Feasibility-Preserving Crossover for Maximum k-Coverage Problem

Yourim Yoon
School of Computer Science & Engineering
Seoul National University
Sillim-dong, Gwanak-gu
Seoul, 151-744, Korea
yryoon@soar.snu.ac.kr

Yong-Hyuk Kim
Department of Computer Science & Engineering
Kwangwoon University
Wolgye-dong, Nowon-gu
Seoul, 139-701, Korea
yhdfly@kw.ac.kr

Byung-Ro Moon
School of Computer Science & Engineering
Seoul National University
Sillim-dong, Gwanak-gu
Seoul, 151-744, Korea
moon@soar.snu.ac.kr

ABSTRACT

The maximum k-coverage problem is a generalized version of covering problems. We introduce the problem formally and analyze its property in relation to the operators of genetic algorithm. Based on the analysis, we propose a new crossover tailored to the maximum k-coverage problem. While traditional n-point crossovers have a problem of requiring repair steps, the proposed crossover has an additional advantage of always producing feasible solutions. We give a comparative analysis of the proposed crossover through experiments.

Categories and Subject Descriptors

G.2.3 [**Discrete Mathematics**]: Applications; G.4 [**Mathematical Software**]: Algorithm Design and Analysis

General Terms

Algorithms, Experimentation

Keywords

Maximum k-coverage, feasibility-preserving crossover, genetic algorithms

1. INTRODUCTION

The maximum k-coverage problem (MKCP) can be considered as a generalized version of covering problems. It was introduced in [9] and has a number of applications to combinatorial optimization problems such as covering graphs by subgraphs, facility location problems, packing and circuit layout design, and scheduling problems.

Let $A = (a_{ij})$ be an $m \times n$ zero-one matrix and weight w_i be given to the i-th row of A for each i. The object of MKCP is to select k columns that cover rows for their weight sum to be maximized. The problem is represented formally as follows:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{m} w_i \cdot I\left(\sum_{j=1}^{n} a_{ij}x_j \geq 1\right) \\ \text{subject to} \quad & \sum_{j=1}^{n} x_j = k \\ & x_j \in \{0,1\}, \quad j = 1, 2, \ldots, n, \end{aligned}$$

where $I(\cdot)$ is an indicator function such that $I(true) = 1$ and $I(false)$=0.

The weight w_i can be different according to i, however, we will focus on the case that each w_i is set to be the same value in this paper. Then MKCP becomes the problem of finding a solution that covers as many rows as possible using a fixed number of columns.

The NP-hardness of MKCP can be easily induced from the minimum set covering problem [7]. Although there have been studies using genetic algorithms for the minimum set covering problem [1, 4], MKCP has not been covered until now. MKCP selects a fixed number of columns, and so it is simpler than in other covering problems to represent a solution. The problem not only matches well with genetic algorithm, but also has an interesting inherent property. We analyze the property of the problem and design a problem-specific crossover for genetic algorithm in this paper.

The remainder of this paper is organized as follows. We analyze the property of MKCP in Section 2 and propose a new crossover based on this property in Section 3. In Section 4, we explain an exceptional advantage of using this crossover, i.e., feasibility-preserving. Finally, we present experimental results in Section 5 and make conclusions in Section 6.

2. CHARACTERISTICS OF MKCP

We can consider as an encoding of the solution for MKCP a length-k integer string whose elements are column indices or a length-n binary string whose elements each represents whether or not the corresponding column is selected. In this study, we focus on the integer string encoding.

Each column of the given matrix A can be considered as a subset composed of column indices with corresponding element 1. For example, $(1\ 0\ 0\ 1\ 0)^T$ corresponds to $\{1, 4\}$. In the following, we will also use this subset representation in addition to the original vector representation. Each subset element is a column index. Hence, the universal set becomes $\{1, 2, \ldots, n\}$.

In MKCP, we want to find a solution that covers as many elements as possible using a fixed number of subsets. As a property of MKCP, we can easily guess that a good solution of MKCP is composed of subsets which do hardly have common elements each other. We will show this property more clearly through the following simple example. Let $k = 2$, $m = 5$, $n = 4$, and 5×4 matrix A be

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

The sets of covered rows for all possible solutions are shown in the following.

solution	coverage	coverage size
$(1,2)$	$\{1,2,3,5\}$	4
$(1,3)$	$\{1,2,4,5\}$	4
$(1,4)$	$\{1,2,3\}$	3
$(2,3)$	$\{3,4,5\}$	3
$(2,4)$	$\{2,3,5\}$	3
$(3,4)$	$\{2,3,4,5\}$	4

As expected, the columns of good solutions ($(1,2)$, $(1,3)$, and $(3,4)$) have fewer common elements than those of relatively bad ones ($(1,4)$, $(2,3)$, and $(2,4)$) as follows.

solution	common element set	coverage size
$(1,2)$	$\emptyset$	4
$(1,3)$	$\emptyset$	4
$(1,4)$	$\{2\}$	3
$(2,3)$	$\{5\}$	3
$(2,4)$	$\{3\}$	3
$(3,4)$	$\emptyset$	4

When we are to solve MKCP using genetic algorithm, we need to design genetic operators to produce chromosomes that have as few common elements between genes as possible. We design a new crossover operator based on this point of view. Consider a case that we apply traditional one-point crossover with the first parent $(1,2)$ and the second one $(3,4)$. If the cut-point lies between the first gene and the second one, the offspring would be $(1,4)$. The solution $(1,4)$ covers the rows $\{1,2,3\}$. However, we rearrange the second parent to be $(4,3)$, the offspring becomes $(1,3)$ and then covers $\{1,2,4,5\}$. The column 1 has no common element with the column 3, but has one common element with the column 4. So, it produces better offspring to locate at the same position with the column 1 the column 4 rather than the column 3. This leads that subsets forming the offspring has fewer common elements. In this simple example, we can obtain an intuition that rearranging gene positions for similar genes, i.e., genes whose corresponding columns have many common elements, to be at the same position would be very helpful to solve MKCP using genetic algorithms.

3. A NEW CROSSOVER

We consider the case of using integer representation as an encoding of a solution. Although a gene has an integer value, the number itself is just a label. Each gene actually indicates the column with the corresponding index. If we use the column vector itself instead of the column index, the solution becomes an $m \times k$ matrix with binary elements. In the above example, the solution $(1,2)$ actually means

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

Both of encodings $(1,2)$ and $(2,1)$ represent the same solution, so

$$\begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}$$

represents the same solution as the above one. Having this representation in mind will be helpful to understand the concept of a new crossover that we are to define in the following.

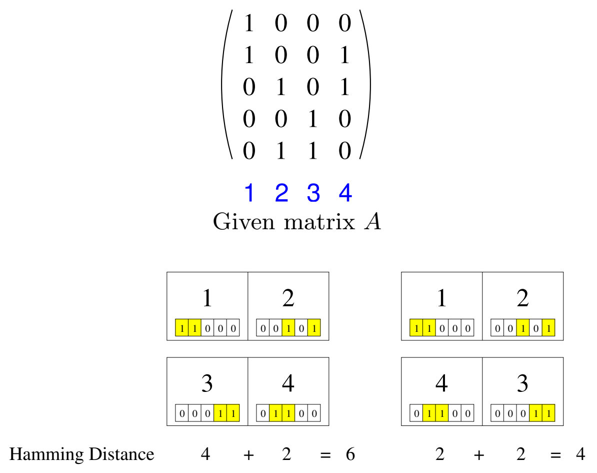


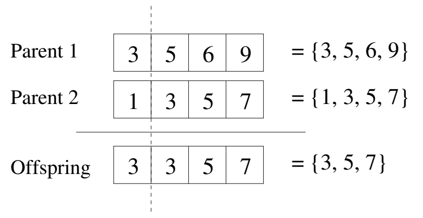
Figure 1: Normalization by Hamming distance

Now we present a new crossover operator for MKCP according to the property given in Section 2. In the previous section, we inferred that the similarity between the genes at the same position of two parents affects the performance of genetic algorithms. We consider all the permutations of columns for the second parent, and choose one that minimizes the sum of distances between pairs of genes with the same indices. We used the Hamming distance H to measure the similarity between two genes. Let the first parent be $x = (x_1, x_2, \ldots, x_k)$, and the second parent be $y = (y_1, y_2, \ldots, y_k)$. We choose the permutation σ^* such that

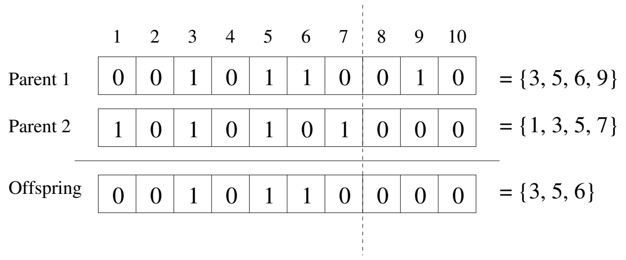
$$\sigma^* = \operatorname*{argmin}_{\sigma \in \Sigma_k} \sum_{i=1}^{k} H(x_i, y_{\sigma(i)}), \qquad (1)$$

where Σ_k denotes the set of all permutations of length k.

We provide an example in Figure 1. Suppose that the chromosomes $(1,2)$ and $(3,4)$ are chosen as two parents. We normalize the second parent $(3,4)$. Since k is 2, the number of possible permutations is only $2! = 2$. For each



(A) Integer representation
($k = 4$, $n = 10$)



(B) Binary representation
($k = 4$, $n = 10$)

Figure 2: An example in which the feasibility is broken

permutation, we compute $\sum_{i=1}^{k} H(x_i, y_{\sigma(i)})$. For the first permutation $\sigma_1 = (1, 2)$, the normalized second parent is, as it is, $(3, 4)$. Each gene indicates the column of the given matrix A. Then $H(x_1, y_{\sigma_1(1)}) = 4$ and $H(x_2, y_{\sigma_1(2)}) = 2$. Their sum is 6. For the second permutation $\sigma_2 = (2, 1)$, the normalized second parent is $(4, 3)$. Then $H(x_1, y_{\sigma_2(1)}) = 2$ and $H(x_2, y_{\sigma_2(2)}) = 2$. Since the sum 4 is smaller than 6, $\sigma_2 = (2, 1)$ is the optimal permutation.

For this process, it is possible to enumerate all $k!$ permutations and find an optimal one among them. However, for a large k, such a procedure is intractable. Fortunately, Kuhn [10] proposed an efficient way to solve the problem. It is called the *Hungarian method*. Roughly speaking, starting from the initial unweighted bipartite graph with no edge, the method iteratively modifies edge weights, adds new edges into the bipartite graph, and applies the maximum matching (or minimum covering) in the resultant bipartite graph. It continues the above process until a perfect matching is found. The Hungarian method gives an optimum assignment and it can be implemented in $O(k^3)$ time [13]. Regardless of the implementation of the Hungarian method, if we see the Hungarian method as a black box, it just takes a two-dimensional array whose elements are distances between two indices and outputs an optimal assignment which gives the minimum sum.

After rearranging the second parent according to the permutation σ^*, we apply a traditional crossover. This method looks similar to normalization in k-ary encoding for grouping problems [11], but their purposes are quite different. We call this new crossover *NH(Normalized by Hamming distance)-Xover*. We expect performance improvement through this new crossover.

4. PRESERVATION OF FEASIBILITY

NH-Xover has another advantage in solving MKCP. Both of the integer encoding and the binary one have the problem of not preserving feasibility when using traditional crossover operators such as one-point or two-point crossover. For each type of encoding, an example that the feasibility is broken by one-point crossover is given in Figure 2. In (A), the index 3 is duplicated in the offspring. The solution must contain $k = 4$ column indices, but the offspring of (A) has only 3 indices because the index 3 occupies two genes. Also in (B), the solution must contain four 1s, but the offspring has only three 1s. This violates the condition of a feasible solution.

In general, a repairing step is required if we use traditional crossovers. However, repairing has the effect of mutation so that it may break the building blocks inherited from parents.

When we use NH-Xover instead of the traditional one-point crossover, repairing is not necessary at all. In the case of using integer encoding, the problem comes from the fact that the common indices in two parents can be in different positions. This fact leads that there can be two same indices in the offspring. NH-Xover naturally makes offspring preserve feasibility. If there are common indices in two parents, the indices are certainly rearranged to be in the same position by the permutation σ^*. The following theorem guarantees this fact.

THEOREM 1. *If $x_p = y_q$, then $\sigma^*(p) = q$, where σ^* is the permutation in Equation (1) and $p, q \in \{1, 2, \ldots, k\}$.*

PROOF. For the proof by contradiction we assume that $\sigma^*(p) \neq q$. There exists an index r such that $\sigma^*(r) = q$. Let σ' be the permutation obtained by exchanging the value of $\sigma^*(p)$ and $\sigma^*(r)$ from σ^*, i.e.,

$$\sigma'(i) = \begin{cases} \sigma^*(r) & \text{if } i = p, \\ \sigma^*(p) & \text{if } i = r, \\ \sigma^*(i) & \text{otherwise.} \end{cases}$$

$$\begin{aligned}
& \sum_{i=1}^{k} H(x_i, y_{\sigma'(i)}) \\
= \quad & H(x_p, y_{\sigma'(p)}) + H(x_r, y_{\sigma'(r)}) + \sum_{i \neq p, i \neq r} H(x_i, y_{\sigma'(i)}) \\
= \quad & H(x_p, y_{\sigma^*(r)}) + H(x_r, y_{\sigma^*(p)}) + \sum_{i \neq p, i \neq r} H(x_i, y_{\sigma^*(i)}) \\
= \quad & H(x_p, y_q) + H(x_r, y_{\sigma^*(p)}) + \sum_{i \neq p, i \neq r} H(x_i, y_{\sigma^*(i)}) \\
= \quad & H(x_r, y_{\sigma^*(p)}) + \sum_{i \neq p, i \neq r} H(x_i, y_{\sigma^*(i)}) \\
& (\because H(x_p, y_q) = 0 \text{ by assumption}) \\
\leq \quad & H(x_r, x_p) + H(x_p, y_{\sigma^*(p)}) + \sum_{i \neq p, i \neq r} H(x_i, y_{\sigma^*(i)}) \\
& (\because \text{triangular inequality}) \\
= \quad & H(x_r, y_q) + H(x_p, y_{\sigma^*(p)}) + \sum_{i \neq p, i \neq r} H(x_i, y_{\sigma^*(i)}) \\
= \quad & H(x_r, y_{\sigma^*(r)}) + H(x_p, y_{\sigma^*(p)}) + \sum_{i \neq p, i \neq r} H(x_i, y_{\sigma^*(i)}) \\
= \quad & \sum_{i=1}^{k} H(x_i, y_{\sigma^*(i)}).
\end{aligned}$$

This contradicts the definition of the permutation σ^* in Equation (1). Hence, $\sigma^*(p) = q$. ⊠

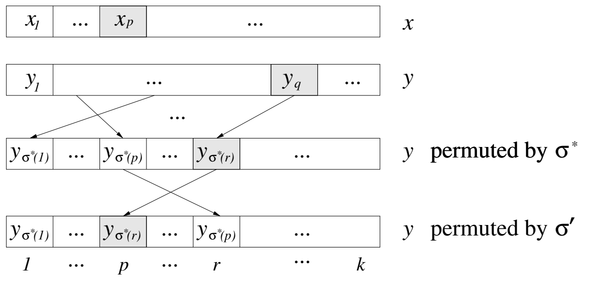


Figure 3: Proof of Theorem 1

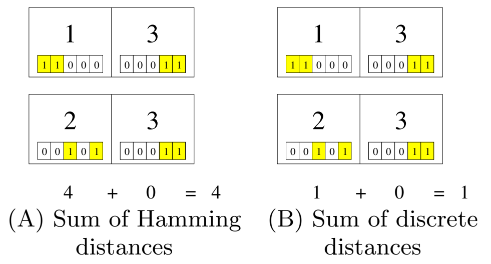


Figure 4: Hamming distance vs. discrete distance

Figure 3 helps understanding the proof of Theorem 1 more clearly.

In the proof of the above theorem, triangular inequality is mainly used. Triangular inequality is an essential property for the definition of the distance. NH-Xover uses Hamming distance, but the above theorem still holds when other distances are used instead of Hamming distance. What will happen if we use discrete distance instead of Hamming distance? Discrete distance becomes one if the two solutions is the same, and zero otherwise. It is a very simple distance, but it satisfies all the conditions for the distance including triangular inequality. Interestingly, although discrete distance seems to provide no great meaningful information between two solutions, the feasibility-preserving effect still holds when we use the distance. However, we cannot expect any other effect except aligning the same indices to the same position by using that crossover. Nevertheless, it is obvious that the feasibility-preserving property of crossover is helpful to solve the problem using genetic algorithm. We also implemented the crossover normalized by discrete distance (ND-Xover) and will compare the crossover with NH-Xover in the next section.

Figure 4 shows the difference between the normalization by Hamming distance and that by discrete distance. In the case of Hamming distance (see the left side (A)), the distance between the genes 1 and 2 at the first position is equal to the Hamming distance between the first column and the second one. In the case of discrete distance (see the right side (B)), the contents of column that each gene indicates are ignored. It is just one if the two indices are the same, and zero otherwise.

5. EXPERIMENTS

5.1 Genetic Framework

The basic evolutionary model we used is quite similar to that of the CHC [6]. CHC has been applied to various problems in the literature [2, 5, 8, 12, 14]. Let the population size be N. A collection of $N/2$ pairs is randomly composed, and crossover is then applied to each pair, generating $N/2$ offspring. Parents and newly generated offspring are ranked and the best N individuals among them are selected for the population in the next generation. In all our experiments, a population size of 400 was used. If the population has no change during $k * r * (1.0 - r)$ generations, it is reinitialized except the best one individual. Here, r is a divergence rate and we set the rate to 0.25. Our GA terminates after 500 generations and outputs the best solution found so far.

We performed the experiments by changing only crossover operator under the same genetic framework. The crossovers that were used in experiments are NH-Xover and ND-Xover.

Table 1: Test Instance Sets

Instance set	m	n	Density (%)	Number of instances
I-4	200	1000	2	10
I-5	200	2000	2	10
I-6	200	1000	5	5
I-A	300	3000	2	5
I-B	300	3000	5	5
I-C	400	4000	2	5
I-D	400	4000	5	5
I-E	500	5000	10	5
I-F	500	5000	20	5
I-G	1000	10000	2	5
I-H	1000	10000	5	5

5.2 Results

Our experiments were conducted on 11 classes of set covering test instances (a total of 65 instances) with various sizes and densities from OR-library [3]. These test instance sets were originally generated for the set covering problems, but they can also be used for the maximum k-coverage problems. The details of these test instances are given in Table 1. Note that the density is the percentage of 1s in the given matrix A in the MKCP instance.

We made our experiments on two cases that $k = 10$ and $k = 20$. As a measure of performance, we used the percentage gap $100 \times |best - output|/best$, where the *best* means the best value found by the authors and the *output* means the result by the experiment for the instance. The smaller the value is, the smaller the difference from the optimum is, i.e., the smaller, the better.

We compared NH-Xover with ND-Xover. Genetic algorithms using NH-Xover and ND-Xover were performed 30 times. Each GA uses the same population size and terminates after the same number of generations, hence GAs using NH-Xover and ND-Xover perform the same number of evaluations. We show the qualities of the best solutions and the average qualities in Table 1-4. The best, average values and %-gaps in the tables are the averages over the corresponding instance sets.[1]

Table 2 and Table 3 show the best results and the average ones with $k = 10$. NH-Xover outperformed ND-Xover for all instance sets in both tables.

[1]The best/average values and their %-gaps can sometimes be inconsistent.

Table 2: Best Results with $k = 10$

Instance set	NH-Xover Best	NH-Xover %-gap	ND-Xover Best	ND-Xover %-gap
I-4	84.4	**0.00**	83.4	0.12
I-5	88.9	**0.11**	87.0	0.78
I-6	141.0	**0.43**	137.8	0.42
I-A	130.0	**0.15**	127.6	0.47
I-B	207.8	**0.00**	202.7	0.58
I-C	161.5	**0.12**	156.9	0.62
I-D	259.4	**0.15**	253.8	0.15
I-E	424.6	**0.00**	417.2	0.89
I-F	494.6	**0.00**	491.4	0.32
I-G	329.2	**0.06**	320.4	0.36
I-H	560.4	**0.14**	550.4	0.28
Average	–	**0.11**	–	0.45

Best results and their average %-gaps from 30 trials.

Table 4: Best Results with $k = 20$

Instance set	NH-Xover Best	NH-Xover %-gap	ND-Xover Best	ND-Xover %-gap
I-4	139.00	**0.29**	133.33	0.79
I-5	142.20	**0.07**	137.23	0.77
I-6	191.00	**0.00**	186.17	0.63
I-A	203.80	**0.29**	196.50	0.68
I-B	278.80	**0.07**	272.59	0.79
I-C	254.00	0.55	246.75	**0.16**
I-D	357.60	**0.11**	351.10	0.28
I-E	496.20	**0.00**	491.97	0.20
I-F	500.00	0.00	500.00	0.00
I-G	533.00	0.63	517.94	**0.00**
I-H	812.20	**0.05**	799.73	0.49
Average	–	**0.19**	–	0.44

Best results and their average %-gaps from 30 trials.

Table 3: Average Results with $k = 10$

Instance set	NH-Xover Ave	NH-Xover %-gap	ND-Xover Ave	ND-Xover %-gap
I-4	84.3	**1.12**	83.0	1.64
I-5	88.3	**2.31**	86.1	3.23
I-6	141.0	**2.70**	137.1	3.15
I-A	129.6	**2.94**	126.1	3.11
I-B	206.6	**2.44**	200.4	3.57
I-C	160.4	**2.79**	156.2	3.23
I-D	259.4	**2.31**	251.7	3.12
I-E	420.8	**1.75**	413.1	2.70
I-F	493.0	**0.66**	488.5	1.13
I-G	328.2	**2.73**	319.0	3.15
I-H	559.6	**1.92**	547.6	2.43
Average	–	**2.15**	–	2.77

Average results and their average %-gaps from 30 trials.

Table 5: Average Results with $k = 20$

Instance set	NH-Xover Ave	NH-Xover %-gap	ND-Xover Ave	ND-Xover %-gap
I-4	137.30	**3.65**	132.80	4.04
I-5	141.20	**3.56**	136.22	4.27
I-6	189.80	**2.53**	185.37	2.94
I-A	203.00	3.85	196.53	**3.84**
I-B	276.80	**2.30**	270.56	3.02
I-C	255.00	3.38	247.08	**3.25**
I-D	357.00	**1.93**	349.77	2.49
I-E	495.20	**0.85**	490.97	1.05
I-F	500.00	0.00	500.00	0.00
I-G	536.40	3.44	518.78	**3.28**
I-H	808.60	**1.58**	796.04	2.04
Average	–	**2.46**	–	2.75

Average results and their average %-gaps from 30 trials.

Table 4 and Table 5 show the best results and the average ones with $k = 20$. For I-F, the solutions found by NH-Xover and ND-Xover always cover all the rows. This is because the density of I-F is too high. So, just with a few number of columns we could easily cover all the rows.

In the results with $k = 20$, NH-Xover could not dominate ND-Xover only for three instance sets I-A, I-C and I-G among 11 sets. For I-A, NH-Xover was not good for the average results, but better for the best results. We guess that larger search space than in the case that $k = 10$ may influence to this inconsistency. However, for both of the best and average, average values (see the rows named "Average" in tables) of %-gaps of NH-Xover over all instance sets were better than those of ND-Xover. We can conclude that NH-Xover shows better performance than ND-Xover from the experimental results.

6. CONCLUDING REMARKS

In this paper we introduced the maximum k-coverage problem and analyzed the property of the problem. Based on the observed property, we proposed a new crossover that produces offspring which have as few common elements as possible. The proposed crossover is easily implemented using Hungarian method and has an additional good property of not being necessary to repair for feasibility. We made experiments and compared the results with another feasibility-preserving crossover. Our new crossover showed better performance for almost all instances.

It is expected that the proposed crossover will show better performance when compared with the genetic algorithms using traditional crossovers combined with repair operation since it is hard to preserve the inherent good building blocks. More investigation about such comparison and analysis through experiments are left for further study.

7. REFERENCES

[1] U. Aickelin. An indirect genetic algorithm for set covering problems. *Journal of the Operational Research Society*, 53(10):1118–1126, 2002.

[2] E. Alba, G. Luque, and L. Araujo. Natural language tagging with genetic algorithms. *Information Processing Letters*, 100(5):173–182, 2006.

[3] J. E. Beasley. OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[4] J. E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94:392–404, 1996.

[5] O. Cordón, S. Damasb, and J. Santamaría. Feature-based image registration by means of the CHC evolutionary algorithm. *Image and Vision Computing*, 24(5):525–533, 2006.

[6] L. J. Eshelman. The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, 1991.

[7] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[8] C. Guerra-Salcedo and D. Whitley. Genetic search for feature subset selection: A comparison between CHC and GENESIS. In *Proceedings of the Third Annual Conference on Genetic Programming*, pages 504–509. Morgan Kaufmann, 1998.

[9] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615–627, 1998.

[10] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[11] A. Moraglio, Y.-H. Kim, Y. Yoon, and B.-R. Moon. Geometric crossovers for multiway graph partitioning. *Evolutionary Computation*, 15(4):445–474, 2007.

[12] A. J. Nebro, E. Alba, G. Molina, F. Chicano, F. Luna, and J. J. Durillo. Optimal antenna placement using a new multi-objective CHC algorithm. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 876–883. ACM, 2007.

[13] C. H. Papadimitriou and K.Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, 1955.

[14] S. Tsutsui and D. E. Goldberg. Search space boundary extension method in real-coded genetic algorithms. *Information Sciences*, 133(3-4):229–247, 2001.