Coincidence Algorithm for Combinatorial Optimisation and Its Applications

Prabhas Chongstitvatana[1], **Warin Wattanapornprom**[1], **Panuwat Olanviwitchai**[2], **Ronnachai Sirovetnukul**[3], **Noppon Kampirom**[2] **and Parames Chutima**[2]
[1]Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, email: prabhas@chula.ac.th
[2]Department of Industrial Engineering, Faculty of Engineering, Chulalongkorn University
[3]Department of Industrial Engineering, Faculty of Engineering, Mahidol University

Abstract

This article introduces Coincidence Algorithm (COIN). It is an algorithm in the class of Evolutionary Algorithm, specialised to solve combinatorial problems. COIN belongs to a subgroup of Evolutionary Algorithms which makes use of models to generate solutions instead of searching traditional population. The model of COIN is a joint probability table of adjacent events (coincidence) derived from the population of candidate solutions. As COIN is especially good at Combinatorial Optimisation, it has been applied successfully to many industrial engineering problems. A full account of COIN and its comparison to contemporary algorithms are presented. The application of COIN to real world problems are illustrated to show examples how COIN can be used.

Keywords: coincidence algorithm, evolutionary computation, estimation of distribution algorithm, combinatorial problems

1. Introduction

Combinatorial optimisation is the optimisation where the domains of feasible solutions are discrete. Examples of this domain are traveling salesman problem, minimum spanning tree problem, set-covering problem, knapsack problem, etc. It is also related to constraint satisfaction problem, such as N-Queen puzzle. For a reasonable problem size, exhaustive search is not feasible. Any searching method can not guarantee to find an optimal solution. Combinatorial optimisation has many applications for operational research.

This article is a cumulative effort of our research team to solve combinatorial problems in the past two years. It is a summary of a number of our previous works [1][2][3]. This article presents Coincidence algorithm in fine details and illustrates two real-world industrial problems.

2. Coincidence Algorithm

2.1 General

COIN belongs to a subgroup of Evolutionary Algorithms that makes use of models to generate solutions. This group of algorithms is called “Estimation of Distribution Algorithms” [4] and also “Competent Genetic Algorithms”. The emphasis is on using some form of model as a repository of “trait” or knowledge extracted from previous candidate solutions. Instead of using genetic operations to create the next generation candidate solutions from the current solutions, EDA sampling the new candidates directly from this model, hence eliminate many difficulties involved in designing and performing those genetic operations.

The model in COIN is a joint probability matrix, H. This matrix represents a kind of Markov Chain. An entry in H_{xy} is a probability of transition from a state x to a state y. We call xy a coincidence of the event x and event y. This matrix, H, fits to represent combinatorial problems. Let’s illustrate this representation using Traveling Salesman Problem (TSP). A solution of a TSP problem is a tour, a combination of cities which can be represented by a string of numbers. Each number denotes a city, so, the following string is a tour of ten cities TSP problem.

1346785290

A coincidence is an adjacent pair of cities in the tour. There are ten coincidences in this tour. They are 1-3, 3-4, 4-6, 6-7, 7-8, 8-5, 5-2, 2-9, 9-0 and 0-1.

The joint probability matrix, H, is a square matrix of size $n \times n$. The sum over each row H_{xy} where y ranges from 1 to n equals to 1.0. It denotes the probability of the occurrence of xy in the tour. Each entry of H_{xy} has a value 0 to 1.0. The diagonal H_{xx} are zero.

Coincidence algorithm searches for solutions similar to any Evolutionary Algorithm, that is, starting from a random population of candidates; it selects some candidates and uses them to update H; with H, a new population of candidate is sampled; the selected candidate again, are used to update H; this process is repeated until satisfactory solutions are found.

```
Steps of the algorithm
1  Initialise H to a uniform distribution.
2  Sample a population from H.
3  Evaluate the population.
4  Select two groups of candidates: better, and worse.
5  Use these two groups to update H.
6  Repeate the steps 2-3-4-5 until satisfactory solutions are
found.
```

Fig. 1 Steps of the algorithm

These steps are quite standard and are similar to any Estimation Distribution Algorithm except for the step 4 and 5. The precise reason for this step will be discussed later. At this moment, let's discuss these steps. The joint probability matrix, H, is central to this algorithm. It is maintained and updated properly throughout the search cycle.

1 Initialise *H*

H is initially filled with a value $1/(n\text{-}1)$ (where *n* is the size of problem) except the diagonal H_{xx} is zero.

2 Sample a population

If the problem does not constrain the starting point, a random *x* is chosen, then, *xy* is sampled according to H_{xy}. The next step is then started at *y*. The next pair is again sampling from *H*. Any element that is a repeat of element that occurs earlier will have to be throwaway. This process is repeated until a combination of length *n* is reached. Each candidate is sampled this way. Sample a population of the required size.

3 Evaluate the population

Each candidate in the population is evaluated for it fitness according to some objective function.

4 Selection of candidates

The whole population is ranked. For a single objective problem this can be simple, the candidate is ranked by its fitness. For a multi objective problem, the most popular choice of ranking is the Pareto ranking [5]. The unique characteristic of COIN is that it selects two groups of candidates: better-group and worse-group. This notion of better/worse is relative to the average fitness of the population. The exact selection method can be varied according to problems, for example, best 10% and worst 10% or some other method "normalized" to the population sizes and/or the deviation of the fitness.

5 Updating the joint probability matrix

The update of *H* is separated into two components: reward and punishment. The reward is the increase of H_{xy} by the occurrence of the pair *xy* found in the better-group candidates. The incremental step is $k/(n\text{-}1)$ where *k* denotes the step size, *n* the length of a candidate. The punishment is the decrease of H_{xy} by the occurrence of the pair *xy* found in the worse-group candidates with similarly calculation to the reward. Here is the equation:

$$H_{xy}(t+1)=H_{xy}(t)+\tfrac{k}{n-1}(r_{xy}-p_{xy}) \qquad (1)$$
$$\tfrac{+k}{(n-1)^2}\left(\textstyle\sum_z p_{xz}-\sum_z r_{xz}\right)$$

Where *k* denotes the step size, *n* the length of a candidate, r_{xy} the number of occurrence of *xy* in the better-group candidates, p_{xy} the number of occurrence of *xy* in the worse-group candidates. H_{xx} are always zero.

The term, $k/(n\text{-}1)(r_{xy} - p_{xy})$, is the reward and punishment of the occurrence of a *xy* pair found in both groups. The last term, $k/(n\text{-}1)^2$ $(\sum p_{xz} - \sum r_{xz})$, represents the adjustment step for all "others" H_{xz} $(z \neq y, z \neq x)$ in the opposite direction hence keeping the sum of all probabilities in a row constant to one.

2.2 Computational Cost and Space

For the problem size *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 requires time $O(m \log m)$
3. The generator require space $O(n^2)$
4. Updating the joint probability matrix requires time $O(mn^2)$

2.3 Compare to other algorithms

There are many algorithms that use the second order statistic and considered to be in Estimation of Distribution Algorithms. These algorithms take dependencies between pairs of variables into account. The algorithms in this class include MIMIC [6], COMIT [7] and BMDA [8].

MIMIC (Mutual Information Maximizing Input Clustering) is one of the most famous algorithms in the bivariate dependency class, proposed by De Bonet et al. in 1997. It is a greedy algorithm that searches in each generation for the best permutation between the variables in order to find the probability distribution. COMIT (Combining Optimizers with Mutual Information Tree) is a successor of PBIL [9]. The algorithm was proposed by Baluja and Davies. The algorithm constructs dependency trees and incrementally learns from the good seen solutions. 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.

For multi objective problems, the most popular algorithm is Non-dominated sorting genetic algorithm II (NSGA-II)[10]. NSGA-II is a Genetic Algorithm that incorporates Pareto-ranking into its selection method. 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.

To measure the performance of COIN, several benchmarks on TSP problems are performed and the results are compared to the experiment of Robles and Larrañaga [11]. The performances of the algorithm are measured in two main aspects: quality of the results and the number of function evaluations. Only the result of the well known Gröstel24 is shown, which can be obtained from the TSPLIB [12]. The experiment of Robles and Larrañaga use both of the discrete and continuous EDAs in the following methods: UMDA [13], TREE [14], EBNA [15], UMDAc, and MIMIC. Moreover the results are also compared with GA in the literature of Larrañaga [16] in 1999 which uses GENITOR [17] algorithm. For each of the combinations shown in the experiment, ten runs are performed and the results are averaged.

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**	1272	1272			1272	1272		

* Size of population 200, mutation used SM
** Learning step *k = 0.1,* Adaptive selection
Optimum 1272

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

TABLE II
NUMBER OF GENERATIONS FOR THE GRÖSTEL24 PROBLEM

	Population & Local Optimization			
Algorithm	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 II illustrates the number of the generation used to find the solutions. Again the result shows that COIN is competitive in the larger population size.

For multi objective problems, COIN is tested with the multi-objective TSP. The Pareto-ranking similar to NSGA-II is used in the selection method of COIN. Two-objective random 100-city asynchronous TSP is generated. The behavior of the algorithm can be seen in Fig. 2 and Fig 3. Fig. 2 shows the population in the generation 1, 100 and 300 respectively. The population migrates towards the optimum as the algorithm progresses.

Fig. 3 shows the effect of reward and punishment. 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.

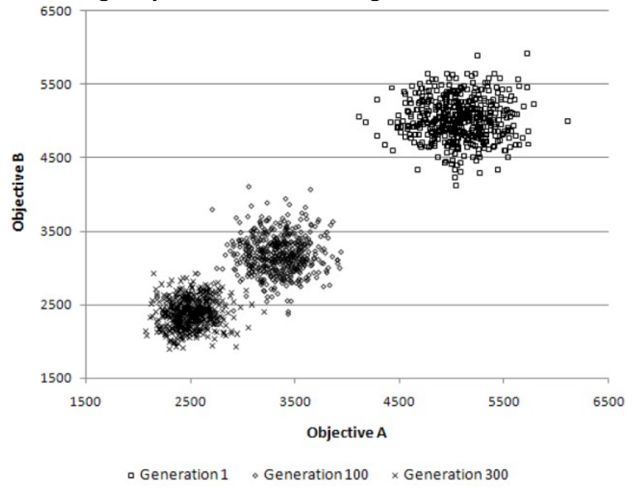


Fig. 2 The population clouds in a random 100-city 2-obj TSP

3. Applications

COIN is especially good at combinatorial problems. In this section, it is applied to solve manufacturing problems. Many interesting problems in manufacturing are multi-objectives. Two case studies are illustrated to show how to use COIN in real world problems.

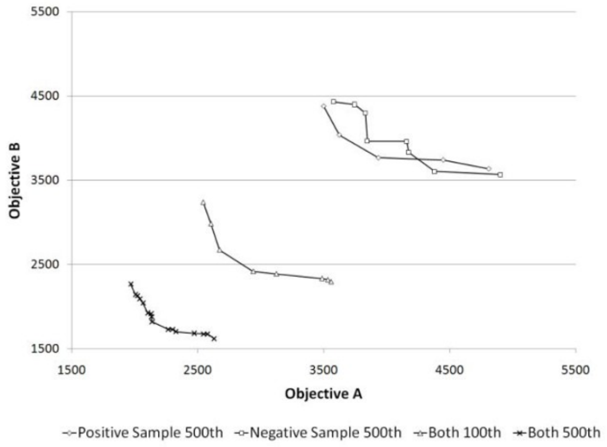


Fig. 3 The parato frontier obtained from different generation in a random 100-city 2-objective TSP.

Proponents of the lean manufacturing and just-in-time (JIT) philosophies assert that U-shaped assembly systems offer several benefits over traditional straight-line layouts [18] especially an improvement in labor productivity. U-lines have become popular in order to obtain the main benefits of smoothed workload, multi-skilled workforce and other principles of the JIT philosophy.

An assembly line is a manufacturing process in which component parts are added to a product in a sequential manner to create a finished product. Assembly lines are special flow-line production systems which are of great importance in the industrial production of high quantity standardized commodities. Recently, assembly lines have even gained importance in low volume production of customized products (mass customization). Balancing an assembly line means allocating the basic assembly tasks to be carried out to different stations to achieve specific goals and all in compliance with given constraints. The main assembly line balancing objective is to balance the task workload across workstations so that no workstation has an excessively high or low task workload.

The U-line arranges machines or tasks around a U-shaped line in the order in which production tasks are serial. The sequence of tasks on the U-line is not fixed, making it possible to reallocate tasks to different line locations. Thus, the assignment of tasks to line locations can be altered. The system is one-piece flow manufacturing moving one piece at a time between tasks within a U-line. The task efficiency is proportional to the worker's performance. Standard operation charts specify exactly how all work is done. Workers can be reallocated periodically when production requirements change (or cycle time changes). This requires workers to have multi-functional skills to operate several different machines or tasks. It also requires workers to work standing up and walking because they need to operate at different locations. Whenever a worker arrives at a task, one performs any needed tasks at the task location, and then walks to the next task. Following the last task of a path, the worker returns to the starting point and works or waits for the start of the next cycle. The characteristics of the single U-shaped assembly line worker allocation problem are shown in Fig. 4.

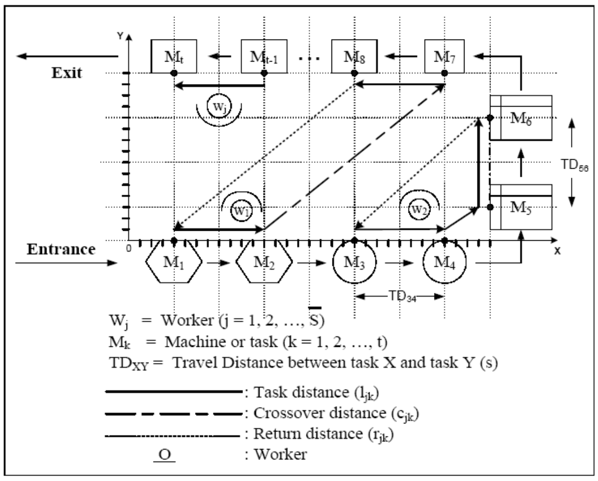

Exit
Entrance
M_t
M_{t-1}
M_8
M_7
M_6
M_5
M_1
M_2
M_3
M_4
W_j
W_1
W_2
TD_{56}
TD_{34}
Y
X
0
W_j = Worker (j = 1, 2, …, S)
M_k = Machine or task (k = 1, 2, …, t)
TD_{XY} = Travel Distance between task X and task Y (s)
: Task distance (l_{jk})
: Crossover distance (c_{jk})
: Return distance (r_{jk})
: Worker

Fig. 4 A single U-shaped assembly line for *j* workers and *k* machines on grid arrangement.

A number of cases with the number of tasks from 7 to 297 are used. Two objectives are: i) minimise the deviation of operation times of workers (DOW), ii) minimise the walking time (WT). The details of task description can be found in [3].

COIN is augmented with diversity preservation procedure when samples the population. A crowding distance approach [10] to generate a diversified population uniformly spread over the Pareto frontier and avoids a genetic drift phenomenon (a few clusters of populations being formed in the solution space). The salient characteristic of this approach is that there is no need to define any parameter in calculating a measure of population density around a solution.

To keep the best solutions found and to survive in the next generation, COIN uses an external list with the same size as the population size to store elitist solutions. All non-dominated solutions created in the previous population are combined with the current elitist solutions.

The multi-objective solution is solved with hierarchical procedure. First, the objective function of a number of workers is minimized. Secondly, only a minimum number of workers are selected to further evaluate a pair of minimum DOW and WT objective values as the Pareto-optimum frontier.

From the experimental results of symmetrical and rectangular U-shaped layouts, incrementing a number of workers in the former objective is sensitive to determining the walking time at only five percent of average processing time (or 0.14 to 65.61 seconds) in most problems. Just a few problems are at the ten and twenty percentage of Average Processing Time (or 0.42 to 4.08 seconds).

The results are compared to NSGA-II which is a popular multi-objective Genetic Algorithm. Both algorithms apply the following parameters:

Parameters of GAs [19]
Population sizes 100
Number of generations 100
Crossover probability 0.7
Mutation probability 0.3
Parameter of COIN [1]
Learning step coefficient (k) = 0.1

The results are shown in Table III and Table IV. COIN performs better than NSGA-II for most problem sets between Columns IV-VI. Furthermore, in terms of CPU time in the last column, the multi-objective COIN is much faster than NSGA-II. Their comparison for Scholl and Klein's 297 tasks at the cycle time of 2,787 time units is exemplified and shown in Fig. 5. Each of the worker allocation problem minimizes m, DOW and WT simultaneously. The final Pareto-optimal frontier consists of many strings that minimize the number of workers and show several pairs of DOW and WT.

TABLE III
NSGA-II WITH DISPLACEMENT FOR WORKER ALLOCATION AT THE SIDE RATIO OF 1:1 (1/3)

Problem / Task	Cycle Time	No. of workers	Convergence	Spread	Ratio	Time (seconds)
1.Merten / 7	7	6	0	0	1	534
	10	5	0.0372	0.5473	1	739
	18	2	0	None**	1	1,307
2.Miltenburg / 10	10	5	0.0170	0.6140	0.9000	1,958
3.Jackson / 11	7	9	0.2955	0.5034	1	1,122
	13	5	0.0968	0.5474	0.2500	2,045
	21	*4*	-	-	-	1,660
4.Thomopoulos / 19	120	4	0.0212	0.5929	0.5556	3,113
5.Heskiaoff / 28	138	9	0.0431	0.8780	0.5294	4,347
	256	5	0.0341	0.6020	0.5161	3,842
	342	4	0.0791	0.6126	0.5417	3,042
6.Kilbridge & Wester / 45	57	12	0.1287	0.7876	0.5000	7,145
	110	6	None**	None**	None**	6,660
	184	4	0.0251	0.7760	0.3571	6,024
7.Kim / 61	600	10	0.0380	0.5929	0.5556	11,126
8.Tongue / 70	160	*27*	-	-	-	13,110
	251	*17*	-	-	-	11,520
	527	8	0.0394	0.8028	0.1143	10,356
9.Arcus / 111	6,837*	24	None**	None**	None**	21,922
	7,916	20	None**	None**	None**	20,334
	17,067	9	0.1220	0.7696	0.2857	18,662
10. Scholl & Klein / 297	1,394	56	0.0468	0.5046	0.6250	170,546
	1,834	41	None**	None**	None**	136,890
	2,787	27	0.0386	0.5792	0.4000	200,973

* Minimum cycle time (5,755) is less than the operation time of 6,615. Thus, the feasible minimum cycle time from the data sets of UALBP-I is replaced.
** One local optimal solution (or one coordinate) on the DOW and WT

TABLE IV
COIN WITH DISPLACEMENT FOR WORKER ALLOCATION AT THE SIDE RATIO OF 1:1 (1/3)

Problem / Task	Cycle Time	No. of workers	Convergence	Spread	Ratio	Time (seconds)
1.Merten / 7	7	6	0	0	1	513
	10	5	0	0.3992	1	376
	18	2	0	None**	1	359
2.Miltenburg / 10	10	5	0.0304	0.6689	0.9091	506
3.Jackson / 11	7	9	0	0.5442	1	791
	13	5	0.0278	0.7711	1	644
	21	*3*	-	-	-	557
4.Thomopoulos / 19	120	4	0.0125	0.5558	0.7419	583
5.Heskiaoff / 28	138	9	0.0092	0.5471	0.6500	1,004
	256	5	0.0066	0.6898	0.6905	966
	342	4	0.0122	0.6271	0.6905	956
6.Kilbridge & Wester / 45	57	12	0.0657	0.6430	0.8000	1,465
	110	6	0.3710	0.7500	1	1,341
	184	4	0.0071	0.7879	0.7879	1,153
7.Kim / 61	600	10	0.0137	0.6627	0.6571	2,462
8.Tongue / 70	160	*26*	-	-	-	3,362
	251	*16*	-	-	-	2,611
	527	8	0.0074	0.7986	0.9512	1,792
9.Arcus / 111	6,837*	24	0.0574	0.5259	0.6667	4,556
	7,916	20	0	0.7500	1	4,122
	17,067	9	0.0069	0.9959	0.7343	3,877
10. Scholl & Klein / 297	1,394	56	0.1244	0.4524	0.7500	18,761
	1,834	41	0.0590	0.5529	1	17,013
	2,787	27	0.0269	0.7171	1	14,044

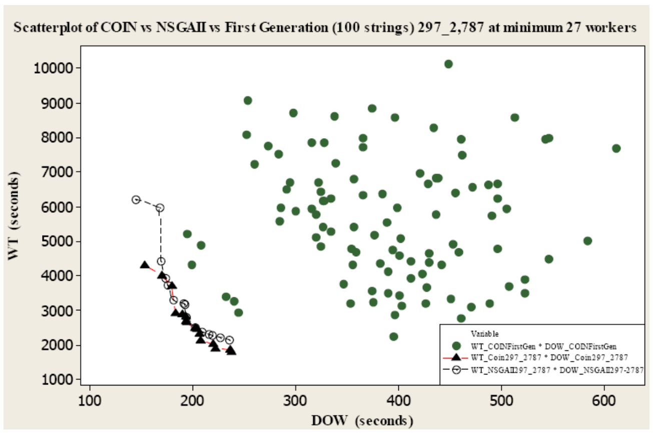


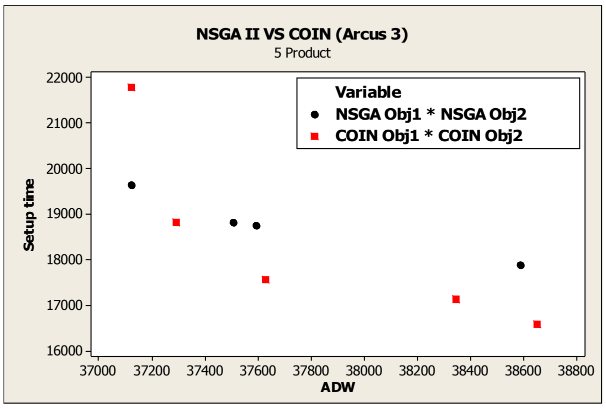
Fig. 5 Comparison of NSGA-II vs COIN for the 297-task problem

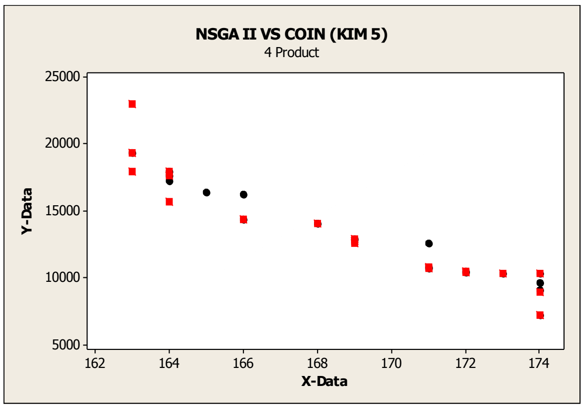
Comparing to NSGA-II on the results of convergence to the Pareto-optimal set, spread and ratio of non-dominated solution, and CPU time, COIN performs better than well-known NSGA-II for most problem sets.

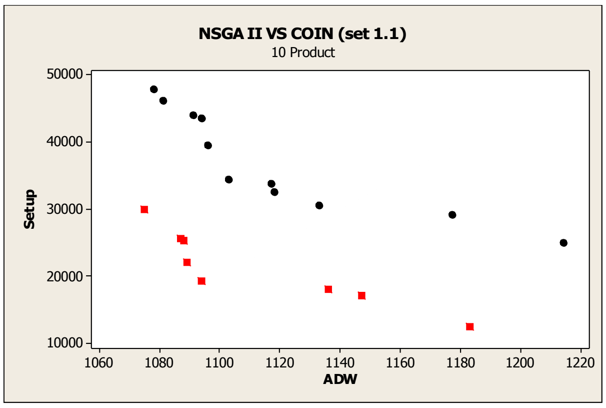
The next case study, the problem sets are from [20]. The problem is sequencing problems on mixed-model U-Shaped assembly lines. Two objectives are: i) minimise setup times and ii) minimise absolute deviations of workloads across workstations (ADW). The results are compared with the results from NSGA-II. Full details can be found in [2]. The parameters in the experiments are:

Parameters of NSGA-II
population size 100
maximum generation 500, 1000
crossover probability 0.5
mutation probability 0.05
Parameter of COIN
population size 100
maximum generation 500, 1000
reward and punishment 10% (15% for large problems)

The results are shown in Fig. 6. The comparison criteria are: convergence to Pareto optimal set, spread of Pareto, ratio of non-dominated solutions, and calculation time. Again, for this problem set, COIN outperforms NSGA-II in all criteria.







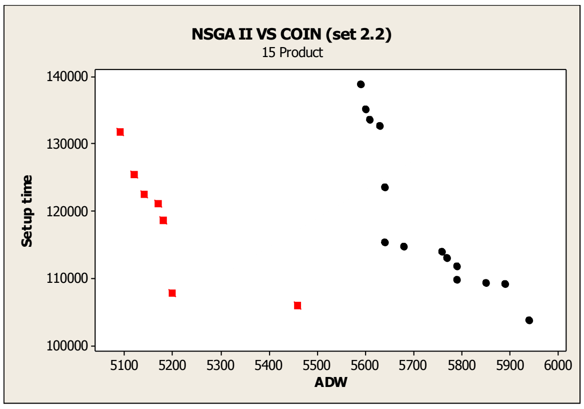


Fig. 6 Comparison of NSGA-II and COIN for several test sets

4. Conclusion

In this article, an evolutionary algorithm, Coincidence algorithm (COIN), is introduced. A joint probability matrix is used as its model. COIN searches and samples candidates for single and multi objectives problems very effectively. The algorithm has been benchmarked against several algorithms. The results show that the proposed algorithm is competitive with other discrete EDAs. Several benchmarks on real world industrial applications have been tested. The results show that the multiple objective version of COIN outperforms NSGA-II in most of the test set.

References

[1] Wattanapornprom, W. and Chongstitvatana, P.: Multi-objective Combinatorial Optimisation with Coincidence Algorithm, IEEE Congress on Evolutionary Computation, May 18-21, (2009)

[2] Parames Chutima, Noppon Kampirom, Warin Wattanapornprom and Prabhas Chongstitvattana: Application of Combinatorial optimization with coincidence for Multi-Objective Sequencing Problems on Mixed-Model U-Shaped Assembly Lines in JIT Production Systems, (Best paper award) Annual Conference of Kasetsart University, Bangkok, (2008)

[3] Ronnachai Sirovetnukul and Parames Chutima: The impact of walking time on U-shaped assembly line worker allocation problems, Engineering Journal, Vol 14, issue 2, Apr. (2010)

[4] Larrañaga, Pedro; & Lozano, Jose A. (Eds.). Estimation of distribution algorithms: A new tool for evolutionary computation. Kluwer Academic Publishers, Boston, (2002).

[5] Fonseca, C. M. and Fleming, P. J.: An Overview of Evolutionary Algorithms in Multiobjective Optimization. Evolutionary Computation, 3(1), pp. 1-16, (1995)

[6] 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)

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

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

[9] 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)

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

[12] TSPLIB, http://www.iwr.uniheidelberg.de/ groups/ comopt/ software/ TSPLIB95/ (retrieving on August 18th, 2008)

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

[14] 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)

[15] 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)

[16] 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)

[17] 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)

[18] C. H. Cheng, J. Miltenburg, and J. Motwani: The effect of straight- and U-shaped lines on quality, IEEE Transactions on Engineering Management, vol. 47, no. 3, pp. 321-334. (2000)

[19] R. K. Hwang, H. Katayama, and M. Gen: U-shaped assembly line balancing problem with genetic algorithm, International Journal of Production Research, vol. 46, no. 16, pp. 4637-4649. (2008)

[20] Kim, Y.K., Kim, J.K., and Kim, Y.H.: An endosymbiotic evolutionary algorithm for the integration of balancing and sequencing in mixed-model U-line, European Journal of Operational Research, 168(3), 838-852. (2006)