The Effect of Problem Reduction in the Integer Programming-based Local Search

Junha Hwang
Dept. of Computer Engineering, Kumoh National Institute of Technology
61 Daehak-ro, Gumi, Gyeongbuk, 39177, Korea
jhhwang@kumoh.ac.kr

***Abstract*—Integer Programming-based Local Search (IPbLS) is a kind of local search. IPbLS is based on the first-choice hill-climbing search and uses integer programming to generate a neighbor solution. IPbLS has been applied to solve various NP-hard combinatorial optimization problems like knapsack problem, set covering problem, set partitioning problem, and so on. In this paper, we investigate the effect of problem reduction in the IPbLS experimentally using the *n*-queens maximization problem. The characteristics of IPbLS are examined by comparing IPbLS using strong problem reduction with IPbLS using weak problem reduction, and also IPbLS is compared with other local search strategies like simulated annealing. Experimental results show the importance of problem reduction in IPbLS.**

Keyword-Integer Programming-based Local Search, Integer Programming, Local Search, Problem Reduction

I. INTRODUCTION

It is practically impossible to find an optimal solution in a reasonable amount of time for NP-hard combinatorial optimization problems because the search space increases exponentially as the problem size increases. Local search is a search technique to solve such large-scale optimization problems, where the current solution is improved by repeatedly finding a better solution from neighbor solutions of the current solution. Local search does not ensure finding an optimal solution, but it is known to be able to find sub-optimal solutions efficiently by generating only a very small part of the large search space.

The most important factor in the local search is how to define a neighbor solution. A candidate solution is represented as a set of decision variables in combinatorial optimization problems, and a neighbor solution is usually made by changing the values of some variables from the current solution. This method is often called *k*-exchange neighborhood[1]. Here, *k* means the maximum number of variables to be changed in order to generate a neighbor solution. As the value of *k* increases, better neighbor solutions are more likely to be generated. However, it takes too much time to generate all neighbor solutions and evaluate them because the number of neighbor solutions becomes too big. The value of *k* is therefore generally set to a small value like 1 or 2.

Another important factor in the local search is how to select the next current solution from neighbor solutions. There are two basic local search algorithms related to selecting the next current solution, which are steepest-ascent hill-climbing search and first-choice hill-climbing search[2]. Steepest-ascent hill-climbing search selects a best neighbor solution among every possible neighbor solutions. On the other hand, first-choice hill-climbing search accepts any solution that improves the current solution while generating a neighbor solution one by one. However, since hill-climbing search suffers from the local maxima problem, many other local search variants like tabu search and simulated annealing have been developed. But such search methods cannot perfectly overcome the local maxima problem as well because they are also basically based on *k*-exchange neighborhood.

Integer programming-based local search (IPbLS) is based on the first-choice hill-climbing search, and uses integer programming (IP) to generate a neighbor solution. IP can be applied when the given combinatorial optimization problem can be represented in linear form, and it is known to find an optimal solution very quickly based on branch and bound and simplex method. IPbLS adopting IP effectively searches better solutions with more neighbor solutions using bigger *k*-exchange neighborhood.

IPbLS repeatedly performs the following steps. First, *k* number of variables are selected from the current solution, and then the values of unselected variables are fixed to the same values in the current solution. Finally, the next current solution is made by determining the values of the rest variables with IP. At this time, fixing the values of unselected variables is viewed as problem reduction process for performing IP. We investigate the performance change of IPbLS according to problem reduction method. For this, two problem reduction methods are compared in the *n*-queens maximization problem (*n*-QMP). One is value fixing method which is used in general IPbLS, and the other is variable removing method using the current solution. In addition, characteristics of IPbLS are investigated through comparison with other existing local search techniques.

The structure of the paper is as follows. Section II describes the related works about IPbLS and *n*-QMP. Section III suggests IPbLS for *n*-QMP and section IV presents experimental results. Finally, in section V we draw the conclusion and discuss future work.

II. RELATED WORK

A. The Integer Programming-based Local Search

When a given combinatorial optimization problem can be represented in linear form it is called integer programming model, and integer programming (IP) is a search method to solve such problems. IP basically uses branch and bound, and related research has been actively conducted to improve the performance of IP itself[3]. The advantage of IP is that it is possible to find an optimal solution within a very short time in small or medium-sized problems. On the other hand, as the problem size grows, it may not possible to apply IP due to memory problem. Even if it is possible to apply, it can take a very long time until an optimal solution is found.

The integer programming-based local search (IPbLS) is a kind of search technique using IP to generate a neighbor solution in local search. Fig. 1 shows the basic algorithm of IPbLS[4]. A candidate solution X is assumed to be represented as a set of decision variables with a value 0 or 1, such as $x_1, x_2, \ldots, x_n$. First, IPbLS makes an initial solution using a heuristic method, and sets it to the current solution. Then, problem reduction and IP steps are repeatedly carried out to improve the current solution. The problem reduction step here can be viewed as the process for selecting variables to participate in the next IP. For this, k number of variables are selected from the current solution X, and these variables participate in the next IP. In other words, the search space is reduced for the next IP by fixing the values of unselected variables to the current values. Since the larger k results in an increase in the number of candidate solutions to be considered in the next IP, it is likely to take more time to find an optimal neighbor solution. But, the possibility to include better neighbor solutions is increased. On the other hand, the smaller the value of k, the quicker an optimal neighbor solution can be found but the poorer the quality of the optimal neighbor solution.

```
Algorithm IPbLS
    X : Variable vector (Current solution).
    k : The number of selected variables to be changed.
    IP : An integer programming solver.
Begin
    X = Make an initial solution using a heuristic method
    While stopping condition is not met Do
        Select k variables from X
        Add objective and all constraints to IP
            • Fix values of unselected variables
        X = Make a neighbor solution with IP
    End While
    Return X
End Begin
```

Fig.1. General Integer Programming-based Local Search

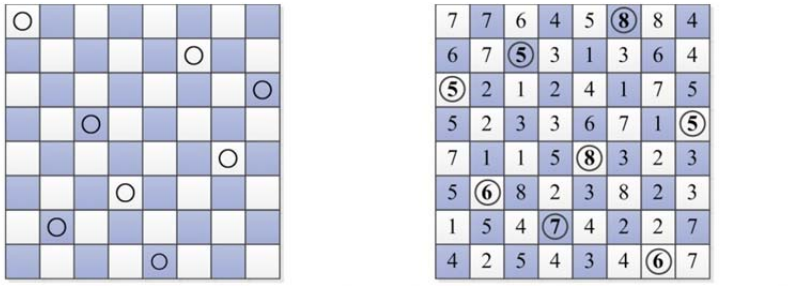
In general, local search algorithms modify one or two variables to make a neighbor solution, because the number of neighbor solutions to be considered is exponentially increased as the number of variables to be able to be modified is increased. This causes a local optima problem in most local search algorithms. However, IPbLS overcomes the local optima problem by modifying the values of more variables using the characteristic of IP.

IPbLS has been applied to a variety of combinatorial optimization problems such as the multidimensional knapsack problem, the set covering problem, the service network design problem, and so on[5, 6, 7]. And also, the performance of IPbLS was confirmed to be superior compared to other heuristic search techniques. In this paper, the effect of IPbLS related to the problem reduction which is a key point of IPbLS, is investigated by applying two problem reduction methods.

B. The n-queens Maximization Problem

In this research, the n-queens maximization problem is used to verify the performance according to the problem reduction method in IPbLS. The original n-queens problem is defined as placing n queens on an $n \times n$ board so that no two queens can threaten each other, i.e., no two queens can be placed on the same row, column or diagonal. Fig. 2(a) shows an example of 8-queens problem. The original n-queens problem is a kind of constraint satisfaction problem, but in this paper it was modified to a constraint optimization problem which is called the n-queens maximization problem (n-QMP). A specific value is given to each queen position on the board in n-QMP. The objective of n-QMP is to maximize the sum of the values of the positions where queens are placed. Therefore, the target problem can be defined as a constraint optimization problem maximizing the sum of the weights of the locations where queens are placed while satisfying the constraints of the n-queens problem. Fig. 2(b) shows an example of an optimal solution for an 8-queens maximization problem.

The original n-queens problem can be represented as following. A solution is represented by n decision variables, and each decision variable can have values from 0 to $(n-1)$. The value of each decision variable means the position of the queen in the corresponding column. This guarantees that only one queen can be placed on each column. And the first constraint is that the values of all decision variables are different, which guarantees that only one queen can be placed on each row. The second constraint is $(i-j) \neq (v_i - v_j)$ and $(i-j) \neq (v_j - v_i)$ when v_i is the value of the i-th decision variable and v_j is the value of the j-th decision variable. This guarantees that no two queens are placed on the same diagonal.



(a) a solution of the 8-queens problem (b) an optimal solution of an 8-queens maximization problem

Fig. 2. An Example of 8-queens Problem

The n-QMP has been used in [4] and [8]. In [4], the constraint programming was used as a neighbor generation method when applying local search to a constraint optimization problem. The existing research [8] proposed a genetic algorithm to solve a constraint optimization problem, but very small problems such as 30-queens problem were used as experimental data.

The n-QMP can be modeled as a linear constraint optimization problem using the expressions from (1) to (5). There are $n \times n$ decision variables. Let's assume that row and column indices start with 0 as shown in the example of Fig. 3.

$$\text{maximize} \quad \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} w_{ij}x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{j=0}^{n-1} x_{ij} = 1 \; for i = 0, \dots, n-1 \tag{2}$$

$$\sum_{i=0}^{n-1} x_{ij} = 1 \; for j = 0, \dots, n-1 \tag{3}$$

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} x_{ij} \leq 1 \; for i+j = 1, \dots, \big((n-1)+(n-1)-1\big) \tag{4}$$

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} x_{ij} \leq 1 \; for i-j = (-(n-2)), \dots, (n-2) \tag{5}$$

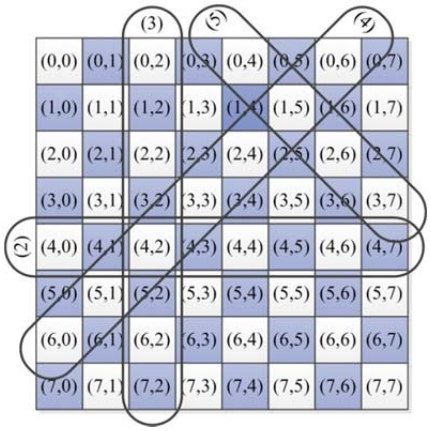


Fig. 3. A Linear Modeling of 8-queens Maximization Problem

The value of a decision variable x_{ij} becomes 1 if a queen is placed at i-th row and j-th column, and the value becomes 0 if no queen is placed at the position. And, w_{ij} denotes the weight assigned to the position of i-th row and j-th column. The expression (1) representing the objective function maximizes the sum of the weights of the positions where queens are placed. The expressions (2) and (3) represent the constraints that only one queen can be placed in any row and any column respectively as shown in Fig. 3(2) and Fig. 3(3). The expression (4) represents the constraint for the diagonal direction like Fig. 3(4). For example, when the value of $(i + j)$ is 6, only one queen is placed on the diagonal (4) in Fig.3 by the expression (4). Similarly, the expression (5) represents the constraint for the diagonal direction like Fig. 3(5).

III. IPbLS for the *n*-queens Maximization Problem

C. Overall Structure

Fig. 4 shows the overall structure of the integer programming-based local search (IPbLS) to solve the *n*-queens maximization problem (*n*-QMP). Basically, a solution of *n*-QMP can be also represented by a 1-dimensional array having *n* decision variables like a solution of the *n*-queens problem. However, 2-dimensional array having $n \times n$ decision variables is needed to solve *n*-QMP using the expression (1)~(5), and it is represented as *X'*. First, a solution for the *n*-queens problem is found using the proposed method in [9]. Second, *k* variables, namely, *k* columns are selected randomly. Next, *n*-QMP is reduced using the values of the unselected variables, and then the reduced problem is solved by integer programming (IP). Now a new solution is obtained, and the above process is repeatedly performed from the second step. The initial solution generation method and the problem reduction method will be more specifically explained in the next sections.

```
Algorithm IPbLS_for_n-queens_maximization_problem
    X : 1D-Array variable (Current solution).
    X' : 2D-Array variable for n-queens maximization problem.
Begin
    X = Make an initial solution using the search method in [9]
    While stopping condition is not met Do
        Select k variables randomly among variables from X
        Reduce the n-queens maximization problem using unselected variables from X
        Add objective and all constraints to IP
        X' = Get an optimal solution for the reduced problem with IP
        X = a new solution from X'
    End While
    return X
End Begin
```

Fig. 4. Integer Programming-based Local Search for the *n*-queens Maximization Problem

D. Initial Solution Generation

When starting from a candidate solution not satisfying all constraints of the *n*-queens problem, it can be hard to even find a solution satisfying all constraints of the *n*-queens problem. So, an initial solution of *n*-QMP is defined as a solution that satisfies all constraints of the *n*-queens problem. In this research, the initial solution is obtained using the method in [9]. The method can find a solution of the *n*-queens problem very quickly using an efficient local search which is similar to random-restart hill-climbing search.

In [9], an initial search phase and a final search phase are carried out in turn to find a solution. During the initial search, an initial permutation of the row positions of the queens is determined one by one from left column to right column. At this time, the row position for each new queen is generated so as not to violate the positions of the columns to the left, if possible. But even so, since conflicts on diagonal lines are inevitable from a certain column, the remaining queens are placed randomly on empty rows in columns on the right regardless of conflicts on diagonal lines. During the final search, two queens are selected to swap. If the swap reduces the number of collisions then the two queens are swapped. The final search is repeated until a solution is found. If no solution could be found after a certain number of swaps, a new search process is started from an initial search phase again.

[9] mentions that the algorithm is capable of solving the *n*-queens problem in linear time, and could find a solution for the 3,000,000-queens problem in less than 55 seconds. In our research, the algorithm was implemented again. According to the experimental results, the 3,000,000-queens problem could be solved in about 1.3 seconds in our experimental environment, and furthermore the 500,000,000-queens problem could be also solved in about 7 minutes. It was confirmed that the algorithm can find a solution for the *n*-queens problem in a very short time. Therefore, the algorithm was adopted in our research as the initial solution generation method for making a valid initial solution of *n*-QMP, namely a solution of the *n*-queens problem.

E. Problem Reduction

After selecting k variables, the n-QMP is reduced using the information of $(n - k)$ unselected variables. The two methods can be used to reduce the problem. One is a simple way to fix the values of the variables, and the other is a method to reduce the number of variables of the original problem by physically excluding the variables for the next IP. They are called "weak problem reduction" and "strong problem reduction" respectively in this paper. We developed a weak problem reduction method and a strong problem reduction method for n-QMP and compared the performance of the proposed problem reduction methods.

The weak problem reduction is very simple. The values of $(n - k)$ unselected variables are fixed to the current values in the current solution, and then IP is run. Let's assume that there is an 8-QMP in Fig. 2(b), and assume that the current solution is [4, 6, 1, 5, 2, 0, 7, 3], the value of k is 5, and the selected columns are (0, 1, 3, 4, 5) as shown in Fig. 5(a). In this case, as shown in Fig. 5(b), the values of the unselected columns (2, 6, 7) are fixed to 0 or 1 according to the current solution. And the values of the selected columns (0, 1, 3, 4, 5) are invalidated, and determined by the next IP. After all, the problem for the next IP becomes a much easier problem than the original n-QMP since the values of some variables are already determined. However, the number of variables for the next IP is the same as that of the original n-QMP. If the number of variables for the next IP is really minimized, it can be expected to be able to obtain the same solution more quickly. The strong problem reduction is just the process of physically removing the variables for the next IP.

Fig. 5(c) shows the strong problem reduction for n-QMP presented in this research. Using the current solution values of the unselected columns (2, 6, 7), namely the positions with value 1 of the columns (2, 6, 7), all variables violating constraints are removed. This is because the values of the removed variables cannot be 1 by the columns (2, 6, 7). After removing all unneeded variables, only 14 variables out of a total of 64 variables are remained for the next IP. Of course, when adding the objective function and constraints for the next IP, the expressions for the remained 14 variables should be made taking into account the original locations on the 8×8 board. In this research, it will be confirmed that the strong problem reduction can significantly improve the performance in terms of speed of execution through experiment.

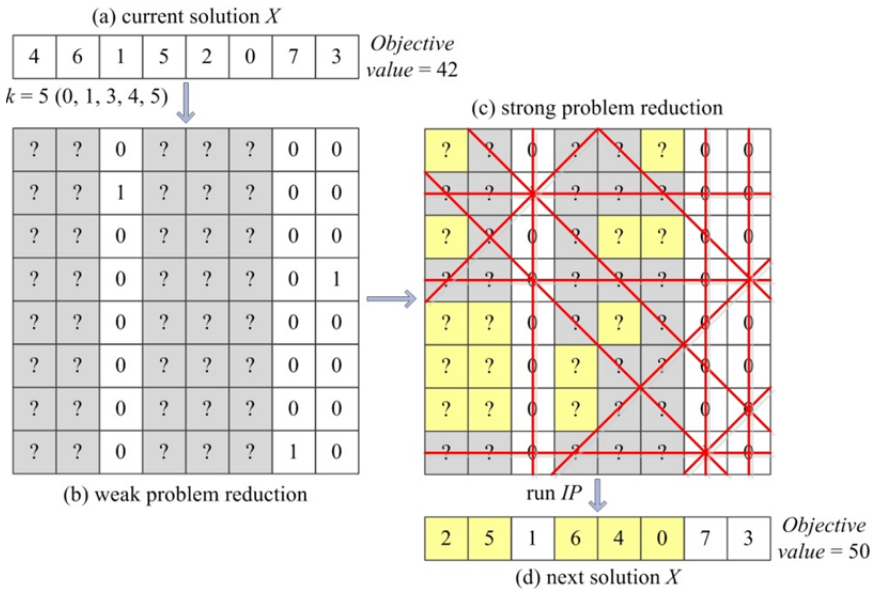


Fig. 5.An Example of Problem Reduction

IV. EXPERIMENTAL RESULTS

F. Experimental Environment

The experimental data set consists of 7 problem instances which consist of 500-queens, 1000-queens, 1500-queens, 2000-queens, 3000-queens, 5000-queens, and 10000-queens maximization problems. The value of weight w_{ij} for each problem was randomly assigned to a value from 1 to n.

All experiments were performed on a PC with Intel Core i7-4770 3GHz CPU, 16GB RAM, and Windows 7 64bit OS. The program was implemented in C++ language, and IBM ILOG CPLEX 12.6.2 was used to implement integer programming (IP) and integer programming-based local search (IPbLS) for solving the n-queens maximization problem (n-QMP). IBM ILOG CPLEX is the most widely used linear and integer programming library in the world for commercial or academic purposes.

G. Experimental Results of Problem Reduction for IPbLS

Table 1 shows the experimental results of the weak problem reduction (WPR), and Table 2 shows the experimental results of the strong problem reduction (SPR). The experimental result value is the average value of 5 runs for each data instance and each *k* value. Each run was performed for 1 hour (3,600 seconds).

When the number of queens is small, the performance was better when the value of *k* is relatively small in both of WPR and SPR. And also the performance was getting better when the value of *k* is relatively big as the number of queens is getting larger. The important thing is that the result of SPR is much better than the result of WPR. Such a phenomenon is clearer as the number of queens increases. The best values of WPR and SPR are 244,949 and 245,979 respectively in 500-queens maximization problem, so the solution gap between WPR and SPR is less than 0.5%. On the other hand, the best values of WPR and SPR are 50,264,357 and 99,193,680 respectively in 10000-queens maximization problem, so the solution gap reaches 97%. From this experiment, we can confirm the importance of the strong problem reduction in IPbLS.

TABLE I. EXPERIMENTAL RESULTS OF WEAK PROBLEM REDUCTION

n \ *k*	50	100	150	200	250	300	350	400	best value
500	241063	**244949**	181294	173150	184157	197740	208676	219322	**244949**
1000	932595	970198	**980688**	903423	614473	641792	665200	696666	**980688**
1500	1996482	2151238	2190337	**2207446**	1873475	1344247	1383364	1411537	**2207446**
2000	3214903	3716601	3852893	3900562	**3923460**	3234847	2318475	2379787	**3923460**
3000	5631263	7054071	7876096	8312628	8540143	8680689	**8689523**	6913946	**8689523**
5000	12813331	13537008	14111259	14876549	15384624	15973241	**16250054**	16047487	**16250054**
10000	50170886	49835915	49887356	50192769	50102855	50017790	**50264357**	50215590	**50264357**

TABLE II. EXPERIMENTAL RESULTS OF STRONG PROBLEM REDUCTION

n \ *k*	50	100	150	200	250	300	350	400	best value
500	244953	**245979**	177332	173553	185069	197085	209000	220047	**245979**
1000	982428	985981	**988291**	866780	620134	636290	674654	691673	**988291**
1500	2212140	2220007	2225100	**2227900**	2104093	1337865	1371963	1418984	**2227900**
2000	3930080	3946495	3955431	3961137	**3964159**	2817846	2378452	2386355	**3964159**
3000	8836811	8874717	8896492	8910334	8918713	8927418	**8929771**	6317537	**8929771**
5000	24502292	24625294	24690493	24735919	24759901	24784224	24804155	**24812101**	**24812101**
10000	96457108	98118397	98616259	98832619	98957115	99044904	99132203	**99193680**	**99193680**

H. Comparison with Other Algorithms

Table 3 shows the experimental results by pure integer programming (PIP), random-restart hill-climbing search (RRHCS), random-restart first-choice hill-climbing search (RRFCHCS), simulated annealing (SA), and IPbLS.

RRHCS makes an initial solution using the same method in IPbLS. A neigbor solution is made by swapping two values of two columns, and RRHCS moves to a best neighbor solution among all neighbor solutions satisfying all constraints of the *n*-queens problem. When the solution is not better than the current solution, RRHCS starts from a new initial solution again. RRFCHCS generates only one neighbor solution, and moves to the solution if the solution is better than the current solution. If a better solution than the current solution is not found even after performing during a certain number of iterations, RRFCHCS starts from a new initial solution again. In this research, the number of iterations was empirically defined as (${}_nC_2 \times 10$).

RRFCHCS was better than RRHCS according to the related experiment. So, SA was added as a method to be compared because SA is based on the first-choice hill-climbing search. SA moves a better neighbor solution like RRFCHCS, but SA can move to a worse neighbor solution stochastically with the probability $e^{\triangle E/T}$[10]. In this research, the initial value of T was set to 10, and updated by multiplying 0.999999 per 10,000 iterations. The values of the parameters were empirically determined from various experiments. In Table 3, the result values of RRHCS, RRFCHCS, SA are the average values of 5 runs, and each run was performed for 1 hour just like IPbLS. The result values of IPbLS show the best values from Table 1 and Table 2.

It was not possible to apply PIP for over 2000-queens maximization problem. Furthermore, the performance of PIP was very bad for even 1000-queens, 1500-queens maximization problems when compared with the other search methods. Through this experiment, we can see that it is difficult to find a good solution for *n*-QMP by PIP alone.

SA shows better performance than RRHCS or RRFCHCS. In addition, though SA is worse than IPbLS using WPR for from 500-queens to 2000-queens, SA shows better results for over 3000-queens. In the simple IPbLS using WPR, overall performance is decreased as the size of the problem increases because the excessive time is required for performing IP. However, IPbLS using SPR showed better results than any other search methods for all data instances. We can confirm the importance of SPR in IPbLS once again through this experiment.

TABLE III. COMPARISON WITH OTHER SEARCH ALGORITHMS

n \ *k*	PIP	RRHCS	RRFCHCS	SA	IPbLS with WPR	IPbLS with SPR
500	244023	230964	231653	238713	244949	**245979**
1000	498610	942355	944514	960385	980688	**988291**
1500	1167985	2140004	2143100	2171871	2207446	**2227900**
2000	fail	3827007	3831513	3872823	3923460	**3964159**
3000	fail	8671997	8681849	8752101	8689523	**8929771**
5000	fail	24279245	24309101	24425492	16250054	**24812101**
10000	fail	69493970	98014855	98183647	50264357	**99193680**

V. CONCLUSION

In this paper, we investigated the importance of the problem reduction in the integer programming-based local search (IPbLS). First, we proposed two problem reduction methods in IPbLS. One is weak problem reduction where the values of variables are simply fixed, and the other is strong problem reduction where some variables are physically removed from the original problem. Experimental results for the *n*-queens maximization problem showed that the performance of IPbLS can be significantly improved through the strong problem reduction. Especially, we confirmed that the strong problem reduction becomes more important as the problem size increases. We can see that using the strong problem reduction is more helpful to find a better solution more quickly when applying IPbLS to solve combinatorial optimization problems.

There is another key parameter influencing the performance of IPbLS besides problem reduction method. It is just the value of *k*. The value of *k* is still determined by a number of experiments, but it is necessary to develop a method to be able to automatically determine the optimal *k* value through a future study about the correlation between the value of *k* and the search performance.

ACKNOWLEDGMENT

This paper was supported by Kumoh National Institute of Technology.

REFERENCES

[1] M.R. Fellows, F.A. Rosamond, F.V. Fomin, D. Lokshtanov, S. Saurabh, and Y. Villanger, "Local search: Is brute-force avoidable?," Journal of Computer and System Sciences, vol. 78, no. 3, pp. 707-719, May 2012,.
[2] S. Russell, and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd Ed., Prentice-Hall, pp. 120-125, 2010.
[3] M. Conforti, G. Cornuéjols, and G. Zambelli, Integer Programming, Springer, 2014.
[4] J. Hwang, "Integer programming-based local search technique for linear constraint satisfaction optimization problem," Journal of The Korea Society of Computer and Information, vol. 15, no. 9, pp. 47-55, Sep. 2010.
[5] J. Hwang, S. Park, and I. Y. Kong, "An integer programming-based local search for large-scale multidimensional knapsack problems," International Journal on Computer Science and Engineering, vol. 3, no. 6, pp. 2257-2264, June 2011.
[6] J. Hwang, "An integer programming-based local search for the set covering problem," Journal of The Korea Society of Computer and Information, vol. 19, no. 10, pp. 13-21, Oct. 2014.
[7] A. Erera, M. Hewitt, M. Savelsbergh, and Y. Zhang, "Improved load plan design through integer programming based local search," Transportation Science, vol. 47, no. 3, pp.412-427, Nov. 2012.
[8] J. Paredis, "Genetic state-space search for constrained optimization problems," in Proc. IJCAI-93, 1993, pp.967-973.
[9] R. Sosic, and J, Gu, "Efficient local search with conflict minimization: A case study of the n-queens problem," IEEE Trans. on Knowledge and Data Engineering, vol. 6, no. 5, pp. 661-668, 1994.
[10] R. A. Rutenbar, "Simulated annealing algorithms: An overview," IEEE Circuits and Devices Magazine, vol. 5, no. 1, pp. 19-26, Jan. 1989.

AUTHOR PROFILE

Junha Hwang received his B.S., M.S. and Ph.D. degree in Computer Engineering from Pusan National University, Korea in 1995, 1997 and 2002 respectively. He is a Professor in Dept. of Computer Engineering, Kumoh National Institute of Technology, Korea since 2002. His main research interests are combinatorial optimization, machine learning and artificial intelligence.