International Journal of Applied Metaheuristic Computing
Volume 13 • Issue 1

Application of Metaheuristic Approaches for the Variable Selection Problem

Myung Soon Song, Kutztown University of Pennsylvania, USA*
Francis J. Vasko, Kutztown University of Pennsylvania, USA
Yun Lu, Kutztown University of Pennsylvania, USA
Kyle Callaghan, Kutztown University of Pennsylvania, USA

ABSTRACT

Variable selection is an old topic from regression models. Besides many conventional approaches, some metaheuristic approaches from the realm of optimization such as GA (genetic algorithm) or simulated annealing have been suggested to date. These methods have a considerable advantage to deal with many problems over the classical methods, but they must control relevant fine-tuning parameters associated with cross-over or mutation, which can be difficult and time-consuming. In this paper, Jaya, one of several parameter-free approaches will be suggested and explored. Several metaheuristic methods will be compared using results from a real-world dataset and a simulated dataset. The impact of using local search will be analyzed.

KEYWORDS

Genetic Algorithm, Jaya Metaheuristic, Local Search, Neighborhood Search, Population-Based Metaheuristics, Regression Models, Simulation, Teaching-Learning-Based Optimization Metaheuristic

INTRODUCTION

Variable selection is a classical topic in regression which has many applications in several areas including, but not limited to, engineering, medicine, psychology, or business.

Among numerous variable selection methods developed, some classical sequential methods such as stepwise selection methods (Desboulets, 2018; Lindsey and Sheather, 2010) have been widely used because they are simple and work very well if there are not too many variables and they have low prediction error. But there are some drawbacks in these methods. Two most serious issues among them are (1) they tend to converge to local optima (Hans et al., 2012; Hocking,1976; Kiezun et al., 2009; Meiri and Zahavi, 2006; Paterini and Minerva, 2010) and (2) they do not work very well in high dimensional spaces. (Hand et al., 2012; Kapetanios, 2007). Later in this section, it will be explained how these problems can be resolved with 'metaheuristics' in optimization research.

The selection of the most adequate variables in regression models can be stated as a combinatorial optimization problem with the objective to select explanatory variables that maximize the adequacy

DOI: 10.4018/IJAMC.298309
*Corresponding Author

of the model according to statistical criteria (objective function). (Meiri, 2006; Paterlini and Minerva, 2010) Some methods or algorithms from optimization research have been used for variable selection, including but not limited to, genetic algorithm (Broadhurst et al., 1997; Kapetanios, 2007; Kiezun et al., 2009; Jirapech-Umpai and Aitken, 2005; Mohan et al., 2018; Paterini and Minerva, 2010; Peng et al., 2005; Sinha et al., 2015), simulated annealing (Kiezun et al., 2009; Meiri and Zahavi, 2006), iterated local search (Hans et al., 2012). These methods are characterized as metaheuristics, a stochastic search strategy dedicated to solving difficult problems (NP-hard problems) in optimization research.

In particular, genetic algorithms (GA hereafter) and simulated annealing (SA hereafter) are known to be very effective to resolve the two issues mentioned above – (1) convergence to local optima (Kapatenios, 2007; Kiezun et al., 2006; Meiri, 2006; Paterini and Minerva, 2010) and (2) handling high dimensional spaces. (Kapatenious, 2007; Meiri, 2006) Brief descriptions of GA and SA can be found in Appendix.

Even if these metaheuristics (GA or SA) have good properties such as tending to reach the global optima and capability to deal with many variables, their performance heavily depend on the choice of 'tuning parameters', which is very experimental and time-consuming in practice. For example, the GA and SA need to fine tune four parameters (crossover type, crossover rate, mutation type, and mutation rate) and five parameters (initial temperature, final temperature, cooling ratio, temperature function, and accept function), respectively.

To resolve these obvious and practical problems, this paper will suggest using 'parameter-free metaheuristics' for variable selection in regression – Jaya (Rao, 2016) and Teaching Based Optimization (TBO hereafter) (Rao et al., 2011).

In the next section, TBO and Jaya will be briefly described.

APPROACH

What is Teaching Based Optimization?

The Teaching-learning-based optimization (TLBO) metaheuristic is a two-phase population-based metaheuristic designed to solve continuous nonlinear optimization problems. It was proposed by Rao et al. (2011) as a method for solving large constrained mechanical design optimization problems which involve no specific parameters to tune. Since the tuning of parameters in other metaheuristics can often be time consuming and largely experimental, Rao et al. (2011) describe a procedure in which the only parameters that need to be specified are those common to all other metaheuristics--population size and termination criterion.

TLBO consists of two phases referred to by Rao et al. (2011) as the teaching phase and the learning phase. The first phase of TLBO, the teaching phase, utilizes a global search procedure which really uses intensification-focused moves as discussed in Hill and Pohl (2019). The "difference mean" is created by subtracting the quality of the best solution with the current mean solution. The objective here is to improve all solutions by this difference. The operator creating a new solution in the teaching phase is given as the formula $X_{new} = X_{old} + r\left(X_{teacher} - T_f \times X_{mean}\right)$ where X_{old} is a current solution of a population being modified, r is a random number in the range [0,1], $X_{teacher}$ is the best solution of a population, T_f = round(1+rand(0.1)) implying that T_f takes on the values 1 or 2 with equal probability and X_{mean} is the mean solution of a population (Rao et al, 2011). Here, two variables r and T_f could have been used as parameters; however, they are defined as being random numbers and therefore their values are **not** specified as input parameters. The teaching phase is completed by checking if the new solution is better than the current.

The second phase of TLBO adjusts each solution relative to a randomly selected solution (another learner). The learning phase involves diversification-focused moves as discussed in Hill and Pohl (2019). The operator is given by the following (for a minimization problem):

$$X_{i,new} = \begin{cases} X_i + r\left(X_i - X_j\right), & \text{if } f\left(X_i\right) < f\left(X_j\right) \\ X_i + r\left(X_j - X_i\right), & \text{otherwise} \end{cases} \quad (1)$$

where, similar to the teaching phase, r is randomly chosen in the range of [0,1], X_i is the current solution and X_j is a randomly chosen solution where $i \neq j$. For both phases of TLBO, since X_j is a vector of real numbers, the actual implementation of TLBO requires the use of these update formulas on each component of X_j. For more information on TLBO, the authors suggest reading Rao et al. (2011).

In this paper, Teaching-based Optimization (TBO), which is a special case of TLBO with only teaching phase will be used because the learning phase will be replaced with a local search which will also be incorporated into Jaya.

What is Jaya?

The Jaya metaheuristic by Rao (2016) is a single phase population-based metaheuristic designed to solve continuous nonlinear optimization problems. It is very similar to the teaching phase of TLBO except that a different transformation formula is used to update each solution in the current population. Specifically, if $X_{j,k,i}$ is the value of the jth variable for the kth candidate solution during the ith iteration, then this value is modified based on the equation $X^{new}_{j,k,i} = X_{j,k,i} + R1_{j,i}(X_{j,best,i} - X_{j,k,i}) - R2_{j,i}(X_{j,worst,i} - X_{j,k,i})$, where $X_{j,best,i}$ is the value of the variable j for the best candidate solution in the current population and $X_{j,worst,i}$ is the value of the variable j for the worst candidate solution in the current population. $X^{new}_{j,k,i}$ is the updated value of $X_{j,k,i}$ and $R1_{j,i}$ and $R2_{j,i}$ are two random numbers for the jth variable during the ith iteration in the range [0,1]. This transformation equation is trying to move the current solution toward the best solution and away from the worst solution. The authors suggest reading Rao (2016) for more details on Jaya.

Binarization of TLBO and Jaya

Both TLBO and Jaya are designed to solve continuous nonlinear optimization problems; whereas variable selection is a zero-one constrained optimization problem (either a variable is in the model or not). The solutions in the population of a problem using the original versions of TLBO or Jaya will be vectors of real (rational) numbers. The solutions in the population for the variable selection problem are bit strings (zeros and ones). To adapt TLBO and Jaya to deal with bit strings, the authors used the approach that Lu and Vasko (2015) used successfully for the Set Covering Problem. In any of the transformation formulas (teaching, learning, or Jaya), the variables are now bits. The random numbers that took on any values between 0 and 1 now take on only 0 or 1 with equal probability. As in the original TLBO, the teaching factor in TLBO takes on the values 1 or 2 with equal probability. Also, in the teaching phase, the mean solution is replaced by the median solution. If, after a transformation formula is performed, a variable value is less than 0, it is set to 0. If it is greater than 1, it is set to 1. Intuitively, if the result of a transformation formula produces a variable that "wants" to have a value less than 0, the authors simply set it to 0. In a like manner, variables that "want" to have a value greater than 1 are set to 1. The empirical results will demonstrate that this simple binarization approach yields good results. Additionally, it is important to note that there are other (more complicated) approaches in the literature for binarization of metaheuristics originally designed to solve continuous nonlinear optimization problems (Lanza-Gutierrez, 2016). However, Vasko and Lu (2017) reported that the simple approach outlined above performed the best for the set covering problem.

DATA AND APPLICATION

Real-world Dataset – Crime

Background

This dataset is generated from Communities and Crime Unnormalized Data Set on UCI Machine Learning Repository. (Redmond, 2011)

The original dataset combines socio-economic data from the '90 Census, law enforcement data from the 1990 Law Enforcement Management and Admin Stats survey, and crime data from the 1995 FBI UCR. This dataset includes 2215 cases and 147 variables, but the 'crime dataset' used in this section consists of 760 randomly selected cases (communities) with the population size between around 14,000 and 43,000 and 31 variables. One variable (the number of burglaries) is used as the response variable and the 30 remaining variables, including per capita income and median gross rent, are used as explanatory variables for a linear regression model.

Among the 760 cases, 380 randomly selected cases are used for the training set to fit the model and the remaining 380 cases are used for the validation set to evaluate the model selected from the training set. These two sets are used for analysis and comparisons in the next section.

Analysis and Result

In this section, a multiple linear regression model is used to find a relationship between the response variable and the explanatory variables described in the previous section. The programming language C++ was used for analysis on the computer with Windows 10 Pro edition (64 bit) and Intel core i5-6300U.

The weighted average of the Akaike Information Criterion (AIC) (Akaike, 1974) is used as the (*ad hoc*) objective function for optimization:

$$AIC_w = pAIC_t + \left(1 - p\right) AIC_v \tag{2}$$

where *p* and *1-p* are the proportions of the training set and the validation set from the whole dataset, respectively.

AIC is formulated as follows:

$$AIC = 2k + 2ln\left(\hat{L}\right) \tag{3}$$

where *k* is the number of parameters and $\hat{L}$ is the maximum value of the likelihood function for the model. In the crime dataset, *p*=0.5 because the training set and the validation set have the same size of 380. AIC_t and AIC_v are the AICs calculated from the training set and the validation set, for each.

AIC_t is used to estimate the coefficients in multiple regression with the training set and then AIC_v is used to evaluate a model derived in the previous step with the cases in the validation set.

Now, it is explained briefly how to conduct variable selection process (for getting relevant variables and the corresponding coefficients) step by step:

Step 1: Generate a population of a fixed size of bit strings for a given metaheuristic method.

Step 2: With a selected bit string from the population in step 1, use the data points in the training set, estimate coefficients and calculate the corresponding AIC_t.

Step 3: Use the data points in the validation set and the coefficients (or model) from step 2, calculate the corresponding AIC_v and then calculate the AIC_w.

Step 4: Repeat steps 2-3 as needed and update population as desired until the stopping criteria are satisfied (either the limit of 3600 second of running time or no observed improvement in terms of AIC_w, whichever comes first.)

Its flowchart is listed in Figure 1 and its detailed pseudocode can be found in the Appendix.

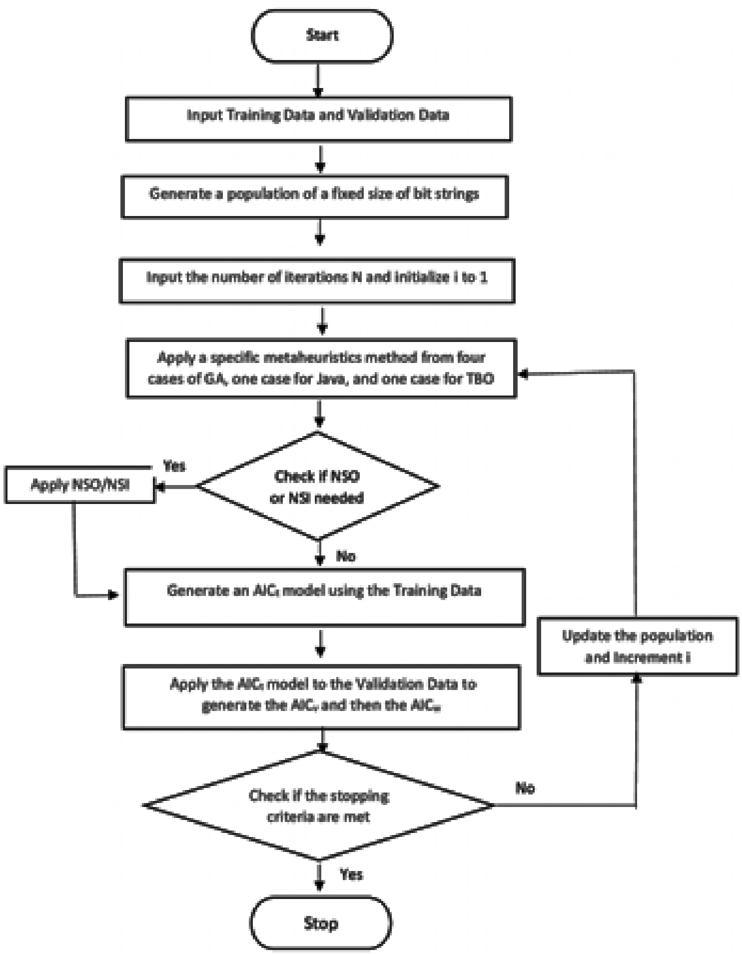
Basic Comparison of Three Metaheuristics

GA, TBO, and Jaya are used as our main metaheuristics in this section, but specific procedures for each method are not given in the steps above due to their complexity. One can easily check them in the references mentioned in section 1 for GAs, if needed. For TBO and Jaya, one can check section 2.

These three methods are used for analysis and compared with one another for their performance and efficiency in terms of the magnitude of objective function (AIC_w) and running time, for each. Four cases for GA and one case for TBO and Jaya, respectively, are used as described:

Case 1: GA with random selection of parents.
Case 2: GA with random selection of parents plus mutation.

Figure 1. Flowchart



Case 3: GA with crossover.
Case 4: GA with crossover plus mutation.
Case 5: Jaya.
Case 6: TBO.

Each case used 5 lists with the population size of 300 for calculation.

Table 1 shows the summary from the 5th list. (other lists show similar results.) The five cases (Case 1 to Case 5) have the same 'Best AICw' with the global optimum (minimum) at 3552.72, which means the five cases successfully attain the best possible AICw, so they have the same 'Explanatory variables selected'. The Exhaustive method, which checks all combinations of the variables, confirms that the five cases achieve the best possible variable selection even if they are not shown in Table 1. All GAs (Cases 1 to 4) and Jaya (Case 5) attained the global optimum, but TBO did not. Jaya was much faster than GAs (16.07 vs. 36.22 or 64.04 or 63.32 or 103.32) and TBO was fast (19.50) comparing with GAs but wound up with a sub-optimum.

More Comparisons With Different Jaya Population Sizes

Based on the results above, Jaya looks superior to other metaheuristic methods. In this section, it will be explored how Jaya can be improved with controlling population size, which is one of only two parameters (population size and stopping criteria) used in Jaya. Four scenarios are defined as J1, J2, J3, and J4 with the different population size of 300, 200, 100, and 50, respectively. Each scenario used five lists with the same population size for calculation.

Table 2 shows the results from the four scenarios. J1 and J2 detected the optimum 2 times and 3 times, respectively, among 5 trials (lists). Neither J3 nor J4 detected the optimum. - J3 and J4 arrived at sub-optima of 3556.73 and 3557.62, respectively, as best results. J2 (population size of 200) showed the best performance in terms of quality (highest number of detecting) and speed (11.71).

Some may argue if the results are reliable because of the limited number of trials or lists, but it must be noted that the main interest of this section is not to calculate a 'success' probability (probability of getting the global optimum) based on a large number of trials but to check whether or not each scenario can achieve the 'goal' (detecting the global optimum in reasonably a smaller number of trials.)

It provides a clue that Jaya can be improved by a certain amount of population size reduction. It also suggests a trade-off between the quality of results and the size of population. If a small-sized population is used, running time will be reduced at the cost of more likelihood of getting sub-optima.

Table 1. Comparison of metaheuristic methods I – Crime.

Case	Best AIC_w [1]	Explanatory variables selected		Time[3]
		Number	Index[2]	
1	3555.72	13	1,2,8,10,12,16,23,24,25,26,27,28,29	36.22
2	3555.72	13	1,2,8,10,12,16,23,24,25,26,27,28,29	64.05
3	3555.72	13	1,2,8,10,12,16,23,24,25,26,27,28,29	63.32
4	3555.72	13	1,2,8,10,12,16,23,24,25,26,27,28,29	103.32
5	3555.72	13	1,2,8,10,12,16,23,24,25,26,27,28,29	16.07
6	3558.65	12	1,2,8,9,10,12,24,25,26,27,28,29	19.50

[1]Best (smallest) AIC_w from 5 lists in each case.
[2]If the index*i* is shown, it implies the i^{th} explanatory variable is selected. (i=1,···, 30)
[3]The unit of time is minute.

Table 2. Comparison of Jaya with different population sizes – Crime

Scenario	Number of lists detecting the optimum[1]	Time[2]
J1	2	14.10
J2	3	11.71
J3	0	13.44
J4	0	2.91

[1]Number of trials in which the minimum AIC_w is detected among 5 lists.
[2]The unit of time is minute.

Improving Jaya With Local or Neighborhood Search

Even if the running time of Jaya may be reduced by using a smaller population, it may lower the quality of the results. Is there any way to have reliable result still with a small size of population?

Local or neighborhood search is any procedure that perturbs a given solution in order to try to improve it. There are entire books written on local search procedures (Hoos and Stutzle, 2005) and it refers to a strategy—not a particular algorithm. In this paper, neighborhood search refers to modifying a solution by randomly selecting one variable and changing its value—from one to zero or zero to one. If the objective function is improved, this becomes the incumbent solution, and the process is repeated until there is no improvement in a set number of trials.

In this paper, the neighborhood search is used in two ways. First, it is used "inside" Jaya to try to improve the best solution found during the execution of Jaya. Second, it is used "outside" Jaya—the best solution found by Jaya has the neighborhood search performed on it.

In Table 3, "outside" and "inside" are referred to as NSO and NSI, for each. The detail of how they work in the algorithm is described:

1. **NSO (Neighborhood Search Outside):** It runs Jaya a minimum of 5 loops until no improvement, then it runs neighborhood search on the best value until no improvements after 10 loops. It switches back to conducting Jaya until 5 operations of no improvements, then do neighborhood search for the 10 operations of no improvement. It keeps switching between operating 5 Jayas and 10 neighborhood searches until best value does not improve after 5 loops.
2. **NSI (Neighborhood Search Inside):** It runs Jaya once and then runs neighborhood search on the best value until no improvements after 10 loops. It switches back to running Jaya once and then do neighborhood search for 10 operations of no improvements. It continues doing more

Table 3. Comparison of Jaya with neighborhood search

Population	Methods of Neighborhood Search	Number of lists detecting the optimum[1]	Time[2]
50	NSO	0	0.04
	NSI	0	2.52
100	NSO	0	0.03
	NSI	2	3.45
150	NSO	1	0.04
	NSI	2	5.53

[1]Number of trials in which the minimum AIC_w is detected among 5 lists
[2]The unit of time is minute.

loops of one Jaya followed by 10 neighborhood searches until the best value does not improve after a total of 5 loops.

Table 3 summarizes the results of the two neighborhood searches with different population sizes. When the population size is 150, both NSO and NSI detected the optimum once and twice, for each. NSI detected the optimum two times, but NSI did not when the size of population is 100. Neither NSO nor NSI detected the optimum when the population size is very small (=50).

NSO was much faster than NSI in all cases, but it detected the minimum AIC_w once when the population size is 150. NSI was slower than NSO in each case but successfully detected the minimum AIC_w consistently with a high likelihood (2 out of 5) except the case of the populations size of 50. The results suggest that it will be better using NSI if the population size is relatively small and NSO if the population size is relatively big.

It must be noted that both NSO and NSI demonstrated remarkable improvement from the results with the initial population size of 300 (the scenario J1 in Table 2), in terms of performance and running time when local or neighborhood search was adapted.

Simulated Dataset

In this section, a simple simulation is conducted to check the sensitivity of the change of initial population size and the ability to detect the 'right' relationship among variables when Jaya is run.

A hypothetical dataset with 500 cases and 41 variables (40 explanatory variables ($x_1, \cdots x_{40}$) and one response variable (y) was generated. All variables are randomly generated from the standard normal distribution except the first explanatory variable x_1 which has the following relationship with others:

$$x_1 = y - 2\left(x_2 + x_3 + x_4 + x_5\right) \tag{4}$$

In other words, the 'answer' relationship between the explanatory variables and the response variable is:

$$\hat{y} = x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 \tag{5}$$

This dataset will be considered as a 'training' set and the Bayesian Information Criterion (BIC) (Schwarz, 1978) will be used as the objective function for optimization in this section. The term BIC_t will be used to be consistent with notations in section 3.1.2.

In Tables 4 and 5, four instances are defined as S1, S2, S3, and S4 with the different population size of 50, 100, 200, and 300, respectively. Each instance uses five lists with the same population size for calculation. The running mechanism of Jaya is the same as section 3.1.2.

Table 4 shows similar results from Table 2. – The bigger population size, the higher chance to detect the right model. It shows that even with smaller size of population such as 50, it is likely to detect the global optimum.

Table 5 illustrates many aspects of the analysis. The best BIC_t are ranged from -19855.90 to -19874.20 and the running time are ranged from 5.05 to 35.38.

All the lists in S1 to S4 detected the intended explanatory variables (x_1, x_2, x_3, x_4, and x_5), which implies that in many situations Jaya can detect a close-to-optimized solution at least. But many of them also include 'noise' or redundant explanatory variables. For example, the list 1 in S1 selected 9 variables among which 4 variables are wrongfully selected. It can be observed that S1, S2, S3, and S4 selected the right model 1 time, 0 time, 3 times and 4 times from the corresponding 5 lists, for each. It reinforces the finding in section 3.1.2.- the larger population, the higher likelihood to get the

Table 4. Jaya with different population size (summary) – Simulation

Instance	Number of lists detecting the optimum[1]	Average Time[2]
S1	1	5.86
S2	0	10.65
S3	3	14.92
S4	4	17.86

[1]Number of lists in which the minimum BIC_t is detected among 5 lists.
[2]The average running time in minute.

Table 5. Jaya with different population size (detailed) – Simulation

Population (Instance)	List	Best BIC_t[1]	Explanatory variables selected			Time[5]
			Index[2]	No. All[3]	No. Error[4]	
50 (S1)	1	-19855.90	1,2,3,4,5,12,28,32,38	9	4	8.05
	2	-19874.20	1,2,3,4,5	5	0	6.60
	3	-19854.50	1,2,3,4,5,6,7,14,35	9	4	5.05
	4	-19862.90	1,2,3,4,5,32,35	7	2	4.78
	5	-19868.10	1,2,3,4,5,28	6	1	4.83
100 (S2)	1	-19863.70	1,2,3,4,5,31,32	7	2	14.10
	2	-19871.90	1,2,3,4,5,9	6	1	9.63
	3	-19860.80	1,2,3,4,5,12,31,39	8	3	8.45
	4	-19867.30	1,2,3,4,5,12,38	7	2	11.53
	5	-19872.00	1,2,3,4,5,12	6	1	9.52
200 (S3)	1	-19871.90	1,2,3,4,5,9	6	1	23.88
	2	-19869.80	1,2,3,4,5,38	6	1	16.48
	3	-19874.20	1,2,3,4,5	5	0	13.30
	4	-19869.30	1,2,3,4,5,20	5	0	11.10
	5	-19874.20	1,2,3,4,5	5	0	9.83
300 (S4)	1	-19862.00	1,2,3,4,5,9,35,38	8	3	35.38
	2	-19874.20	1,2,3,4,5	5	0	23.07
	3	-19874.20	1,2,3,4,5	5	0	10.27
	4	-19874.20	1,2,3,4,5	5	0	8.53
	5	-19874.20	1,2,3,4,5	5	0	12.07

[1]Best (smallest) BIC_t in each list.
[2]If the index i is shown, it implies the i^{th} explanatory variable is selected. (i=1,⋯, 40)
[3]Number of all selected explanatory variables.
[4]Number of falsely selected explanatory variables.
[5]The unit of time is minute.

optimum. It can be observed that the number of 'noise' variables tends to decrease as the populations size increases. Also, it must be noted that S3 can compete S4 in terms of quality of the results with a considerably reduced size of population (200 vs. 300).

CONCLUSION

In this paper, variable selection, one of the classical topics in regression, was dealt with using metaheuristic methods. It can be stated as a combinatorial optimization problem with the goal to select variables that maximize (or minimize) the given objective function. Even if some metaheuristics such as Genetic Algorithm (GA) or Simulated Annealing (SA) have shown better performance in many problems over conventional methods, it turned out that 'fine tuning of parameters' is very challenging.

This paper explored some 'parameter-free' metaheuristics like Teaching-Based Optimization (TBO) and Jaya and compared them to GA. The previous sections illustrated that Jaya is superior to other metaheuristic methods in terms of performance and efficiency when it is properly used with relatively small population and neighborhood search.

It must be noted that one of the main purposes of this paper is not to develop a complete package to solve many different problems but to suggest how parameter-free metaheuristics such as Jaya can be used for variable selection.

Also, it must be admitted that the algorithms used for the datasets serve as an initial trial for the development of better parameter-free metaheuristic algorithms to come. Even if some simulations in high dimensional, say 100, space were conducted, their results were not included in the paper, due to issues arising from complexity and too much 'noise', which implies that there is a lot of room for improvement.

Lastly, a possible direction for future research may include, but is not limited to, handling highly correlated variables, and developing stronger computing methods to manage 'the curse of dimensionality' to some extent.

FUNDING BODY

The publisher has waived the Open Access Processing fee for this article.

REFERENCES

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *19*(6), 716–723. doi:10.1109/TAC.1974.1100705

Akaike, H. (1978). A Bayesian analysis of the minimum AIC procedure. *Annals of the Institute of Statistical Mathematics*, *30*(1), 9–14. doi:10.1007/BF02480194

Broadhurst, D., Goodacre, R., Jones, A., Rowland, J. J., & Kell, D. B. (1997). Genetic algorithms as a method for variable selection in multiple linear regression and partial least squares regression, with applications to pyrolysis mass spectrometry. *Analytica Chimica Acta*, *348*(1–3), 71–86. doi:10.1016/S0003-2670(97)00065-2

Delahaye, D., Chaimatanan, S., & Mongeau, M. (2018). *Simulated Annealing: From Basics to Applications. Handbook of Metaheuristics*. doi:10.1007/978-3-319-91086-4

Desboulets, L. (2018). A Review on Variable Selection in Regression Analysis. *Econometrics*, *6*(4), 45. doi:10.3390/econometrics6040045

Fan, J., & Li, R. (2001). Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. *Journal of the American Statistical Association*, *96*(456), 1348–1360. doi:10.1198/016214501753382273

Hans, C., Dobra, A., & West, M. (2012). Shotgun Stochastic Search for "Large p" Regression. *Journal of the American Statistical Association*, *102*(478), 507–516. doi:10.1198/016214507000000121

Henderson, D., Jacobson, S. H., & Johnson, A. W. (2003). The Theory and Practice of Simulated Annealing. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics. International Series in Operations Research & Management Science* (Vol. 57). Springer. doi:10.1007/0-306-48056-5_10

Hill, R., & Pohl, E. (2019). A structural taxonomy for metaheuristic optimization search methods. *International Journal of Metaheuristics.*, *7*(2), 127–151. doi:10.1504/IJMHEUR.2019.098261

Hocking, R. (1976). The Analysis and Selection of Variables in Linear Regression. *Biometrics*, *32*(1), 1–49. doi:10.2307/2529336

Hoos, H. H., & Stutzle, T. (2005). *Stochastic Local Search Foundations and Applications* (1st ed.). Morgan Kauffman.

Jirapech-Umpai, T., & Aitken, S. (2005). Feature selection and classification for microarray data analysis: Evolutionary methods for identifying predictive genes. *BMC Bioinformatics*, *6*(1), 148. doi:10.1186/1471-2105-6-148 PMID:15958165

Kapetanios, G. (2007). Variable selection in regression models using nonstandard optimization of information criteria. *Computational Statistics & Data Analysis*, *52*(1), 4–15. doi:10.1016/j.csda.2007.04.006

Kiezun, A., Lee, I.-T. A., & Shomron, N. (2009). Evaluation of optimization techniques for variable selection in logistic regression applied to diagnosis of myocardial infarction. *Bioinformation*, *3*(7), 311–313. doi:10.6026/97320630003311 PMID:19293999

Lanza-Gutierrez, J. M., Crawford, B., Soto, R., Berrios, N., Gomez-Pulido, J. A., & Paredes, F. (2017). Analyzing the effects of binarization techniques when solving the set covering problem through swarm optimization. *Expert Systems with Applications*, *70*, 67–82. doi:10.1016/j.eswa.2016.10.054

Lindsey, C., & Sheather, S. (2010). Variable Selection in Linear Regression. *The Stata Journal: Promoting Communications on Statistics and Stata*, *10*(4), 650–669. doi:10.1177/1536867X1101000407

Lu, Y., & Vasko, F. J. (2015). An OR Practitioner's Solution Approach for the Set Covering Problem. *International Journal of Applied Metaheuristic Computing*, *6*(4), 1–13. doi:10.4018/IJAMC.2015100101

Meiri, R., & Zahavi, J. (2006). Using simulated annealing to optimize the feature selection problem in marketing applications. *European Journal of Operational Research*, *171*(3), 842–858. doi:10.1016/j.ejor.2004.09.010

Mitchell, M. (2016). *An Introduction to Genetic Algorithms.* The MIT Press.

Mohan, S., Buchanan, B. R., Wollenberg, G. D., Igne, B., Drennen, J. K. III, & Anderson, C. A. (2018). Variable selection optimization for multivariate models with Polar Qualification System. *Chemometrics and Intelligent Laboratory Systems*, *180*, 1–14. doi:10.1016/j.chemolab.2018.06.002

Paterini, S., & Minerva, T. (2010). Regression Model Selection Using Genetic Algorithms. In Recent Advances in neural networks, fuzzy systems & evolutionary computing. Iasi, Romania: Wseas.us.

Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(8), 1226–1238. doi:10.1109/TPAMI.2005.159 PMID:16119262

Rao, R., & Kalyankar, V. (2011). *Parameters optimization of advanced machining processes using TLBO algorithm*. http://www.ppml.url.tw/EPPM/conferences/2011/download/SESSION1/21_32.pdf

Redmond, M. (2011). *Communities and Crime Unnormalized Data Set*. UCI Machine Learning Repository. Uci. Edu. http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime+Unnormalized

Schwarz, G. (1978). Estimating the Dimension of a Model. *Annals of Statistics*, *6*(2), 461–464. doi:10.1214/aos/1176344136

Sinha, A., Malo, P., & Kuosmanen, T. (2015). A Multiobjective Exploratory Procedure for Regression Model Selection. *Journal of Computational and Graphical Statistics*, *24*(1), 154–182. doi:10.1080/10618600.2014.899236

Vasko, F. J., & Lu, Y., Y. (2017). Binarization of Continuous Metaheuristics to Solve the Set Covering Problem: Simpler is Better. *21st Triennial Conference of The International Federation of Operational Research Societies (IFORS)*.

Venkata Rao, R. (2016). Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, 19–34. 10.5267/j.ijiec.2015.8.004

APPENDIX

What is the Genetic Algorithm?

The genetic algorithms (GAs) are evolutionary and search-based optimization procedures based on the principles of genetics and natural selection, inspired by Darwin's theory of evolution. It is often used to find optimal or sub-optimal solutions to difficult problems such as NP-hard problems.

Even if there is no rigorous definition of "genetic algorithm" accepted by all in the evolutionary−computation community that differentiates GAs from other evolutionary computation methods, it can be said that most methods called "GAs" have at least the following elements in common: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring. (Mitchell, 2016)

GAs are methods for moving from one population of "chromosomes" (strings of ones and zeros, or "bits") to a new population by using a kind of "natural selection" together with the genetics−inspired operators of crossover, and mutation. Each chromosome consists of "genes" (bits), each gene being an instance of a particular "allele" (0 or 1). The selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more offspring than the less fit ones. Crossover exchanges subparts of two chromosomes, roughly mimicking biological recombination between two single−chromosome (haploid) organisms. Mutation randomly changes the allele values of some locations in the chromosome. (Mitchell, 2016)

What is the Simulated Annealing?

The simulated annealing (SA) is a metaheuristic local search algorithm which can escape from local optima. Its ease of implementation, convergence properties and its use of hill-climbing moves to escape local optima have made it a popular technique over the past two decades. It is typically used to address discrete, and to a lesser extent, continuous optimization problems. The main advantage of SA is its simplicity. SA avoids the drawback of the Monte-Carlo approach (which can be trapped in local minima), thanks to an efficient Metropolis acceptance criterion. (Delahaye et al., 2018)

Simulated annealing is so named because of its analogy to the process of physical annealing with solids, in which a crystalline solid is heated and then allowed to cool very slowly until it achieves its most regular possible crystal lattice configuration (i.e., its minimum lattice energy state), and thus is free of crystal defects. If the cooling schedule is sufficiently slow, the final configuration results in a solid with such superior structural integrity. Simulated annealing establishes the connection between this type of thermodynamic behavior and the search for global minima for a discrete optimization problem. Furthermore, it provides an algorithmic means for exploiting such a connection. (Henderson et al., 2003)

At each iteration of a simulated annealing algorithm applied to a discrete optimization problem, the objective function generates values for two solutions (the current solution and a newly selected solution) are compared. Improving solutions are always accepted, while a fraction of non-improving (inferior) solutions are accepted in the hope of escaping local optima in search of global optima. The probability of accepting non-improving solutions depends on a temperature parameter, which is typically non-increasing with each iteration of the algorithm. (Henderson et al., 2003)

Pseudocode

```
GET TrainingDataSetFile
GET ValidationDataSetFile

SET VariableSize FROM TrainingDataSetFile

INPUT ListSize, maxIterations, maxTries, fitnessValueType

SET Lists AS Array OF 5 list
FOR each Lists
     INPUT ListPath
     IF ListPath exist
          GET list FROM ListPath
     ELSE
          CALL Generate
     END IF
END FOR

INPUT MethodUsing

FOR each Lists
     IF MethodUsing IS Generate
          CALL Generate
     ELSE IF MethodUsing IS Random
          CALL Random
     ELSE IF MethodUsing IS Random_Mutate
          CALL Random_Mutate
     ELSE IF MethodUsing IS Crossover
          CALL Crossover
     ELSE IF MethodUsing IS Crossover_Mutate
          CALL Crossover_Mutate
     ELSE IF MethodUsing IS NeighborhoodSearch
          CALL NeighborhoodSearch
     ELSE IF MethodUsing IS Jaya
          CALL Jaya
     ELSE IF MethodUsing IS NeighborhoodSearch_Inside_Jaya
          CALL NeighborhoodSearch_Inside_Jaya
     ELSE IF MethodUsing IS Jaya_Then_NeighborhoodSearch
          CALL Jaya_Then_NeighborhoodSearch
     ELSE IF MethodUsing IS TBO
          CALL TBO
     END IF
     Save list
END FOR

Save reportFile
Exit Program

FUNCTION Generate
     SET currentAmount TO 0
```

```
    WHILE currentAmount TO ListSize
        FOR 0 TO VariableSize
            SET bit IN variable TO 0 OR 1 randomly.
        END FOR
        CALL CalculateFitnessValue
        IF FIND variable IN list
            CONTINUE LOOP
        ELSE
            ADD variable TO list
            INCREMENT currentAmount
        END IF
    END WHILE
END FUNCTION

FUNCTION Random
    SET tries TO 0
    WHILE tries IS less than maxTries AND less than 2 hours THEN
        SET P1 AND P2 TO random selected variables IN list
        FOR each bit IN Child
            IF P1 bit equal P2 bit THEN
                SET bit TO P1
            ELSE
                SET bit TO 0 OR 1 randomly
            END IF
        END FOR
        CALL CalculateFitnessValue
        IF Child exists IN List OR FitnessValue IS less than middle FitnessValue IN
          List
            INCREMENT tries
        ELSE
            SET tries TO 0
            SET r TO random value between ListSize AND (ListSize / 2)
            SET variable AT r IN list TO Child
        END IF
    END WHILE
END FUNCTION

FUNCTION Random_Mutate
    SET Child FROM CALL Random
    SET ChildMutate TO Child
    SET tries TO 0
    WHILE tries IS less than maxTries AND less than 2 hours THEN
        SET rbit TO 0 TO VariableSize randomly
        IF ChildMutate bit AT rbit IS 0 THEN
            SET ChildMutate bit AT rbit TO 1
        ELSE
            SET ChildMutate bit AT rbit TO 0
        END IF
        CALL CalculateFitnessValue
        IF ChildMutate exists IN List OR FitnessValue IS less than middle
```

```
        FitnessValue IN List
            INCREMENT tries
        ELSE
            SET tries TO 0
            SET r TO random value between ListSize AND (ListSize / 2)
            SET variable AT r IN list TO ChildMutate
        END IF
    END WHILE
END FUNCTION

FUNCTION Crossover
    SET tries TO 0
    WHILE tries IS less than maxTries AND less than 2 hours THEN
        SET P1 AND P2 TO random selected variables IN list
        SET splitHalf TO random value between 1 AND (VariableSize - 1)
        SET currentBit TO 0
        WHILE currentBit less than VariableSize
            IF currentBit less than splitHalf THEN
                SET child1 bit TO P1
                SET child2 bit TO P2
            ELSE
                SET child1 bit TO P2
                SET child2 bit TO P1
            END IF
            INCREMENT currentBit
        END WHILE
        CALL CalculateFitnessValue OF child1
        IF child1 exists IN List OR FitnessValue IS less than middle FitnessValue IN
          List
            INCREMENT tries
        ELSE
            SET tries TO 0
            SET r TO random value between ListSize AND (ListSize / 2)
            SET variable AT r IN list TO child1
        END IF
        CALL CalculateFitnessValue OF child2
        IF child2 exists IN List OR FitnessValue IS less than middle FitnessValue IN
          List
            INCREMENT tries
        ELSE
            SET tries TO 0
            SET r TO random value between ListSize AND (ListSize / 2)
            SET variable AT r IN list TO child2
        END IF
    END WHILE
END FUNCTION

FUNCTION Crossover_Mutate
    SET Child1 AND Child2 FROM CALL Crossover
    SET Child1Mutate TO Child1
```

```
    SET Child2Mutate TO Child2
    SET tries TO 0
    WHILE tries IS less than maxTries AND less than 2 hours THEN
         SET rbit1 TO 0 TO VariableSize randomly
         IF Child1Mutate bit AT rbit1 IS 0 THEN
              SET Child1Mutate bit AT rbit1 TO 1
         ELSE
              SET Child1Mutate bit AT rbit1 TO 0
         END IF
         SET rbit2 TO 0 TO VariableSize randomly
         IF Child2Mutate bit AT rbit2 IS 0 THEN
              SET Child2Mutate bit AT rbit2 TO 1
         ELSE
              SET Child2Mutate bit AT rbit2 TO 0
         END IF
         CALL CalculateFitnessValue OF Child1Mutate
         IF Child1Mutate exists IN List OR FitnessValue IS less than middle
           FitnessValue IN List
              INCREMENT tries
         ELSE
              SET tries TO 0
              SET r TO random value between ListSize AND (ListSize / 2)
              SET variable AT r IN list TO Child1Mutate
         END IF
         CALL CalculateFitnessValue OF Child2Mutate
         IF Child2Mutate exists IN List OR FitnessValue IS less than middle
           FitnessValue IN List
              INCREMENT tries
         ELSE
              SET tries TO 0
              SET r TO random value between ListSize AND (ListSize / 2)
              SET variable AT r IN list TO Child2Mutate
         END IF
    END WHILE
END FUNCTION

FUNCTION NeighborhoodSearch
    SET iteration TO 0
    WHILE iteration IS less than maxIterations AND less than 2 hours THEN
         SET update TO variable AT list OF index 0
         SET rbit TO random value between 0 AND VariableSize
         IF update bit AT index rbit equal 0 THEN
              SET update bit AT rbit TO 1
         ELSE
              SET update bit AT rbit TO 0
         END IF
         CALL CalculateFitnessValue OF update
         IF update value less than variable value AT list OF index 0 THEN
              SET variable AT list OF index 0 TO update
              SET iteration TO 0
```

```
          ELSE
               INCREMENT iteration
          END IF
     END WHILE
END FUNCTION

FUNCTION Jaya
     SET iteration TO 0
     WHILE iteration IS less than maxIterations AND less than 2 hours THEN
          SET pastTop TO variable AT list OF index 0
          SET bottom TO variable AT list OF index (listSize - 1)
          FOR each variable IN list
               SET update TO variable
               FOR each bit IN update
                    SET r1 TO random value between 0 AND 1
                    SET r2 TO random value between 0 AND 1
                    UPDATE bit TO bit + (r1 * (pastTop - bit)) - (r2 * (bottom-bit))
               END FOR
               CALL CalculateFitnessValue OF update
               IF update value less than variable value THEN
                    SET variable TO update
               END IF
          END FOR
          IF variable AT list OF index 0 value less than pastTop value THEN
               SET iteration TO 0
          ELSE
               INCREMENT iteration
          END IF
     END WHILE
END FUNCTION

FUNCTION NeighborhoodSearch_Inside_Jaya
     SET iteration TO 0
     WHILE iteration IS less than maxIterations AND less than 2 hours THEN
          SET pastTop TO variable AT list OF index 0
          SET bottom TO variable AT list OF index (listSize - 1)
          FOR each variable IN list
               SET update TO variable
               FOR each bit IN update
                    SET r1 TO random value between 0 AND 1
                    SET r2 TO random value between 0 AND 1
                    UPDATE bit TO bit + (r1 * (pastTop - bit)) - (r2*(bottom-bit))
               END FOR
               CALL CalculateFitnessValue OF update
               IF update value less than variable value THEN
                    SET variable TO update
               END IF
          END FOR
          IF variable AT list OF index 0 value less than pastTop value THEN
               SET iteration TO 0
```

```
          ELSE
               INCREMENT iteration
          END IF
     END WHILE
END FUNCTION

FUNCTION Jaya_Then_NeighborhoodSearch
     SET iteration TO 0
     WHILE iteration IS less than maxIterations AND less than 2 hours THEN
          SET pastTop TO variable AT list OF index 0
          CALL Jaya
          CALL NeighborhoodSearch
          SET newTop TO variable AT list OF index 0
          IF newTop value less than pastTop value THEN
               SET iteration TO 0
          ELSE
               INCREMENT iteration
          END IF
     END WHILE
END FUNCTION

FUNCTION TBO
     SET iteration TO 0
     WHILE iteration IS less than maxIterations AND less than 2 hours THEN
          SET pastTop TO variable AT list OF index 0
          SET middle TO variable AT list OF index (listSize / 2)
          FOR each variable IN list
               SET update TO variable
               FOR each bit IN update
                    SET r TO random value between 0 AND 1
                    SET Tf TO random value between 1 AND 2
                    UPDATE bit TO bit + (r * (pastTop - (Tf * middle)))
               END FOR
               CALL CalculateFitnessValue OF update
               IF update value less than variable value THEN
                    SET variable TO update
               END IF
          END FOR
          IF variable AT list OF index 0 value less than pastTop value THEN
               SET iteration TO 0
          ELSE
               INCREMENT iteration
          END IF
     END WHILE
END FUNCTION

FUNCTION CalculateFitnessValue
     SET m FROM TrainingDataSetFile
     SET Y_training FROM TrainingDataSetFile
     SET Y_validation FROM ValidationDataSetFile
```

```
SET n TO 0
FOR each bit IN variable
     IF bit equal 1 THEN
          INCREMENT n
END FOR

INITIALIZE X_training AS matrix OF (m AND (n + 1))
INITIALIZE X_validation AS matrix OF (m AND (n + 1))

SET col TO 0
SET currentColumn TO 1
FOR col less than VariableSize
     IF bit AT col IS 1 THEN
          FOR each rows IN TrainingDataSetFile
               SET X_training AT (row AND currentColumn) TO
                    TrainingDataSetFile AT (row AND col)
               SET X_validation AT (row AND currentColumn) TO
                    ValidationDataSetFile AT (row AND col)
          END FOR
          INCREMENT currentColumn
     END IF
END FOR
FOR each rows IN TrainingDataSetFile
     SET X_training AT (row AND 0) TO 1
     SET X_validation AT (row AND 0) TO 1
END FOR

SET X_transpose TO (transpose OF X_training)
SET X_transTimesX TO (X_transpose * X_training)
SET X_inverse TO (inverse OF X_transTimesX)
SET X_invTimesTrans TO (X_inverse * X_transpose)
SET O TO (X_invTimesTrans * Y_training)

INITIALIZE h_training AS matrix OF (m AND 1)
FOR each rows IN TrainingDataSetFile
     SET h TO 0
     SET col TO 0
     WHILE col less than (n + 1)
          SET h TO h + ((O AT (col AND 0)) * (X_training AT (row AND col)))
          INCREMENT col
     END WHILE
     SET h_training AT (row AND 0) TO h
END FOR

INITIALIZE h_validation AS matrix OF (m AND 1)
FOR each rows IN ValidationDataSetFile
     SET h TO 0
     SET col TO 0
     WHILE col less than (n + 1)
          SET h TO h + ((O AT (col AND 0)) * (X_validation AT (row AND col)))
```

```
            INCREMENT col
        END WHILE
        SET h_validation AT (row AND 0) TO h
    END FOR

    INITIALIZE RSS_training TO 0
    FOR each rows in TrainingDataSetFile
        SET RSS_training TO RSS_training +
            (POWER OF ((h_training AT (row AND 0) - (Y_training AT (row AND 0))) TO 2))
    END FOR

    INITIALIZE RSS_valiation TO 0
    FOR each rows in ValidationDataSetFile
        SET RSS_valiation TO RSS_valiation +
            (POWER OF ((h_validation AT (row AND 0) - (Y_validation AT (row AND 0))) TO 2))
    END FOR

    SET AIC_T TO (m * (log OF (RSS_training / m))) + (2 * (n + 1))
    SET AIC_V TO (m * (log OF (RSS_valiation / m))) + (2 * (n + 1))
    SET AIC_M TO (0.5 * AIC_T) + (0.5 * AIC_V)

    SET BIC_T TO (m * (log OF (RSS_training / m))) + ((log OF m) * (n + 1))
    SET BIC_V TO (m * (log OF (RSS_valiation / m))) + ((log OF m) * (n + 1))
    SET BIC_M TO (0.5 * BIC_T) + (0.5 * BIC_V)

    IF fitnessValueType IS AIC_Training THEN
        RETURN AIC_T
    ELSE IF fitnessValueType IS AIC_Validation THEN
        RETURN AIC_V
    ELSE IF fitnessValueType IS AIC_Middle THEN
        RETURN AIC_M
    ELSE IF fitnessValueType IS BIC_Training THEN
        RETURN BIC_T
    ELSE IF fitnessValueType IS BIC_Validation THEN
        RETURN BIC_V
    ELSE IF fitnessValueType IS BIC_Middle THEN
        RETURN BIC_M
    END IF
END FUNCTION
```

Myung Soon Song is currently an assistant professor of the department of Mathematics at Kutztown University of Pennsylvania. He received his Ph.D. and MA degrees in Statistics from University of Pittsburgh in 2011 and 2007, respectively. He also received his MS degree in Actuarial Science from University of Iowa in 2001 and his BS degree in Mathematics from Seoul National University in 1996. His research interests include, but not limited to, meta-analysis, multilevel modeling, and optimization.

Francis J. Vasko is a Professor Emeritus of Mathematics at Kutztown University of Pennsylvania. Before coming to Kutztown University in September 1986, he worked for more than eight years as an employee in the Research Department at Bethlehem Steel solving a variety of real-world applications in operations research. He then served as a consultant to Bethlehem Steel Corporation from September 1986 thru March 2003. Since 2003, he has also done consulting work for several companies. His current research focuses on using a large variety and combination of solution techniques (both math optimization-based and metaheuristics) in order to more accurately model and solve important real-world applications in production planning, strategic planning, and resource allocation.

Yun Lu is a Professor in the Department of Mathematics at Kutztown University. She received her M.A. degree in computer science and Ph.D. degree in mathematics from Wesleyan University in 2006 and 2007, respectively. Her research interests include optimization, algorithms, mathematical logic, and bioinformatics.

Kyle Callaghan is going for a master's degree in computer science. Kyle completed five years at Kutztown University where he also worked as a graduate assistant doing work discussed in this paper.