# An Integer Programming-based Local Search for Large-scale Maximal Covering Problems

Junha Hwang[1], Sungyoung Kim[2]
Department of Computer Engineering
Kumoh National Institute of Technology
Gumi, Korea
[1]jhhwang@kumoh.ac.kr, [2]sykim@kumoh.ac.kr

***Abstract*—Maximal covering problem (MCP) is classified as a linear integer optimization problem which can be effectively solved by integer programming technique. However, as the problem size grows, integer programming requires excessive time to get an optimal solution. This paper suggests a method for applying integer programming-based local search (IPbLS) to solve large-scale maximal covering problems. IPbLS, which is a hybrid technique combining integer programming and local search, is a kind of local search using integer programming for neighbor generation. IPbLS itself is very effective for MCP. In addition, we improve the performance of IPbLS for MCP through problem reduction based on the current solution. Experimental results show that the proposed method considerably outperforms any other local search techniques and integer programming.**

***Keywords-Maximal Covering Problem; Integer Programming; Local Search; Integer Programming-based Local Search***

## I. INTRODUCTION

Maximal Covering Problem (MCP) is to choose *d* columns out of *n* columns so as to maximize the number of covered rows from the given *m* by *n* 0-1 matrix [1]. MCP is known as an NP-hard optimization problem. By the way, since MCP can be formulated as a linear expression, integer programming (IP) can be basically used to solve it [2]. However, IP severely suffers from the curse of dimensionality as the problem size grows. Thus IP needs too much time to find a sub-optimal solution as well as an optimal solution when the size of the given MCP is large. To overcome this problem, local search techniques like tabu search, simulated annealing and genetic algorithm have been applied to MCP [3]. Local search can find a good solution for large-scale MCP within a relatively short time even if it does not guarantee an optimal solution.

In this paper, we suggest a method to apply integer programming-based local search (IPbLS) to solve MCP. IPbLS is a recent method proposed for linear combinatorial optimization problems [4, 5, 6]. It is basically like first choice hill climbing [7] and uses integer programming as a tool for neighbor generation. Unlike general local search, IPbLS can not only consider larger neighborhood to get a neighbor solution but also find an optimal neighbor solution. According to the experimental results, IPbLS itself is effective to solve MCP. In addition, we improve the performance of IPbLS for MCP through problem reduction which is achieved by selecting subset of rows and columns needed for the current stage's IP using the current solution. We could confirm the superiority of the proposed method through the experimental results.

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

## II. RELATED WORK

### A. Maximal Covering Problem

Fig. 1 shows an example of MCP which is composed of 5 rows and 4 columns. If the value of *d* is given as 2, the optimal solution is to select column 1 and 4.

MCP can be expressed as the linear equation (1) [8]. However, this formula means minimizing uncovered rows instead of maximizing covered rows of the original MCP. Finally, the two expressions are completely identical. In equation (1), the objective function minimizes the number of uncovered rows, and the first and second constraints mean selecting *d* columns from the *n* columns. In the third constraint, $a_{ij}$ is a 0-1 constant value of row *i* and column *j*. The objective is achieved by the third and fourth constraints since the value of $y_i$ becomes 1 if row *i* is not covered.

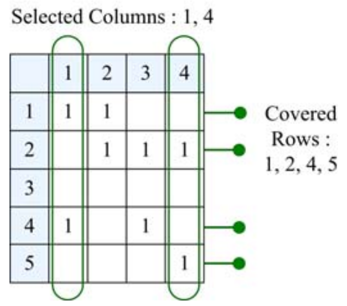


Figure 1. Example of Maximal Covering Problem

$$\begin{aligned}
&\textit{Minimize} && \sum_{i=1}^{m} y_i \\
&\textit{Subject to} && \sum_{j=1}^{n} x_j = d \\
& && x_j \in \{0,1\},\ j = 1,\dots,n \\
& && \sum_{j=1}^{n} a_{ij} x_j + y_i \ge 1 \\
& && y_i \in \{0,1\},\ i = 1,\dots,m
\end{aligned} \tag{1}$$

Many real-world problems can be expressed as MCP and the representative applications are crew scheduling problem and facility location problem [9, 10]. Crew scheduling is the process of assigning crews to operate transportation systems such as airplane, train, subway or bus. Facility location is the process of selecting locations of given facilities so as to maximize served customers. Integer programming has been applied to solve MCP due to its linear nature [1, 11]. Integer programming guarantees an optimal solution. However, it is impractical to apply to real-world large-scale MCP since it is difficult to get a good solution when the size of the problem is large. In recent years, some researchers have begun to focus on local search. Genetic algorithm has been used which includes unexpressed genes to maintain the diversity of the search [12], and first choice hill climbing and simulated annealing have been applied for MCP [13].

## B. Integer Programming-based Local Search

Fig. 2 shows the structure of IPbLS. IPbLS is basically based on first choice hill climbing. Most of the combinatorial optimization problems can be expressed as *n* variables, and a neighbor solution can be made by changing the values of *k* variables out of *n* variables. General first choice hill climbing makes neighbor solutions by randomly changing the values until one is generated that is better than the current solution, and then moves to the neighbor solution [7].

```
Algorithm IPbLS
    $x$ : Variable vector (Current solution).
    $Obj$ : Objective function.
    $k$ : 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 $Obj$ and all constraints to $IP$
            • Fix values of unselected variables from $x$
        $x$ = Make a neighbor solution with $IP$
    End While
    return $x$
End Begin
```

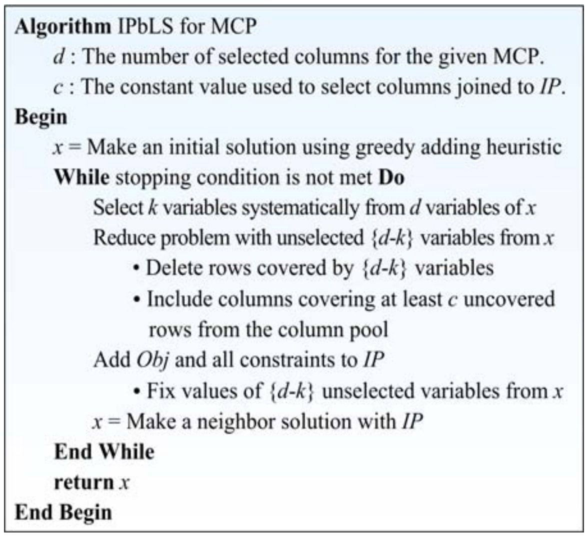
Figure 2. General Integer Programming-based Local Search

However, IPbLS selects *k* variables and then moves to a locally optimal neighbor solution since it considers all the neighborhood solutions being able to be made by changing the *k* variables. Integer programming used as a neighbor generation tool makes this possible. In addition, *k* of IPbLS can be set to a far larger value unlike local search including first choice hill climbing where the value of *k* is set relatively small. This is also due to integer programming which can find an optimal solution more efficiently when *k* is relatively large.

In [9], a similar approach to IPbLS was proposed, in which tabu search and integer programming were combined and integer programming was used as a diversification tool. However, while IPbLS continuously uses integer programming for neighbor generation, [9] rarely uses integer programming just for diversification. In other words, [9] did not actively use integer programming. Some researches have used IPbLS to solve various optimization problems [4, 5, 6]. Reference [4] introduced IPbLS to solve a nurse scheduling problem and [5] adopted IPbLS to solve a classic network design problem. In [6], IPbLS was used to solve N-Queens maximization problem which is a kind of linear constraint satisfaction optimization problem. Of course, all the problems can be modeled as linear equations and solved by integer programming. In this paper, we suggest IPbLS for MCP by taking advantage of the characteristics of MCP.

## III. Integer Programming-based Local Search for Maximal Covering Problem

Fig. 3 shows the overall structure of IPbLS for MCP which we propose in this paper. A solution in MCP can be also expressed by *n* variables and each variable has value 0 or 1. The value 1 of a certain variable means that the corresponding column is selected and the value 0 means that the corresponding column is not selected. First, an initial solution is generated by greedy adding heuristic which selects *d* columns by repeatedly selecting a column so as to maximally cover yet uncovered rows [1, 9]. When some columns cover the same number of uncovered rows, one of them is randomly selected.



```
Algorithm IPbLS for MCP
    d : The number of selected columns for the given MCP.
    c : The constant value used to select columns joined to IP.
Begin
    x = Make an initial solution using greedy adding heuristic
    While stopping condition is not met Do
        Select k variables systematically from d variables of x
        Reduce problem with unselected {d-k} variables from x
            • Delete rows covered by {d-k} variables
            • Include columns covering at least c uncovered
              rows from the column pool
        Add Obj and all constraints to IP
            • Fix values of {d-k} unselected variables from x
        x = Make a neighbor solution with IP
    End While
    return x
End Begin
```

Figure 3. Integer Programming-based Local Search for MCP

After setting the initial solution to the current solution, IPbLS repeatedly performs the process of generating a neighbor solution and moving to the solution until a certain stopping criteria is met. Basic method for generating a neighbor solution is to invalidate the value of *k* columns out of *d* columns included in the current solution, and then select new *k* columns from the currently unselected columns including invalidated *k* columns in the column pool. Some literatures call this process *k*-exchange in that it changes no more than *k* columns [12, 13]. The more specific methods for each step are as follows.

- Selecting *k* variables systematically from *d* variables included in the current solution *x* : we can randomly select *k* variables, but this makes difficult to improve the current solution since the search becomes similar to a random search. Therefore, it is better to select columns systematically so as to raise the possibility of improving the current solution. In other words, we need to select columns in the direction of minimizing the number of newly generated uncovered rows if possible. To do this, two methods can be considered. One is to select an optimal combination using integer programming and the other is to select one by one using a greedy scheme. The former, by the way, rather tends to stuck in local optima since it is difficult to change the current solution due to the lack of randomness. Thus, we adopt a greedy scheme in this paper. If tie occurs, a column is selected at random.

- Reducing problem using unselected {*d*-*k*} variables : The original MCP consists of *m* rows and *n* columns, and should select *d* columns. However, in the each iteration for IPbLS, the problem is reduced based on the information of {*d*-*k*} unselected columns, namely unchanged columns. For example, suppose that the values of *m*, *n*, *d* are 6, 10, 4 respectively and unselected columns from the current solution are column 1 and 6 as shown in Fig. 4(a). At this point, we don't have to consider rows 1, 2 and 4, since column 1 and 6 won't be changed and the rows are covered by them. The rows can be, therefore, reduced like Fig. 4(b). Now we select the columns to participate to the current IP out of 8 columns in the column pool except for column 1 and 6. At this point, we just select the columns covering more than *c* uncovered rows. If we suppose that the value of *c* is 1 and just the columns 4, 5 and 8 cover more than 1 column, the reduced problem is like Fig. 4(c). Finally, the reduced problem is to select 2 columns from MCP having 3 rows and 3 columns. The objective of problem reduction is to improve the performance of IP which is sensitive to the problem size. It seems not to degrade the final solution quality even if the value of *c* is somewhat large. The appropriate values of *c* may be different by the given problems or data.

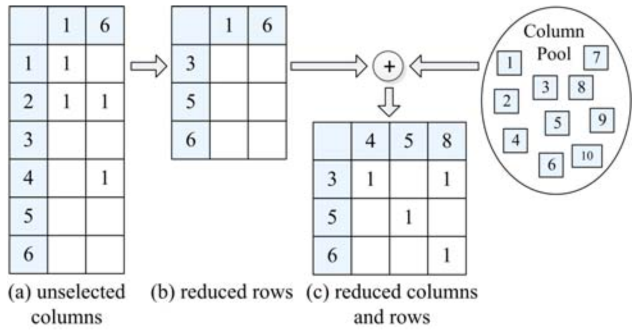


Figure 4. Problem Reduction Steps

- Making a neighbor solution with IP : At last, IP solves the reduced problem to generate an optimized neighbor solution.

While general IPbLS always moves to an equal or better neighbor solution since it is based on first choice hill climbing, the IPbLS for MCP proposed in this paper can move to a worse neighbor solution than the current solution according to the participating columns for IP. In this respect, the IPbLS for MCP is similar to simulated annealing which allows moving to a worse solution.

## IV. EXPERIMENTAL RESULT

### A. Experimental Environment

Table I shows the experimental data. MCP312~MCP814 can be classified as small-scale or medium-scale problems, and have been used for previous researches [12] and [13]. MCP1000~MCP3000 can be classified as large-scale problems and were prepared by ourselves for this paper. As shown in the Table I, the data include various numbers of rows, columns, selected columns and covered rows per column.

TABLE I. EXPERIMENTAL DATA

| Problem | rows(m) | columns(n) | selected columns(d) | covered rows per column |
|---|---|---|---|---|
| MCP312 | 312 | 55,807 | 39 | 7, 8 |
| MCP384 | 384 | 40,544 | 40 | 9, 10 |
| MCP453 | 453 | 72,094 | 45 | 9, 10 |
| MCP520 | 520 | 111,578 | 53 | 8, 9, 10 |
| MCP634 | 634 | 142,265 | 65 | 8, 9, 10 |
| MCP814 | 814 | 179,514 | 83 | 8, 9, 10 |
| MCP1000 | 1000 | 250,000 | 95 | 10, 11, 12 |
| MCP1400 | 1400 | 100,000 | 160 | 9, 10 |
| MCP2000 | 2000 | 300,000 | 270 | 8 |
| MCP3000 | 3000 | 600,000 | 350 | 10 |

All the experiments were carried on a PC with Intel Core2 Duo with 3GHz and 2GB memory. Execution time of one run was limited to 20 minutes which is enough time to converge for most data and algorithms, and all the experimental results were averaged over 10 run. The value of the result means the number of uncovered rows, and the smaller this value is the better. We used ILOG CPLEX 12.1 as an integer programming development tool [14]. ILOG CPLEX is the present most widely used linear and integer programming library in the world for commercial and academic purpose.

## B. *Experimental Results*

The first experiment is about the influence of parameter setting of IPbLS for MCP. There are two parameters to consider. One is the value of *k* that is the number of selected columns to be changed, and the other is the value of *c* that is the minimum number of covering uncovered rows when we select columns from the column pool. Table II shows the results of MCP520, MCP1400 and MCP3000, in which the vertical axis means the value of *k* and the horizontal axis means the value of *c*.

TABLE II. RESULTS OF IPbLS FOR MCP520, MCP 1400, MCP3000

(a) MCP520

| *c* *k* | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **Avg.** |
|---|---|---|---|---|---|---|---|---|---|---|
| **10** | 4.2 | 7.5 | 5.5 | 4.3 | 4.0 | 5.6 | 5.7 | 11.5 | 41.3 | 10.0 |
| **15** | 4.0 | 4.3 | 4.4 | 4.4 | 4.2 | 4.5 | 4.5 | 4.4 | 4.4 | 4.3 |
| **20** | 2.4 | 2.2 | 2.3 | 2.5 | 2.5 | **2.0** | **2.0** | **2.0** | 3.3 | 2.4 |
| **25** | **2.0** | **2.0** | **2.0** | **2.0** | **2.0** | **2.0** | **2.0** | **2.0** | **2.0** | **2.0** |
| **30** | 3.0 | 2.1 | **2.0** | 2.1 | **2.0** | 2.2 | **2.0** | **2.0** | **2.0** | 2.2 |
| **35** | 2.3 | 2.3 | 3.3 | 2.6 | 2.6 | 2.6 | 2.3 | 2.4 | 2.0 | 2.5 |
| **40** | 2.3 | 2.2 | 2.9 | 2.0 | 2.8 | 2.9 | 2.2 | 2.7 | 2.2 | 2.5 |
| **45** | 2.2 | 2.6 | 2.3 | 2.4 | 2.2 | 2.3 | 2.5 | 2.3 | 2.4 | 2.4 |
| **Avg.** | 2.8 | 3.2 | 3.1 | **2.8** | **2.8** | 3.0 | 2.9 | 3.7 | 7.5 | 3.5 |

(b) MCP1400

| *c* *k* | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **Avg.** |
|---|---|---|---|---|---|---|---|---|
| **10** | 29.7 | 30.7 | 29.3 | 28.8 | 30.2 | 39.7 | 43.0 | 33.1 |
| **15** | 28.1 | 27.9 | 28.4 | 28.5 | 28.7 | 30.0 | 43.0 | 30.7 |
| **20** | 28.4 | 28.3 | 28.1 | 28.4 | 27.8 | 29.2 | 43.0 | 30.5 |
| **25** | 26.3 | 26.7 | 26.6 | **25.3** | 25.9 | 27.1 | 40.1 | 28.3 |
| **30** | 27.3 | 27.0 | 27.0 | 27.0 | 26.1 | 26.1 | 28.0 | **26.9** |
| **35** | 29.9 | 31.0 | 31.4 | 31.2 | 28.4 | 28.6 | 30.8 | 30.2 |
| **40** | 32.5 | 31.6 | 31.1 | 31.4 | 30.8 | 29.4 | 34.9 | 31.7 |
| **45** | 31.8 | 31.6 | 31.8 | 31.6 | 31.2 | 30.8 | 32.3 | 31.6 |
| **Avg.** | 29.3 | 29.4 | 29.2 | 29.0 | **28.6** | 30.1 | 36.9 | 30.4 |

(c) MCP3000

| *c* *k* | **0** | **1** | **2** | **3** | **4** | **5** | **Avg.** |
|---|---|---|---|---|---|---|---|
| **10** | 7.1 | 6.0 | 5.8 | 6.1 | 17.5 | 18.0 | 10.1 |
| **15** | 5.5 | 5.5 | 5.0 | 4.6 | 6.7 | 18.0 | 7.6 |
| **20** | 4.7 | 4.5 | 4.3 | 4.3 | 6.1 | 18.0 | 7.0 |
| **25** | 4.1 | 4.4 | 3.9 | 4.9 | 5.5 | 18.0 | 6.8 |
| **30** | 3.8 | 3.6 | 3.5 | **3.2** | 4.4 | 18.0 | **6.1** |
| **35** | 5.2 | 4.9 | 5.1 | 4.1 | 5.4 | 14.7 | 6.6 |
| **40** | 6.2 | 6.1 | 6.2 | 5.7 | 5.8 | 6.7 | 6.1 |
| **45** | 7.2 | 6.9 | 6.3 | 5.6 | 5.8 | 9.5 | 6.9 |
| **Avg.** | 5.5 | 5.2 | 5.0 | **4.8** | 7.2 | 15.1 | 7.1 |

According to the Table II(a) that is the result of MCP520, the search performance is the best when the value of *k* is 25 or 30, and the search performance is gradually degraded as the value of *k* increases or decreases. From the result, we can see the effect according to the value of *k*. However, there seems to be no effect by changing the value of *c* except for too large values like 7 and 8. Now let's take a look at Table II(b) and Table II(c) which are the results of MCP1400 and MCP3000 respectively. In MCP1400, we can see that the search performance is the best when the value of *k* is 30 and the value of *c* is 4, and the performance is gradually degraded as the values of *k* and *c* are increase or decrease. MCP3000 shows similar results around when the value of *k* is 30 and the value of *c* is 3. We can see that the search performance depends on the value of *k* and *c* through this experiment, and especially the value of *c* is more influential for the larger-scale problems.

The second experiment is about comparison with other search algorithms. This result supports the conclusion of this paper. Used search algorithms for comparison are integer programming (IP) and Local Search (LS), and

LS includes first choice hill climbing (FHC), simulated annealing (SA) and tabu search (TABU). ILOG CPLEX for implementing IP provides a branch & cut algorithm and a dynamic search algorithm [14]. According to the IP related experiment, we could confirm the dynamic search algorithm is superior to the branch & cut algorithm, therefore we used the dynamic search algorithm for IP. We should also determine some parameters for local search algorithms including first choice hill climbing. We can confirm that 5-exchange is the most effective to generate a neighbor solution by previous research [13]. Thus, we used 5-exchange for neighbor generation, and we also used the best parameter settings discovered in the previous research [13] for the other parameters. The experimental results are as Table III. In relatively small-scale problems, an optimal solution can be found in the given execution time, and the numbers in parenthesis indicate the execution time in seconds when an optimal solution is found. The numbers of IPbLS indicate averages of ten runs when the best *k* and *c* values are applied for each problem. The pairs of the best (*k*-*u*) for each problem are as follows: MCP312 (20-8), MCP384 (35-2), MCP453 (30-6), MCP520 (25-7), MCP634 (30-8), MCP814 (30-6), MCP1000 (15-6), MCP1400 (25-3), MCP2000 (30-5) and MCP3000 (30-3).

TABLE III. COMPARISON OF SEARCH ALGORITHMS

| Alg. / Problem | IP | LS | | | IPbLS |
|---|---|---|---|---|---|
| | | FHC | SA | TABU | |
| **MCP312** | **0(44)** | 1.6 | 1.2 | 3 | **0(17)** |
| **MCP384** | **1(71)** | 5.0 | 4.3 | 6.1 | 1.2 |
| **MCP453** | **8(72)** | 12.1 | 11.6 | 13.3 | **8(296)** |
| **MCP520** | **2(1193)** | 4.7 | 4.1 | 6.7 | **2(97)** |
| **MCP634** | 120 | 5.1 | 4.8 | 8.6 | **0(253)** |
| **MCP814** | 164 | 8.2 | 7.6 | 11.7 | **1.7** |
| **MCP1000** | 331 | 31.3 | 35 | 30.8 | **25.1** |
| **MCP1400** | 455 | 32.3 | 34.1 | 30 | **25.3** |
| **MCP2000** | 648 | 34.2 | 37.6 | 35.4 | **25.2** |
| **MCP3000** | 943 | 8.0 | 12.4 | 8.7 | **3.2** |

According to the Table III, we can see that IP is the best for relatively small-scale problems like MCP312~MCP520, and furthermore, IP can find an optimal solution in a very short time. However, IP cannot find a comparable solution as well as an optimal solution for medium-scale and large-scale problems like MCP634~MCP3000 in the given execution time. LS can find better solutions than IP for large-scale problems. FHC and SA are somewhat better than TABU for small-scale problems but FHC and TABU are somewhat better than SA for large-scale problems. Moving to a worse solution like SA seems not helpful for especially large-scale problems, but this phenomenon in TABU seems to be relaxed since TABU selects a neighbor solution from many candidate solutions. However, IPbLS is far better than LS for all the problems, and far better than IP for medium-scale and large-scale problems. In addition, IPbLS is never inferior to IP in terms of both solution quality and execution time even for small-scale problems. IPbLS can rather find an optimal solution more quickly for MCP312 and MCP520, and IPbLS can also find an optimal solution for MCP634. We can confirm the superiority of IPbLS for MCP through this experiment.

Finally, for reference, Fig. 5 shows the typical performance curve of the search algorithms with respect to time for MCP2000.

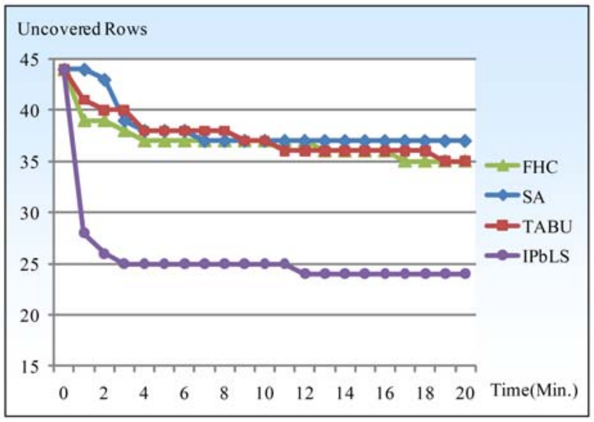


Figure 5. Performance Curve of Search algorithms w.r.t Time

IP could find a solution that has 648 uncovered rows, but there was no change since then. Thus IP was excluded from Fig. 5. We can see SA is so slowly improved by unnecessarily moving to worse solutions in early search stage, and FHC and TABU are relatively superior to SA. However, we can confirm that IPbLS is far superior to all the other algorithms through not only early stage but also all the time. This trend is similarly observed in all the other test problems. We can confirm the superiority of IPbLS once again.

## V. CONCLUSION AND FUTURE WORK

In this paper, we have presented a method applying IPbLS for MCP. Primarily IPbLS itself was adopted for MCP and problem reduction using the current solution was introduced to improve the performance of IP for neighbor generation. We could confirm that the proposed scheme is very effective for MCP in comparison to pure IP and other local search algorithms. There are some future works. First, IPbLS can be adopted many other optimization problems as well as MCP. Thus we have a plan to develop IPbLS for other optimization problems like set covering problem, knapsack problem, and so on. Second, we will investigate theoretically the effect of IPbLS since there is no sufficient theoretical background. In addition, the research for the automatic parameter settings for IPbLS will be done in parallel.

## REFERENCES

[1] R. L. Church, and C. S. ReVelle, “The maximal covering location problem,” Regional Science, vol. 32, pp. 101-118, 1974.

[2] L. A. Wolsey, Integer Programming, Wiley, 1998.

[3] E. Aarts, and J. K. Lenstra, Local Search in Combinatorial Optimization, Wiley, 2003.

[4] S. Hasegawa, and Y. Kosugi, “Solving nurse scheduling problem by integer-programming-based local search,” 2006 IEEE International Conference on Systems, Man and Cybernetics, vol. 2, pp. 1474-1480, 2006.

[5] M. Hewitt, G. L. Nemhauser, and M. W. Savelsbergh, “Combining exact and heuristic approaches for the capacitated fixed charge network flow problem”, INFORMS Journal on Computing, vol. 22, pp. 314-325, 2009.

[6] 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, 2010.

[7] S. Russel, and P. Norvig, Artificial Intelligence A Modern Approach, 2nd ed., Prentice Hall, 2003.

[8] J. Hwang, “Integration of integer programming and neighborhood search algorithm for solving a nonlinear optimization problem,” Journal of The Korea Society of Computer and Information, vol. 14, no. 2, pp. 27-35, 2009.

[9] J. Hwang, C. S. Kang, K. R. Ryu, Y. Han, and H, R. Choi, “A hybrid of tabu search and integer programming for subway crew paring optimization,” Proc. of the 6th IASTED International Conference on Artificial Intelligence and Soft Computing, pp. 72-77, 2002.

[10] L. Xia, M. Xie, W. Xu, J. Shao, W. Yin, and J. Dong, “An empirical comparison of five efficient heuristics for maximal covering location problems,” Proc. of the 2009 IEEE International Conference Service Operations and Logistics, and Informatics, pp. 747-753, 2009.

[11] O. Berman, and D. Krass, “The generalized maximal covering location problem,” Computers & Operations Research, vol. 29, no. 6, pp. 563-581, 2002.

[12] T. Park and K. R. Ryu, “Crew pairing optimization by a genetic algorithm with unexpressed genes,” Journal of Intelligent Manufacturing, vol. 17, no. 4, pp. 375-383, 2006.

[13] J. Hwang, “Neighborhood search algorithms for the maximal covering problem,” Journal of The Korea Society of Computer and Information, vol. 11, no. 1, pp. 129-138, 2006.

[14] IBM ILOG, CPLEX User’s Manual and Reference Manual, Version 12.1, 2009.

## AUTHORS 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 an Associate 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.

**Sungyoung Kim** received his B.S., M.S. and Ph.D. degree in Computer Engineering from Pusan National University, Korea in 1994, 1996 and 2003 respectively. He is an Associate Professor in Dept. of Computer Engineering, Kumoh National Institute of Technology, Korea since 2004. His main research interests are computer vision, image processing and artificial intelligence.