the problem.

7) $CPX\_PARAM\_AGGCUTLIM$: Constraint aggregation limit for cut generation limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding (MIR) cuts.

8) $CPX\_PARAM\_CLIQUES$: MIP cliques switch decides whether or not clique cuts should be generated for the problem.

9) CPX_PARAM_COVERS : MIP covers switch decides whether or not cover cuts should be generated for the problem.

10) $CPX\_PARAM\_CUTPASS$: Number of cutting plane passes sets the upper limit on the number of cutting plane passes CPLEX performs when solving the root node of a MIP model.

11) $CPX\_PARAM\_CUTSFACTOR$: Row multiplier factor for cuts limits the number of cuts that can be added. The number of rows in the problem with cuts added is limited to $CutsFactor$ times the original number of rows. If the problem is presolved, the original number of rows is that from the presolved problem.

12) $CPX\_PARAM\_DISJCUTS$: MIP disjunctive cuts switch decides whether or not

disjunctive cuts should be generated for the problem.

13) $CPX\_PARAM\_EACHCUTLIM$: Type of cut limit sets a limit for each type of cut. This parameter allows you to set a uniform limit on the number of cuts of each type that CPLEX generates.

14) $CPX\_PARAM\_FLOWCOVERS$: MIP flow cover cuts switch decides whether or not to generate flow cover cuts for the problem.

15) $CPX\_PARAM\_FLOWPATHS$: MIP flow path cut switch decides whether or not flow path cuts should be generated for the problem.

16) $CPX\_PARAM\_FRACCAND$: Candidate limit for generating Gomory fractional cuts limits the number of candidate variables for generating Gomory fractional cuts

17) $CPX\_PARAM\_FRACPASS$: Pass limit for generating Gomory fractional cuts limits the number of passes for generating Gomory fractional cuts.

18) $CPX\_PARAM\_GUBCOVERS$: MIP GUB cuts switch decides whether or not to generate GUB cuts for the problem.

19) CPX_PARAM_IMPLBD : MIP implied bound cuts switch decides whether or not to generate implied bound cuts for the problem.

20) $CPX\_PARAM\_ZEROHALFCUTS$: MIP zero-half cuts switch decides whether or not to generate zero-half cuts for the problem.

21) CPX_PARAM_CRAIND : Simplex crash ordering decides how CPLEX orders variables relative to the objective function when selecting an initial basis.

22) $CPX\_PARAM\_DPRIIND$: Dual simplex pricing algorithm decides the type of pricing applied in the dual simplex algorithm.

23) CPX_PARAM_PERIND (int) : Simplex perturbation switch decides whether to perturb problems.

24) CPX_PARAM_PERLIM : Simplex perturbation limit sets the number of degenerate iterations before perturbation is performed.

25) $CPX\_PARAM\_PPRIIND$: Primal simplex pricing algorithm sets the primal simplex pricing algorithm.

26) CPX_PARAM_REINV : Simplex refactoring frequency sets the number of iterations between refactoring of the basis matrix.

27) $CPX\_PARAM\_SIFTALG$: Sifting subproblem algorithm sets the algorithm to be used for solving sifting subproblems.

28) $CPX\_PARAM\_SINGLIM$: Simplex singularity repair limit restricts the number of times CPLEX attempts to repair the basis when singularities are encountered during the simplex algorithm.

29) $CPX\_PARAM\_STRONGCANDLIM$: MIP strong branching candidate list limit controls the length of the candidate list when CPLEX uses strong branching as the way to select variables.

30) $CPX\_PARAM\_STRONGITLIM$: MIP strong branching iterations limit controls the number of simplex iterations performed on each variable in the candidate list when CPLEX uses strong branching as the way to select variables.

31) CPX_PARAM_SCAIND : Scale parameter decides how to scale the problem matrix.

32) $CPX\_PARAM\_BBINTERVAL$: MIP strategy best bound interval sets the best

bound interval for MIP strategy.

33) CPX_PARAM_BRDIR : MIP branching direction decides which branch, the up or the down branch, should be taken first at each node.

34) CPX_PARAM_FPHEUR : Feasibility pump switch turns on or off the feasibility pump heuristic for mixed integer programming (MIP) models.

35) $CPX\_PARAM\_HEURFREQ$: MIP heuristic frequency decides how often to apply the periodic heuristic.

36) $CPX\_PARAM\_LBHEUR(int)$: Local branching heuristic controls whether CPLEX applies a local branching heuristic to try to improve new incumbents found during a MIP search.

37) $CPX\_PARAM\_MIPSEARCH$: MIP dynamic search switch sets the search strategy for a mixed integer program (MIP).

38) CPX_PARAM_SUBALG : MIP subproblem algorithm decides which continuous optimizer will be used to solve the subproblems in a MIP, after the initial relaxation.

39) $CPX\_PARAM\_PARALLELMODE$: Parallel mode switch sets the parallel optimization mode. Possible modes are automatic, deterministic, and opportunistic.

40) CPX_PARAM_PROBE : MIP probing level sets the amount of probing on variables to be performed before MIP branching. Higher settings perform more probing. Probing can be very powerful but very time-consuming at the start.

41) $CPX\_PARAM\_RINSHEUR$: RINS heuristic frequency decides how often to apply the relaxation induced neighborhood search (RINS) heuristic.

42) $CPX\_PARAM\_STARTALG$: MIP starting algorithm sets which continuous opti-

mizer will be used to solve the initial relaxation of a MIP.

43) $CPX\_PARAM\_AGGFILL$: Preprocessing aggregator fill limits variable substitutions by the aggregator. If the net result of a single substitution is more nonzeros than this value, the substitution is not made.

44) CPX_PARAM_AGGIND : Preprocessing aggregator application limit invokes the aggregator to use substitution where possible to reduce the number of rows and columns before the problem is solved.

45) $CPX\_PARAM\_BNDSTRENIND$: Bound strengthening switch decides whether to apply bound strengthening in mixed integer programs (MIPs).

46) $CPX\_PARAM\_COEREDIND$: Coefficient reduction setting decides how coefficient reduction is used. Coefficient reduction improves the objective value of the initial (and subsequent) LP relaxations solved during branch and cut by reducing the number of non-integral vertices.

47) CPX_PARAM_DEPIND : Dependency switch decides whether to activate the dependency checker.

48) $CPX\_PARAM\_MIPCBREDLP$: MIP callback switch between original model and reduced, presolved model controls whether your callback accesses node information of the original model (off) or node information of the reduced, presolved model (on, default).

49) $CPX\_PARAM\_PROBETIME$: Time spent probing limits the amount of time in seconds spent probing.

50) $CPX\_PARAM\_PREDUAL$: Presolve dual setting Decides whether CPLEX presolve should pass the primal or dual linear programming problem to the linear programming op-

timization algorithm.

51) $CPX\_PARAM\_PREIND(int)$: Presolve switch decides whether CPLEX applies presolve during preprocessing.

52) $CPX\_PARAM\_PRELINEAR$: Linear reduction switch decides whether linear or full reductions occur during preprocessing.

53) $CPX\_PARAM\_PREPASS$: Limit on the number of presolve passes made limits the number of presolve passes that CPLEX makes during preprocessing.

54) $CPX\_PARAM\_PRESLVND$: Node presolve switch decides whether node presolve should be performed at the nodes of a mixed integer programming (MIP) solution.

55) CPX_PARAM_REDUCE : Primal and dual reduction type decides whether primal reductions, dual reductions, both, or neither are performed during preprocessing.

56) $CPX\_PARAM\_RELAXPREIND$: Relaxed LP presolve switch decides whether LP presolve is applied to the root relaxation in a mixed integer program (MIP). Sometimes additional reductions can be made beyond any MIP presolve reductions that were already done.

57) $CPX\_PARAM\_REPEATPRESOLVE$: Reapply presolve after processing the root node decides whether to re-apply presolve, with or without cuts, to a MIP model after processing at the root is otherwise complete.

58) $CPX\_PARAM\_SYMMETRY$: Symmetry breaking decides whether symmetry breaking reductions will be automatically executed, during the preprocessing phase, in a MIP model.

59) $CPX\_PARAM\_THREADS$: Global default thread count sets the default number of

parallel threads that will be invoked by any CPLEX parallel optimizer.

Table A.1: CPLEX 59 parameters

Number	Parameter	Levels	Default
1	$CPX\_PARAM\_MIPEMPHASIS$	[0,1,2,3,4]	0
2	$CPX\_PARAM\_NODESEL$	[0,1,2,3]	1
3	CPX_PARAM_VARSEL	[-1,0,1,2,3,4]	0
4	$CPX\_PARAM\_DIVETYPE$	[0,1,2,3]	0
5	$CPX\_PARAM\_FRACCUTS$	[-1,0,1,2]	0
6	$CPX\_PARAM\_MIRCUTS$	[-1,0,1,2]	0
7	$CPX\_PARAM\_AGGCUTLIM$	[3,10,100,1000,10000]	3
8	$CPX\_PARAM\_CLIQUES$	[-1,0,1,2]	0
9	CPX_PARAM_COVERS	[-1,0,1,2]	0
10	$CPX\_PARAM\_CUTPASS$	[-1,0,10,100,1000,10000]	0
11	$CPX\_PARAM\_CUTSFACTOR$	[4,10]	4
12	$CPX\_PARAM\_DISJCUTS$	[-1,0,1,2]	0
13	$CPX\_PARAM\_EACHCUTLIM$	[0,10,100,1000,10000,2100000000]	2100000000
14	$CPX\_PARAM\_FLOWCOVERS$	[-1,0,1,2]	0
15	$CPX\_PARAM\_FLOWPATHS$	[-1,0,1,2]	0
16	$CPX\_PARAM\_FRACCAND$	[10,100,200,1000,10000]	200
17	$CPX\_PARAM\_FRACPASS$	[0,10,100]	0
18	$CPX\_PARAM\_GUBCOVERS$	[-1,0,1,2]	0
19	CPX_PARAM_IMPLBD	[-1,0,1,2]	0
20	$CPX\_PARAM\_ZEROHALFCUTS$	[-1,0,1,2]	0
21	CPX_PARAM_CRAIND	[0,1]	1
22	$CPX\_PARAM\_DPRIIND$	[0,1,2]	0
23	CPX_PARAM_PERIND (int)	[0,1]	0

Number	Parameter	Levels	Default
24	CPX_PARAM_PERLIM	[0,10,100,1000,10000]	0
25	$CPX\_PARAM\_PPRIIND$	[0,1,2]	0
26	CPX_PARAM_REINV	[0,10,100,1000,10000]	0
27	$CPX\_PARAM\_SIFTALG$	[0,1,2]	0
28	$CPX\_PARAM\_SINGLIM$	[10,100,1000,10000]	10
29	$CPX\_PARAM\_STRONGCANDLIM$	[10,100,1000,10000]	10
30	$CPX\_PARAM\_STRONGITLIM$	[0,10,100,1000,10000]	0
31	CPX_PARAM_SCAIND	[-1,0,1]	0
32	$CPX\_PARAM\_BBINTERVAL$	[0,1,7,10,100,1000,10000]	7
33	CPX_PARAM_BRDIR	[-1,0,1]	0
34	CPX_PARAM_FPHEUR	[-1,0,1,2]	0
35	$CPX\_PARAM\_HEURFREQ$	[-1,0,10,100,1000,10000]	0
36	$CPX\_PARAM\_LBHEUR(int)$	[0,1]	0
37	$CPX\_PARAM\_MIPSEARCH$	[0,1,2]	0
38	CPX_PARAM_SUBALG	[0,1,2,5]	0
39	$CPX\_PARAM\_PARALLELMODE$	[-1,0,1]	0
40	CPX_PARAM_PROBE	[-1,0,1,2,3]	0
41	$CPX\_PARAM\_RINSHEUR$	[-1,0,10,100,1000,10000]	0
42	$CPX\_PARAM\_STARTALG$	[0,1,2,5,6]	0
43	$CPX\_PARAM\_AGGFILL$	[10,100,1000,10000]	10
44	CPX_PARAM_AGGIND	[-1,0,10,100,1000,10000]	-1
45	$CPX\_PARAM\_BNDSTRENIND$	[-1,0,1]	-1
46	$CPX\_PARAM\_COEREDIND$	[0,1,2]	2

Number	Parameter	Levels	Default
47	CPX_PARAM_DEPIND	[-1,0,1,2,3]	-1
48	$CPX\_PARAM\_MIPCBREDLP$	[0,1]	1
49	$CPX\_PARAM\_PROBETIME$	[0,10,100,1000,10000,1E+75]	1.00E+75
50	$CPX\_PARAM\_PREDUAL$	[-1,0,1]	0
51	$CPX\_PARAM\_PREIND(int)$	[0,1]	1
52	$CPX\_PARAM\_PRELINEAR$	[0,1]	1
53	$CPX\_PARAM\_PREPASS$	[-1,0,10,100,1000,10000]	-1
54	$CPX\_PARAM\_PRESLVND$	[-1,0,1,2]	0
55	CPX_PARAM_REDUCE	[0,1,2,3]	3
56	$CPX\_PARAM\_RELAXPREIND$	[0,1]	-1
57	$CPX\_PARAM\_REPEATPRESOLVE$	[-1,0,1,2,3]	-1
58	$CPX\_PARAM\_SYMMETRY$	[-1,0,1,2,3,4,5]	-1
59	$CPX\_PARAM\_THREADS$	[0,1,4]	0

Appendix B

CPLEX Limited Case D-Optimal Model

Profiler settings

Easy(time)

Figure B.1 shows the best settings as (1 1 4 1 2 0).

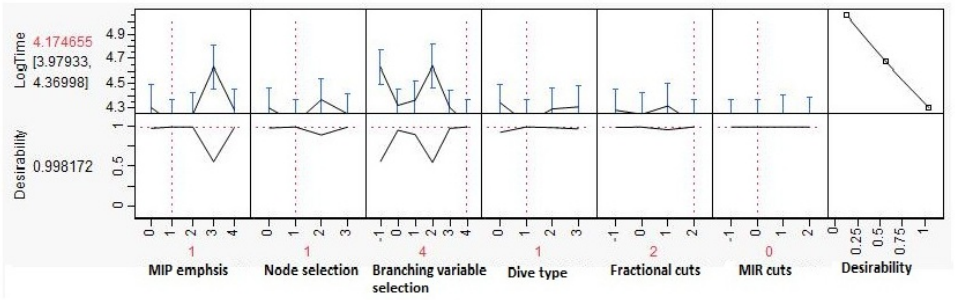


Figure B.1: Parameter setting for the six parameters for easy instance with solution time as response

Medium(time)

Figure B.2 shows the best settings as (2 2 -1 0 1 -1).

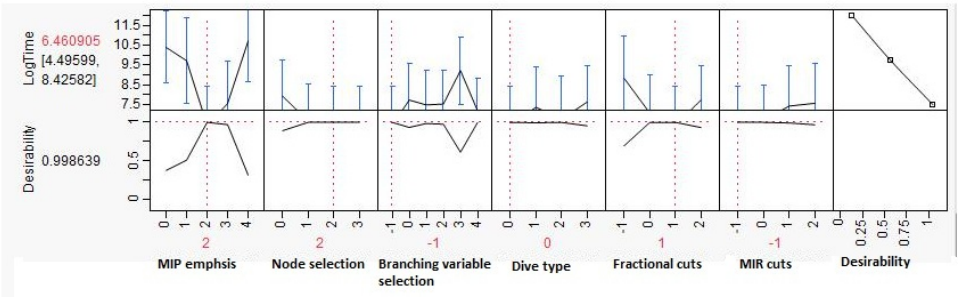


Figure B.2: Parameter setting for the six parameters for medium instance with solution time as response

Medium(gap)

Figure B.3 shows the best settings as (2 1 0 0 2 2).

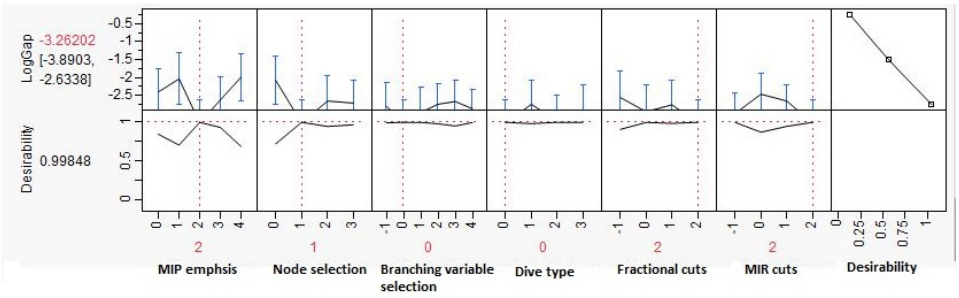


Figure B.3: Parameter setting for the six parameters for medium instance with optimality gap as response

Hard(gap)

Figure B.4 shows the best settings as (4 1 4 1 1 1).

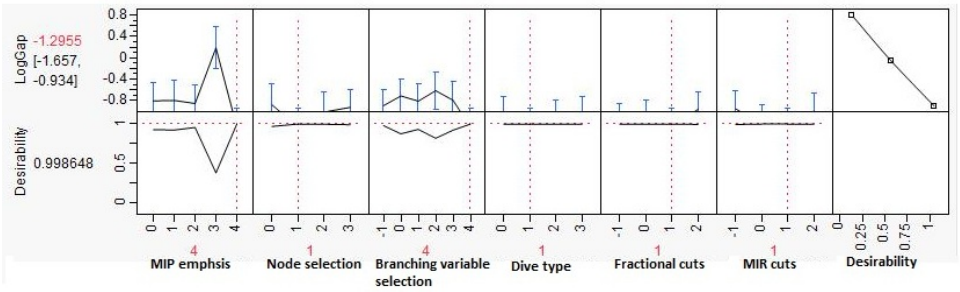


Figure B.4: Parameter setting for the six parameters for hard instance with optimality gap as response

Appendix C

CPLEX Full Case D-Optimal Model

Profiler settings

Easy(time)

Figure C.1 shows the best settings as (0 -1 0 1 1 1).

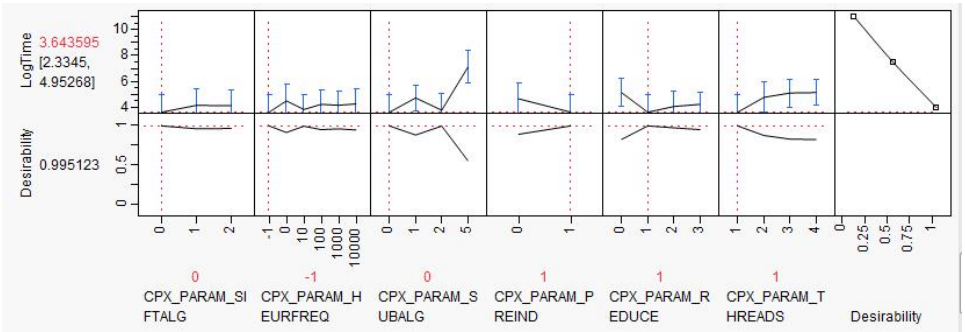


Figure C.1: Parameter setting for the six parameters for easy instance with solution time as response

Medium(time)

Figure C.2 shows the best settings as (100 10000 100 0 0 2).

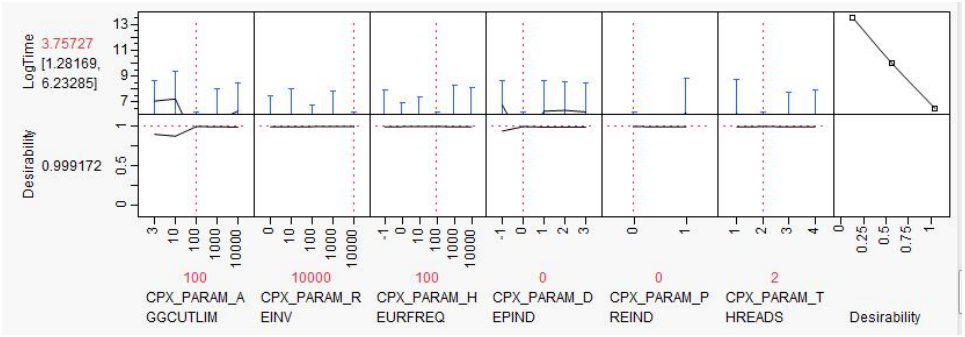


Figure C.2: Parameter setting for the six parameters for medium instance with solution time as response

Medium(gap)

Figure C.3 shows the best settings as (10000 -1 10 0 0 4).

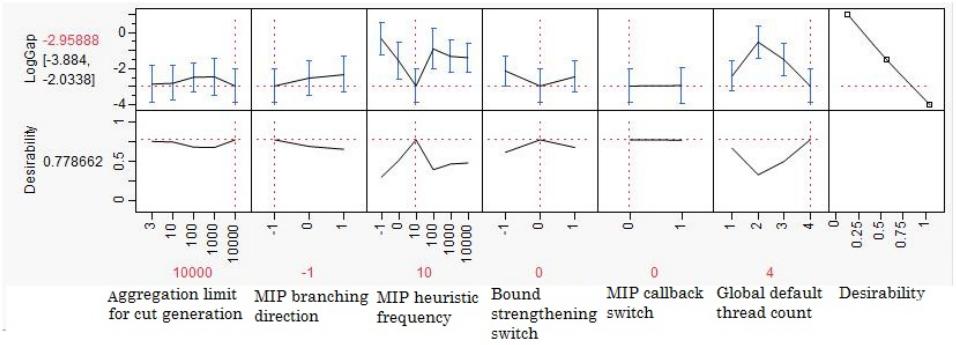


Figure C.3: Parameter setting for the six parameters for medium instance with optimality gap as response

Hard(gap)

Figure C.4 shows the best settings as (1 1 100 10 10 3).

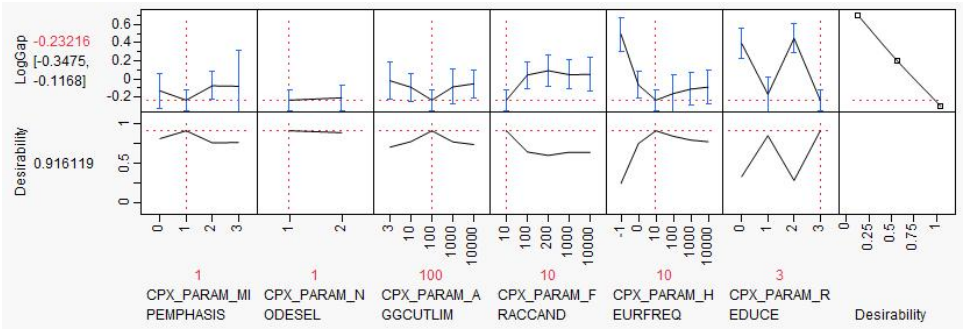


Figure C.4: Parameter setting for the six parameters for hard instance with optimality gap as response

Vita

Radha S Koripalli was born March 12, 1985 in Visakhapatnam,India. Ms. Radha received her Bachelor of Science in Electronics and Communication Engineering from Jawaharlal nehru university, India in 2006.