Multi-objective Combinatorial Optimisation with Coincidence Algorithm

Warin Wattanapornprom, Panuwat Olanviwitchai, Parames Chutima, and Prabhas Chongstitvatana.

Abstract — **Most optimization algorithms that use probabilistic models focus on extracting the information from good solutions found in the population. A selection method discards the below-average solutions. They do not contribute any information to be used to update the models. This work proposes a new algorithm, *Combinatorial Optimization with Coincidence* (COIN) that makes use of both good and not-good solutions. A Generator represents a probabilistic model of the required solution, is used to sample candidate solutions. Reward and punishment schemes are incorporated in updating the generator. The updating values are defined by selecting the good and not-good solutions. It has been observed that the not-good solutions contribute to avoid producing the bad solutions. The multi-objective version of COIN is also introduced. Several benchmarks of multi-objective problems of real world industrial applications are reported.**

I. Introduction

"GOD does not play dice, coincidence is god's way of remaining anonymous." – Albert Einstein has left challenge to solve the mysteries of the coincidences in the universe. EDA algorithms try to extract the knowledge found in the solutions in order to reproduce the better solution. According to Minsky [1], negative knowledge hidden in seemingly positive knowledge plays a controlling role in diverse areas including expert system, emotion, and search. Combinatorial Optimization with Coincidence (COIN) algorithm adopts the negative knowledge to enhance the search by avoiding the reproduction of undesired solutions. This paper, we introduce the COIN algorithm which is invented to solve single-objective problems, and then present the adaptation of COIN in multi-objective problems.

The structure of the paper is as follows. The related works are discussed in Section II. The proposed algorithm, Combinatorial Optimization with Coincidence, is explained in Section III. Section IV introduces the multi-objective version of COIN. The experiments are reported and the results are discussed in Section V. Finally, Section VI concludes the work.

W. Wattanapornprom and P. Chongstitvatana are with the Department of Computer Engineering, Faculty of Engineering Chulalongkorn University, Thailand (e-mail: yongkrub@gmail.com and prabhas@chula.ac.th and).

P. Olanviwitchai and P. Chutima are with the Department of Industrial Engineering, Faculty of Engineering Chulalongkorn University, Thailand (e-mail: househomeme_1234@hotmail.com and parames.c@chula.ac.th).

II. Related Works

There are many algorithms that use the second order statistic and considered to be the algorithms in Bivariate Dependency class in Estimation of Distribution Algorithms. These algorithms take dependencies between pairs of variables into account. The algorithms in this class include MIMIC [2], COMIT [3] and BMDA [4].

A. MIMIC

One of the most famous algorithms in the bivariate dependency class is MIMIC (Mutual Information Maximizing Input Clustering), proposed by De Bonet et al. in 1997. A greedy algorithm that searches in each generation for the best permutation between the variables in order to find the probability distribution, $p_l^\pi(x)$ that is closest to the empirical distribution of the set of selected points when using the Kullback-Leibler distance, where

$$p_l^\pi(x) = p_l(x_{i1}|x_{i2}) * p_l(x_{i2}|x_{i3}) * \dots * p_l(x_{in-1}|x_{in})p_l(x_{in}) \quad \textbf{(1)}$$

and $\pi = (i_1, i_2, \dots, i_n)$ denotes a permutation of the indexes $1, 2, \dots, n$.

This algorithm avoids searching through all $n!$ permutations by selecting X_{in} as the variable with the smallest estimated entropy then, at every following step, to pick the variable from the set of variables not chosen so far whose average conditional entropy with respect to the variable selected in the previous step is the smallest.

B. COMIT

The dependency tree version of PBIL [5] is later called COMIT(Combining Optimizers with Mutual Information Tree). The algorithm was proposed by Baluja and Davies[3][6]. The algorithm constructs dependency trees and incrementally learns from the good seen solution so far using the algorithm proposed by Chow and Liu [7].

C. BMDA

Pelikan and Mühlenbein proposed an algorithm call BMDA (Bivariated Marginal Distribution Algorithm) using factorization of the joint probability distribution. It is based on the construction of a dependency graph, which is always acyclic but does not have to be a connected graph. BMDA adds the dependency to the graph using the greatest dependency between any of the previously incorporated variables and the set of not yet added variables.

These three algorithms are based on the model construction of the knowledge. None of them uses the below average solution in learning the model. Moreover there is no evidence of applying them in multi-objective problems.

D. EBCOAs

In 2004, T.Miquelez et al. proposes the use of classifiers in the form of Bayesian networks for the optimisation. The new algorithm is called Evolutionary Bayesian Classifier-based Optimization Algorithm[8] (EBCOA). This algorithm is the first EDA that bad instances are considered for the learning procedure so that the algorithm avoid producing less fitted individuals when the generation progresses. This method is also applied to continuous optimisation domains[9].

In the multi-objective version of COIN we adopt the non-dominated sorting of NSGA-II [10] in order to select the candidates to give the reward and punishment to the algorithm.

E. NSGA-II

Non-dominated sorting genetic algorithm II (NSGA-II) is one of the most popular genetic algorithms in recent years. It has the ability to find multiple Pareto-optimal solutions in one single run. In NSGA-II, the population is sorted according to the level of non-domination. The diversity among non-dominated solutions is maintained using a measure of density of solution in the neighborhood. NSGA-II is able to find much better widespread solutions and better convergence near the true Pareto-optimal front in most problems.

Non-dominated sorting of NSGA-II has been widely used in many multi-objective EDAs. The algorithm which adopt this technique are multi-objective hBOA[11] and RM-MEDA[12]

F. mohBOA

The multi-objective version of the famous algorithm call hBOA proposed by M.Peligan et al. combine hBOA, NSGA-II and clustering in the objective space. The algorithm is shown to scale up well on a number of problems on which standard multiobjective evolutionary algorithms perform poorly.

G. RM-MEDA

A regularity model-based multi-objective EDA proposed by Q. Zhang et al. This algorithm models a promising area in the decision space by a probability distribution whose centroid is a (m-1)-D piecewise continuous manifold. The localprincipal component analysis algorithm is used for building such a model. A non-dominated sorting-based selection is used for choosing solution for the next generation.

III. Combinatorial Optimization With Coincidence

The proposed algorithm is explained in this section. The main idea is to allow learning from the below average solutions as well as the traditional learning from the good solutions. The coincidence found in a situation should be able to statistically describe the chance of the situation to be happening whether the situation is good or bad. Thus the learning of the coincidence found in the bad solutions should be used to avoid the bad situation as well.

The COIN algorithm uses a generator to generate the population. The population is evaluated the same way as traditional evolutionary algorithms. However, COIN uses both good and not-good (to be defined precisely later) solutions to update the generator. The generator is initialized so that it can generate a random tree with equal probability to be in any configuration.

```
Step 1. Initialize the generator.
Step 2. Generate the population using the generator.
Step 3. Evaluate the population.
Step 4. Select the candidates. There are two methods:
    a. Uniform selection: select the top and bottom $c$ percent.
    b. Adaptive selection: select the above and below the
       average $\pm 2\sigma$
Step 5. For each joint probability $h(x_i|x_j)$, update the generator
    according to the reward and punishment :-
    $X_{i,j}(t+1) = X_{i,j}(t) + \frac{k}{(n-1)}\left(r_{i,j}(t+1) - p_{i,j}(t+1)\right) + \frac{k}{(n-1)^2}\left(\sum_{j=1}^{n} p_{i,j}(t+1) - \sum_{j=1}^{n} r_{i,j}(t+1)\right)$
    where $X_{i,j}$ denotes the joint probability $h(x_i|x_j)$, k is the
    learning coefficient, $r_{i,j}$ denotes the number of coincidence
    $X_i,X_j$ found in the good solutions, $p_{i,j}$ denotes the number of
    coincidence $X_i,X_j$ found in the not-good solutions, $n$ is the size
    of the problem.
Step 6. Repeat Step 2. Until the terminate condition is met.
```

Fig. 1. Pseudo code for COIN

The algorithm searches from a fully connected tree initially and incrementally strengthening or weakening the connections. As generation progressed the probabilities in the generator are increased or decreased according to the mutual information found in the good or not-good solutions.

The COIN algorithm is shown in Fig. 1. It begins by initializing the generator then the population is sampling from the generator. The generator is updated by each of the coincidence found in the selected good and not-good candidates. The generating, evaluation and updating steps are repeated until terminate.

A. The Generator

The generator of COIN is a matrix of size $n \times n$ containing multiple dependency trees. Each row denotes a dependency tree, where each of the members in the row is the joint probability $h(X_i|X_j)$. X_i indicates the row of the matrix, while

X_j indicates the column. A coincidence is denoted by X_i,X_j of the event X_i follows by the event X_j. The coordinate $X_{i,j}$ indicates the joint probability $h(X_i|X_j)$.

Initializing the generator

The generator is initialized so that each of the joint probabilities $h(X_i|X_j)$ except the $X_{i,j}$ equal to $\frac{1}{(n-1)}$. This initialization represents the uniform distribution of each joint probability in the dependency tree.

Generating the population

The generator generates each candidate the same way MIMIC and COMIT do. Given a dependency tree, π, we can sample from it as follow:

1. Begin at any node, X_i in the tree. This is the root node. Pick its value according to its empirical probability $h(X_i)$.
2. Perform a depth-first traversal of the tree from the root. For each X_i, choose its value according to the empirical probability $h(X_i|X_j)$.

Updating the generator

When each coincidence X_c,X_r is found in good candidates it is used to update the joint probability $h(X_c|X_r)$ by rewarding the coordinate $X_{c,r}$ in the matrix. The coordinate $X_{c,r}$ is rewarded by gathering the probability $\frac{k}{(n-1)^2}$ from the set $X_{c,j}$ where j range from 1 to n and k is the coefficient denoting the learning step, and $r_{c,r}$ is the total number of coincidence $X_{c,r}$ counted from the good solution. The reward equation of $X_{c,r}$ is

$$X_{c,r}(t+1) = X_{c,r}(t) + \frac{k}{(n-1)}\left(r_{c,r}(t+1)\right) - \frac{k}{(n-1)^2}\left(\sum_{j=1}^{n} r_{c,j}(t+1)\right) \quad \textbf{(2)}$$

Contrary to the rewarding, when each coincidence X_c,X_p is found in a not-good candidate it is used to update the joint probability $h(X_c|X_p)$ by punishing the $X_{c,p}$ in the matrix. The $X_{c,p}$ is punished by scattering its own probability $\frac{k}{(n-1)^2}$ to every member in the set $X_{c,j}$ where j range from 1 to n and k is the coefficient denoting the learning step, and $p_{c,p}$ is the total number of coincidence $X_{c,p}$ counted from the not-good solution. The punishment equation of $X_{c,p}$ is

$$X_{c,p}(t+1) = X_{c,p}(t) - \frac{k}{(n-1)}\left(p_{c,p}(t+1)\right) + \frac{k}{(n-1)^2}\left(\sum_{j=1}^{n} p_{c,j}(t+1)\right) \quad \textbf{(3)}$$

Combining reward and punishment when a coincidence $X_{c1,c2}$ is found in both good and not-good solutions we will get:

$$X_{c1,c2}(t+1) = X_{c1,c2}(t) + \frac{k}{(n-1)}\left(r_{c1,c2}(t+1) - p_{c1,c2}(t+1)\right) + \frac{k}{(n-1)^2}\left(\sum_{j=1}^{n} p_{c1,j}(t+1) - \sum_{j=1}^{n} r_{c1,j}(t+1)\right) \quad \textbf{(4)}$$

There is some constraint in updating the generator. Since the joint probability is updated by increasing or decreasing by a constant rate, a joint probability must not become negative. Moreover when a joint probability is increased up to a point, it will dominate such that no other branch will be explored. Therefore we need to maintain the probability value by disallow the punishment if it would decrease the probability down below 1/10 of the initial value and disallow the reward when the joint probability exceeds 8/10 of overall probability.

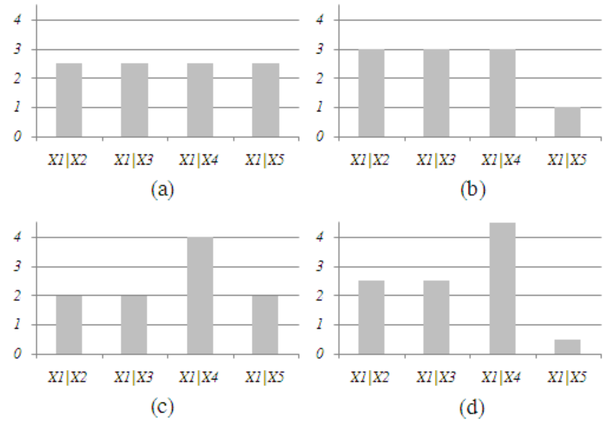

Fig. 2. the effect of the rewards and punishment to a dependency tree

The effect of reward and punishment is shown in Fig. 2. The figure 2 (a) shows the dependency $X_1|X_j$ after it has just been initialized. Each of the joint probabilities $X_1|X_2$, $X_1|X_3$, $X_1|X_4$, and $X_1|X_5$ is initialized as 0.25 uniformly. The figure 2 (b) illustrates the punishment of $X_1|X_5$ when the coincidence $X_1|X_5$ is found in a not-good solution. If the learning step k is equal to 0.2, $X_1|X_5$ has to scatter its joint probabilities to every node under X_1. As in this case it has to donate 0.05 each to $X_1|X_2$, $X_1|X_3$, $X_1|X_4$, including itself. The figure 2 (c) presents the way to reward $X_1|X_4$ by gathering joint probabilities of value 0.05 from each of the members. The figure 3.2 (d) shows a result of reward $X_1|X_4$ and punishment $X_1|X_5$ at the same time.

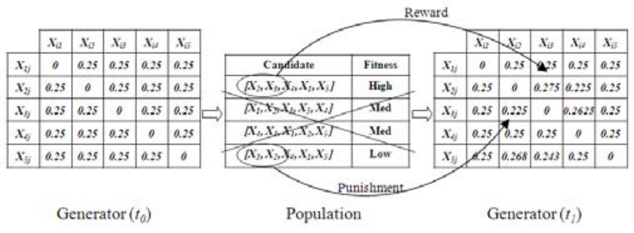

Fig.3. Updating the generator $k=0.1$

Fig. 3 illustrates the process of initializing the generator, generating the first population, selection of good and not-good candidates and finally updating the generator using the selected candidates. The generator is initialized so that each node of the dependency is equally to 0.25. The population is generated from the initiated generator. The candidates are sorted and classified into three classes: high fitness, medium fitness, and low fitness. The high fitness candidates are considered to be the good solutions while the low fitness candidates are considered to be the not-good solutions in the population.

As seen in the fig. 3, the candidate [X_2, X_3, X_4, X_1, X_5] is classified as a good solution and later is used to update the generator as rewards. The candidate [X_3, X_2, X_4, X_1, X_5] is classified as a not-good solution and is used to punish the generator in the opposite way. Since the coincidences X_4,X_1 and X_1,X_5 are found in both good and not-good solutions, they are counted as a one-time reward and a one-time punishment so the row X_{1j} and row X_{4j} remain unchanged. While the coincidences X_5,X_2, X_2,X_3 and X_3,X_4 are considered to be coincidences found in the good solutions, hence these coincidences are used to update the row X_5, X_2 and X_3. The coincidences X_5,X_3; X_3,X_2 and X_2,X_4 are used to punish the generator as they are found in the not-good solutions.

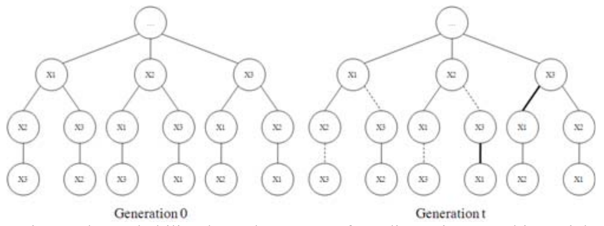

Fig. 4. The probability dependency tree of a 3 dimensions combinatorial problem

Fig. 4 represents the search space of a 3 dimensional combinatorial problem. In the generation 0, all joint probabilities are equal. As the generation progresses, the joint probabilities are increased or decreased. It can be seen that some of the connections are weaken. Whereas some of the connections are strengthen as the coincidences are found in the good solutions.

B. Selection

Two selection methods are considered: a uniform method selects from the top and bottom *c* percent of the population, an adaptive method selects from the population above and below the average in the band of two standard deviations.

In the adaptive selection, if the population contains more good candidates, the selector will select more not-good solutions rather than good solutions. Conversely the selector would select more of the good solution when the overall candidates in the population are not-good. This mechanism maintains the diversity among the candidates. The punishment can spread out the population when the algorithm seems to converge as the punishment of a joint probability forces to scatter its value to the others.

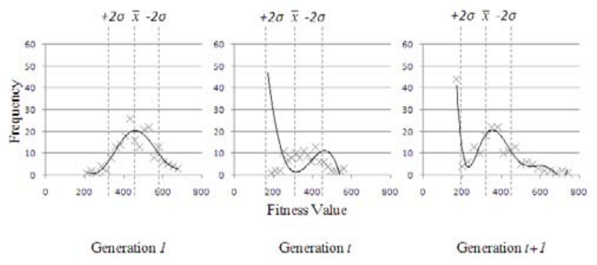

Fig. 5. The population distribution in a 10 cities TSP problem

A solid example can be seen in the figure 5. It illustrates the distribution of the population in generation 1, *t* and *t+1*. In the generation *t* the diversity is low due to the high frequency of the candidates with the same quality. Thus more of the candidates are used for punishment. The result shows the distribution of the population after the punishment. Some of the candidates even go beyond the first generation. This indicates the migration away from the local optima.

C. Computational Cost and Space Issues

If the problem size is *n*, and *m* candidates in each generation, the computational cost and space complexity are as follow:

1. Generating the population requires time $O(mn^2)$ and space $O(mn)$
2. Sorting the population requites time $O(m \log m)$
3. The generator require space $O(n^2)$
4. Updating the generator require time $O(mn^2)$

IV. Multi-Objective COIN

The multi-objective version of coin is slightly different from the single-objective COIN in the selection method. We adopt the non-dominate sorting of NSGA as the way to select the population used in updating the generator. Again we use the not-good solutions to train the generator. The not-good solutions are defined different from the single-objective COIN. They are obtained from the non-dominated frontier of the opposite side of the objectives we are optimizing. The number of the selected candidates depends on the rank of the frontiers. The first n^{th} ranks of the not-good non-dominated solutions are not the same as the last n^{th} ranks of the good non-dominated solutions. This paper does not discuss many other alternative definitions of the not-good solutions in multi-objective problems.

V. Experiments

To measure the performance of COIN, we perform several benchmarks on TSP problems and compare them to the experiment of Robles and Larrañaga [13]. We aim to measure the performance of the algorithm in two main aspects: quality of the results and the number of function evaluations. This paper we show only the result of the well known Gröstel24, which can be obtained from the TSPLIB [22].

The experiment of Robles and Larrañaga use both of the discrete and continuous EDAs in the following learning methods: UMDA [13], MIMIC, TREE [7], EBNA [15], $UMDA_c$, and $MIMIC_c$, EGNA [16] and EMNA [17]. Moreover we compare the results with GA in the literature of Larrañaga [18] in 1999 which uses GENITOR [19] algorithm. For each of the combinations shown in the experiment, we perform 10 runs and average the results.

However the TSP coding for continuous EDAs in the original literature uses real numbers which later sorted into the ordering path based on these numbers, is known to be a poor alternative coding compared with path representations based on permutations. Thus the results from continuous EDAs are not compare in this paper due to the use of different representations.

TABLE I
TOUR LENGTH FOR THE GRÖSTEL24 PROBLEM

Algorithm	Population & Local Optimization							
	500-without		500-with		1000-without		1000-with	
	Best	Aver	Best	Aver	Best	Aver	Best	Aver
GA-ER*	1272	1272						
GA-OX2*	1300	1367						
UMDA	1339	1495	1272	1272	1329	1496	1272	1272
MIMIC	1391	1486	1272	1272	1328	1451	1272	1272
TREE	1413	1486	1272	1272	1429	1442	1272	1272
EBNA	1431	1528	1272	1272	1329	1439	1272	1272
COIN *unif*	1272	1280			1272	1278		
COIN *adpt*	1272	1272			1272	1272		

* Size of population 200, mutation used SM
unif denotes uniform selection with learning step $k = 0.1$
adpt denotes adaptive selection with learning step $k = 0.1$
Optimum 1272

Table 1 shows the best results and average results obtained for each of population size, with and without local optimization and learning type of EDAs. The table also shows results obtained for the GA using the crossover operators ER and OX2. The results show that COIN with adaptive selection can find the optimum of Gröstel24 without the need of local optimizer and it is competitive with all the EDAs in the experiment.

TABLE II
NUMBER OF GENERATIONS FOR THE GRÖSTEL24 PROBLEM

Algorithm	Population & Local Optimization			
	500-without	500-with	1000-without	1000-with
	Gen	Gen	Gen	Gen
UMDA	75	19	78	12
MIMIC	47	4	58	4
TREE	37	4	46	2
EBNA	72	16	79	7
COIN	67		48	

Table 2 illustrates the number of the generation used to find the solutions. Again the result shows that COIN is competitive in the larger population size.

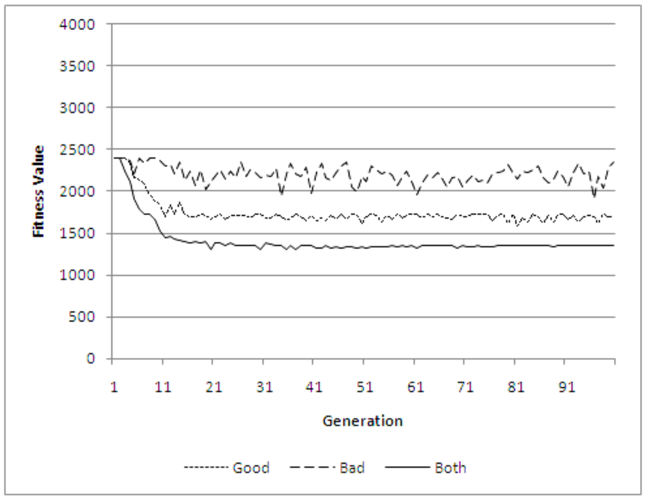

Fig. 6. the best population generated from the generator.

Fig. 6 shows the convergence of the Gröstel24 problem using only good solutions, only not-good solutions and using both types. It shows that the use of both good and not-good solutions outperform the use of only good or bad solutions. Learning from not-good solutions creates more variety amongst the best results but retaining the average results.

In this problem, the adaptive selection outperforms the uniform selection. Surprisingly the punishments counted from the experiments are 2/3 of the overall selection.

To study the effect of reward and punishment of good and bad instances in multi-objective problems, the multi-objective COIN is tested in a multi-objective TSP problem. We setup an experiment using kroa100 and krob100 as a bi-objective 100 tours TSP problem obtained from the TSPLIB. The population size we used in the experiment is 500 and the learning step k is equal to 0.1. Then we took some snapshots at the number of generations equal to 100 and 500 respectively. The behavior of the algorithm can be seen in Fig. 7 and Fig 8.

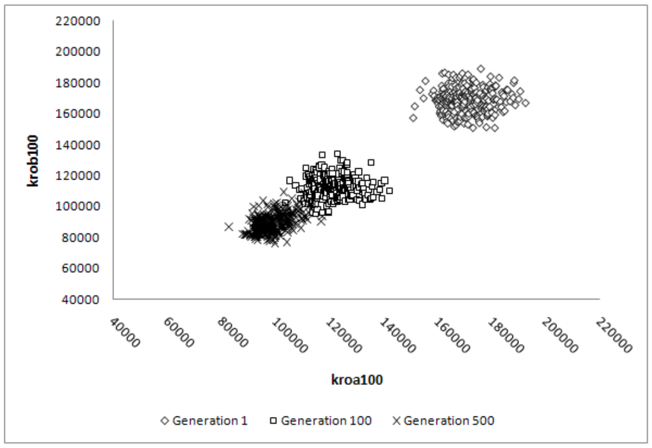

Fig. 7. the population clouds in a bi-objective kroa/b100 TSP.

Fig. 7 shows the population in the generation 1, 100 and 500 respectively. The population migrates towards the optimum fitness area as the generation progresses.

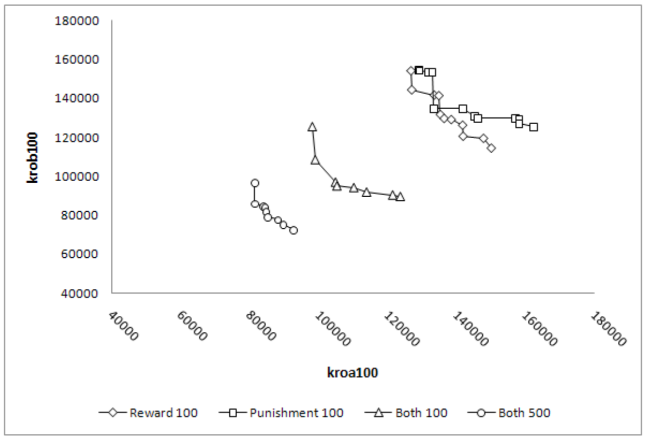

Fig. 8. the parato frontier obtained from different generation and updating method in a bi-objective kroa/b100 TSP.

Fig. 8 shows the effect of COIN algorithm in using only rewarding and only punishment compared with using both. Two curves at the upper-right hand corner are the result from using only reward or punishment for 500 generations. Contrast this with the rest of the curves in the lower-left corner which use both reward and punishment together for 100 and 500 generations. The use of both reward and punishment for just 100 generations outperforms the result from using only either one for 500 generations.

To measure the performance of multiple-objective COIN, we perform several benchmarks on real world applications. The applications we use in the experiments include multi-objective balancing problems on mixed-model U-shaped assembly lines in JIT production systems.

U-shaped assembly line balancing is considered to be NP-Hard This kind of assembly line has advantage in reduction of the waste walking time to switch from workstation to workstation, thus enhance reduction of employee and cost.

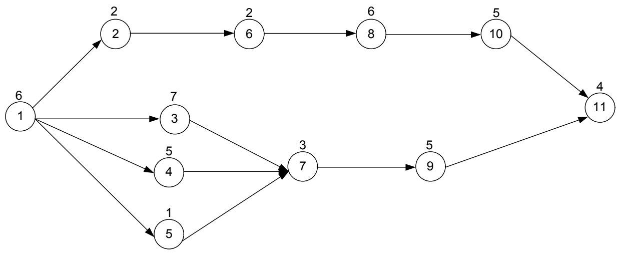

Fig. 9. The precedence diagram with assembly network (Jackson 1956).

Fig. 9 illustrates the Jackson's problem[20] with 11 tasks. Given each workstation $ws = 1$ to m , number of tasks $i = 1$ to n , each task uses time t_i. The total time used in the fig. 9 is equal to 10 while it used up to 14 if the line is straight.

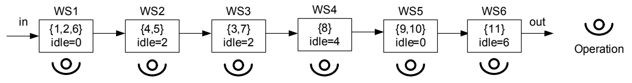

Fig. 10. Completed line assignment for straight assembly line.

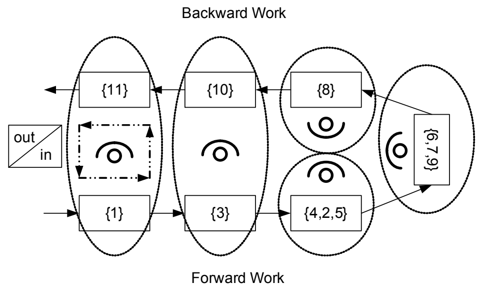

Fig. 11. Completed line assignment for U-shaped assembly line.

After fitting the tasks and workstations in to the assembly line, we can see that the U-shaped assembly line in the Fig. 11 use one less workstation than the straight line in the Fig. 10. As the employee who processes the task number 1 can be shared with the task number 11.

We carry three experiments based on the work of Hwang and Katayama [21] in three objectives:

Given m is number of workstation
SN_k is number of relatedness of work in the workstation k
S_{max} is total maximum time in the workstation
S_k is total time in the workstation k

, the three objectives are:

1. To minimize the number of workstation.

$$f_1(X) = Min\ m \qquad (5)$$

2. To minimize the relatedness of the workstations.

$$f_2(X) = m - m \Big/ \sum_{k=1}^{m} SN_k \qquad (6)$$

3. To minimize the distribution of workload in each station.

$$f_3(X) = Min \sqrt{\sum_{k=1}^{m} \left(S_{\max} - S_k\right)^2 \Big/ m} \qquad (7)$$

This experiment compares NSGA-II and COIN in four aspects:

1. Convergence to the Pareto-optimal set.
2. Spread to the Pareto-optimal set.
3. Ratio of non-dominated solution.
4. Processing time.

TABLE III
PROBLEM SETS OF HWANG AND KATAYAMA

Problem set	Product	Task	Time)sec)	Density Network
Thomopolous (1970)	3	19	2	0.122807
Kim(2006)	4	61	10	0.036066
Arcus(1963)	5	111	10,000	0.028337

The density network shown in Table III indicates the relationship of the task. If the density network has high value (limited to 1), the possibility of assigning task to workstation is low. In contrast, the possibility of assigning task to workstation is high when the density network is low.

We perform the experiment using Matlab 7.0. The test environment is on Intel Core2 Duo 2.00 GHz with 1.49 GB of RAM. According to the Table IV, COIN has higher convergence rate than the NSGA-II. NSGA-II give more spread solution in the Pareto-optimal set, but the spread of the solution in this experiment has less significant due to the ratio of non dominated solution of COIN. It is equal to 1 in every test set as none of the NSGA-II's solution can dominate the COIN's solution. Fig. 12 compares only the Pareto-optimal solution for two objectives since the number of workstation in every problems are equal. Moreover, in terms of real CPU time, the multi-objective COIN is much faster than NSGA-II. The total processing time of NSGA-II in Thomopolous (19 tasks), Kim (61 tasks) and Arcus (111 tasks) are 124, 347 and 735minites , while COIN uses only 3, 15, and 40 minutes respectively.

TABLE IV
RESULT OF THE EXPERIMENT IN HWANG AND KATAYAMA'S PROBLEMS

Benchmarking	Problems and Algorithms					
	Thomopolous (19 task)		Kim (61 task)		Arcus (111 task)	
	NSGA-II	COIN	NSGA-II	COIN	NSGA-II	COIN
Convergence	0.295	0	0.847	0	0.189	0
Spread	0.566	0.523	0.742	0.774	0.485	0.710
Ratio of solution	0	1	0	1	0	1
Time (min)	124	3	347	15	735	40

Population size = 100, Generation = 200
NSGA-II: Crossover probability = 0.7, Mutation probability = 0.3
COIN: $k = 0.1$

VI. CONCLUSION

In this paper, we propose a new algorithm named COIN and present an extension of COIN in multiple objective problems. The proposed algorithm learns the joint probability of the coincidence of candidates. This knowledge is used in the generator that generates solutions. The generator represents a dependency tree, where each of the edge represents the joint probability of the coincidence X_i, X_j. The algorithm has been benchmarked against several algorithms. In a Gröstel24 TSP problem, the results show that the proposed algorithm is competitive with other discrete EDAs. We perform several benchmarks on real world industrial applications. The result shows that the multiple objective version of COIN outperforms NSGA-II in every test set.

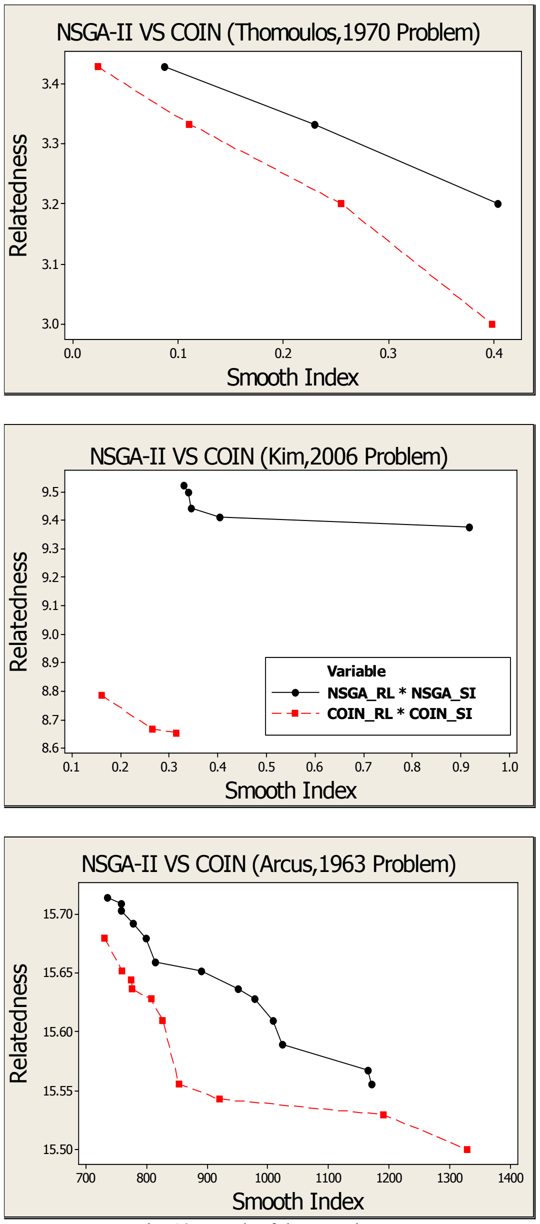

Fig. 12. Result of the experiment.

References

[1] Minsky, M., Negative Expertise, International Journal of Expert Systems 7(1), (1994)

[2] De Bonet, J.S., Isbell, C.L., and Viola, P.: MIMIC: Finding Optima by Estimating Probability Densities. Advance in Neural Information Processing Systems, volume 9. (1997)

[3] Baluja, S. and Davies, S.: Combining Multiple Optimization Runs with Optimal Dependency Trees. Technical Report CMU-CS-97-157, Carnegie Melon University (1997)

[4] Pelikan, M. and Mühlenbein, H.: The Bivariate Marginal Distribution Algorithm. Advance in Soft Computing-Engineering Design and Manufacturing, pages 521-535 (1999)

[5] Baluja, S.: Population Based Incremental Learning: A Method for Integrating Genetic Search-Based Function Optimization and Competitive Learning. Technical Report CMU-CS-94-163, Carnegie Mellon University (1994)

[6] Baluja, S. and Davies, S.: Fast Probabilistic Modeling for Combinatorial Optimization. In AAAI-98 (1998)

[7] Chow, C. and Liu, C.: Approximating Discrete Probability Distributions with Dependency trees. IEEE Transactions on Information Theory, 14:462-467. (1967)

[8] T. Miquélez, E. Bengoetxea, P. Larrañaga Evolutionary computation based on Bayesian classifiers. International. Journal of Applied Mathematics and Computer Science, 14(3), 101-115. (2004)

[9] T. Miquelez, E. Bengoetxea, A. Mendiburu, P. Larrañaga Combining Bayesian classifiers and estimation of distribution algorithms for optimization in continuous domains. Connection Science, 19(4), 297-319. (2007)

[10] K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6, 2(April 2002) 182-197. (2002)

[11] M. Pelikan, K. Sastry, Goldberg, D.E.: Multiobjective hBOA, Clustering and Scalability, IlliGAL Report No. 2005005, (2005)

[12] Q. Zhang, A. Zhou, and Y. Jin, RM-MEDA: A Regularity Model-Based Multiobjective Estimation of Distribution Algorithm, IEEE Trans. Evoutionaryl.Computation., vol. 21, no. 1, pp. 41–63, (2008)

[13] Robles V. and Larrañaga P.:Solving the Traveling Salesman Problem with EDAs. In Estimation of Distribution Algorithm: A New Tool for Evolutionary Computation (2002)

[14] Mühlenbein, H.: The Equation for Response to Selection and Its Use for Prediction. Evolutionary Computation, 5:303-346. (1998)

[15] Etxeberria, R. and Larranga, P.: Global Optimization with Bayesian Networks. In II Symposium on Artificial Intelligence. CIMAF99. Special Session on Distributions and Evolutionary Optimization, pages 322-339. (1999)

[16] Larrañaga, P., Etxeberria, R., Lozano, J. A., and Pena, J.M.: Optimization by Learning and Simulation of Bayesian and Gaussian Networks. Technical Report. KZZA-IK-4-99, Department of Computer Science and Artificial Intelligence, University of the Basque Country. (1999)

[17] Larrañaga, P., Lozano, J. A., and Bengoetxea, E.: Estimation of Distribution Algorithms Based on Multivariate Normal and Gaussian Networks. Technical Report KZZA-IK-1-01, Department of Computer Science and Artificial Intelligence, University of the Basque Country. (2001)

[18] Larrañaga, P., Kujipers, C.M. H., Murga, R.H., Inza, I., and Dizdarevic, S.: Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. Artificial Intelligence Review, 13:129-170. (1999)

[19] Whitley, D., Starkweather, D., and Fuquay, D.: Scheduling Problems and Travelling Salesman: The Genetic Edge Recombination Operator. Proceedings of the International Joint Conference on Artificial Intelligence, pages 133-140. Morgan Kaufmann Publishers (1989)

[20] Jackson, J.R.: A Computing Procedure for a Line Balancing Problem. Management *Science*. Vol. 2,No.3: pp.261-271. (1956)

[21] Hwang, R. and H. Katayama, A Multi-Decision Genetic Approach for Workload Balancing of Mixed-Model U-shaped Assembly Line Systems, International Journal of Production Research: 1-26 (2007)

[22] TSPLIB,http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ (retrieving on August 18^{th,} 2008)