Applications of combinatorial optimisation

Prabhas Chongstitvatana
Faculty of Engineering
Chulalongkorn University

Applications of combinatorial optimisation

Prabhas Chongstitvatana

Faculty of Engineering

Chulalongkorn University

Combinatorial optimisation

- The domains of feasible solutions are discrete.
- Examples
  - Traveling salesman problem
  - Minimum spanning tree problem
  - Set-covering problem
  - Knapsack problem

Evolutionary Computation

EC is a probabilistic search procedure to obtain solutions starting from a set of candidate solutions, using improving operators to “evolve” solutions. Improving operators are inspired by natural evolution.

Coincident Algorithm (COIN)

Evolutionary Algorithms that makes use of models to generate solutions.

- “Estimation of Distribution Algorithms”
- “Competent Genetic Algorithms”

Model in COIN

- A joint probability matrix, *H*.
- Markov Chain.
- An entry in H_{xy} is a probability of transition from a state *x* to a state *y*.
- *xy* a coincidence of the event *x* and event *y*.

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.

Updating of *H*

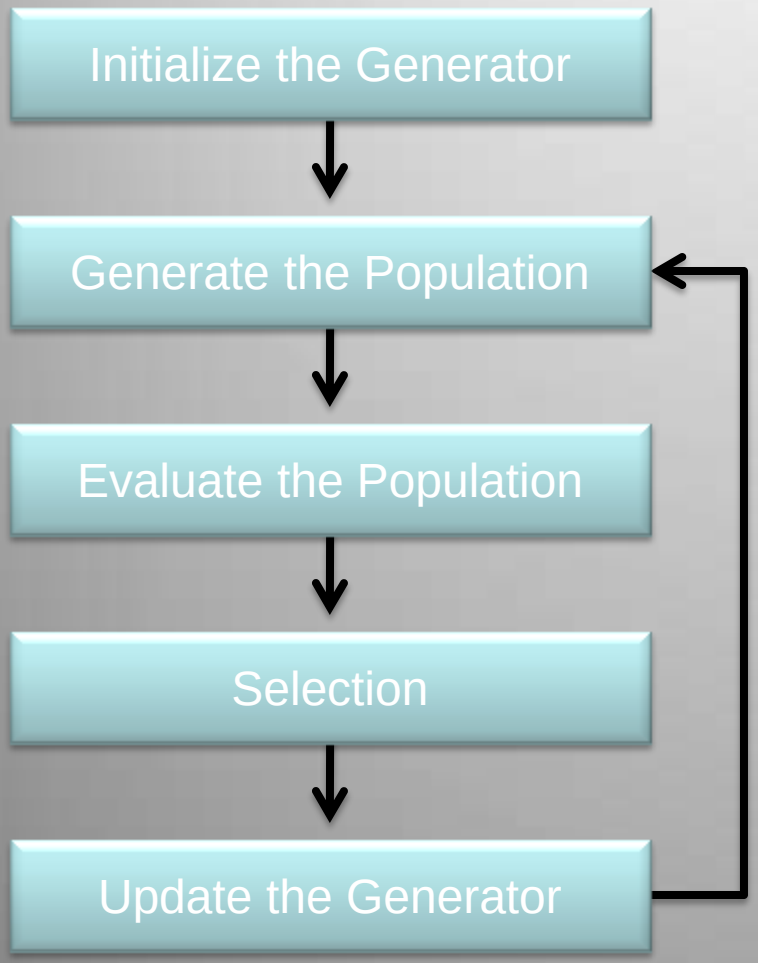
$$H_{xy}(t+1) = H_{xy}(t) + \frac{k}{n-1}\left(r_{xy} - p_{xy}\right) + \frac{k}{(n-1)^2}\left(\sum_z p_{xz} - \sum_z r_{xz}\right)$$

- *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.

Coincidence Algorithm

- Combinatorial Optimisation with Coincidence
  - Coincidences found in the good solution are good
  - Coincidences found in the not good solutions are not good
  - How about the Coincidences found in both good and not good solutions!

Coincidence Algorithm



Next Incidence

Prior Incidence

	X_1	X_2	X_3	X_4	X_5
X_1	0	0.25	0.25	0.25	0.25
X_2	0.25	0	0.25	0.25	0.25
X_3	0.25	0.25	0	0.25	0.25
X_4	0.25	0.25	0.25	0	0.25
X_5	0.25	0.25	0.25	0.25	0

The Generator

Coincidence Algorithm

Initialize the Generator

	X_1	X_2	X_3	X_4	X_5
X_1	0	0.25	0.25	0.25	0.25
X_2	0.25	0	0.25	0.25	0.25
X_3	0.25	0.25	0	0.25	0.25
X_4	0.25	0.25	0.25	0	0.25
X_5	0.25	0.25	0.25	0.25	0

$$P(X_1|X_2) = P(X_1|X_3) = P(X_1|X_4) = P(X_1|X_5)$$

$$= 1/(n-1)$$

$$= 1/(5-1)$$

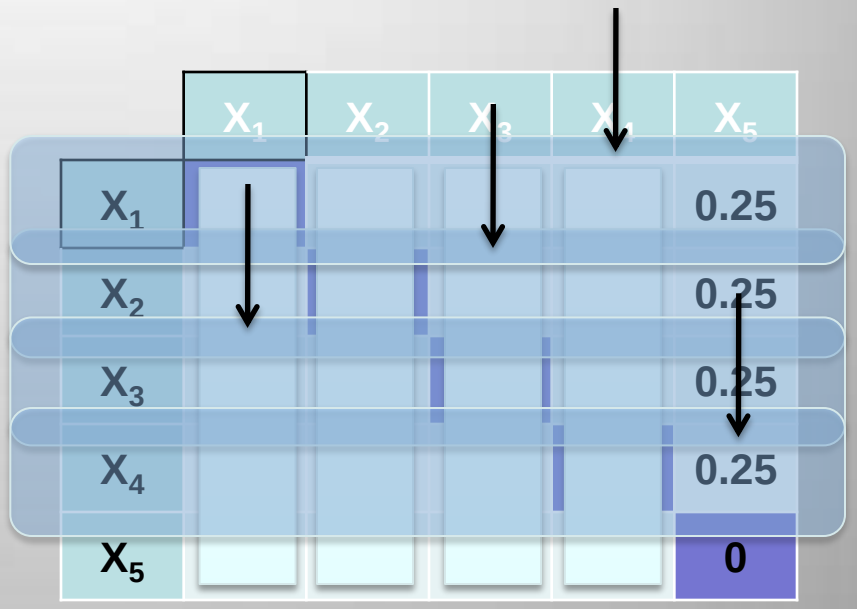
$$= 0.25$$

Coincidence Algorithm

Initialize the Generator

↓

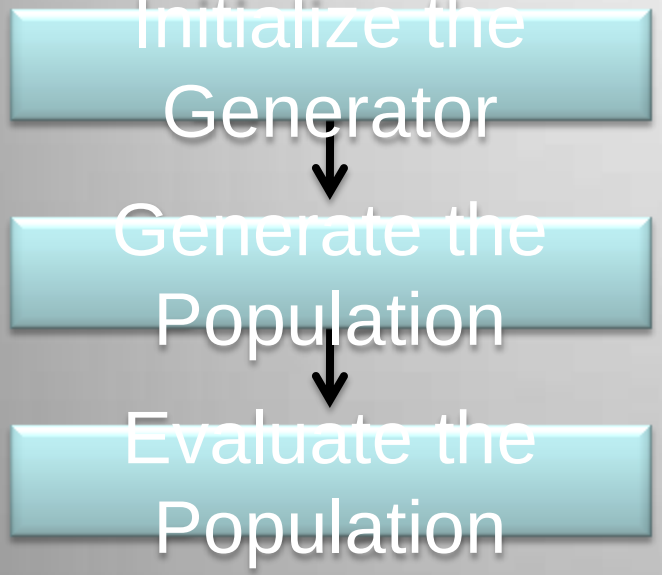
Generate the Population



1. Begin at any node X_i
2. For each X_i, choose its value according to the empirical probability $h(X_i|X_j)$.

X	X	X	X	X
2	3	1	4	5

Coincidence Algorithm



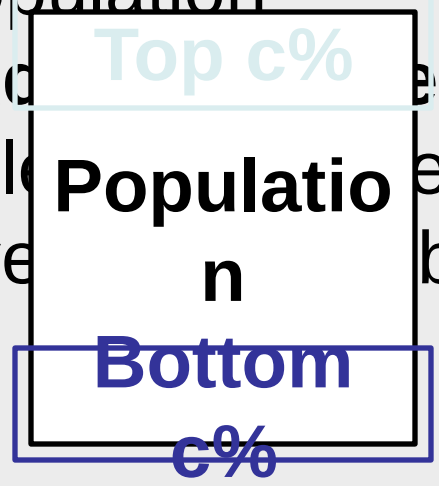
	X_1	X_2	X_3	X_4	X_5
X_1	0	0.25	0.25	0.25	0.25
X_2	0.25	0	0.25	0.25	0.25
X_3	0.25	0.25	0	0.25	0.25
X_4	0.25	0.25	0.25	0	0.25
X_5	0.25	0.25	0.25	0.25	0

Population					Fit
X_2	X_3	X_1	X_4	X_5	7
X_1	X_2	X_3	X_5	X_4	6
X_4	X_3	X_1	X_2	X_5	3
X_1	X_3	X_2	X_4	X_5	2

Coincidence Algorithm

Selection

- **Uniform Selection :**
selects from the top and bottom *c* percent of the population
- **Ac ection :**
sel e population above and below the ave band of two standard deviations.



Coincidence Algorithm

Update the Generator

Note:

Reward:
If an incidence X_i,X_j is found in the good string, the joint probability $P(X_i,X_j)$ is rewarded by gathering the probability d from other $P(X_i|X_{else})$

k=0.2

	X_1	X_2	X_3	X_4	X_5
X_1	0	0.20	0.20	0.40	0.20
X_2	0.20	0	0.40	0.20	0.20
X_3	0.40	0.20	0	0.20	0.20
X_4	0.20	0.20	0.20	0	0.40
X_5	0.25	0.25	0.25	0.25	0

Population					Fit
X_2	X_3	X_1	X_4	X_5	7
X_1	X_3	X_2	X_4	X_5	2

Discard

Coincidence Algorithm

Update the Generator

Note:

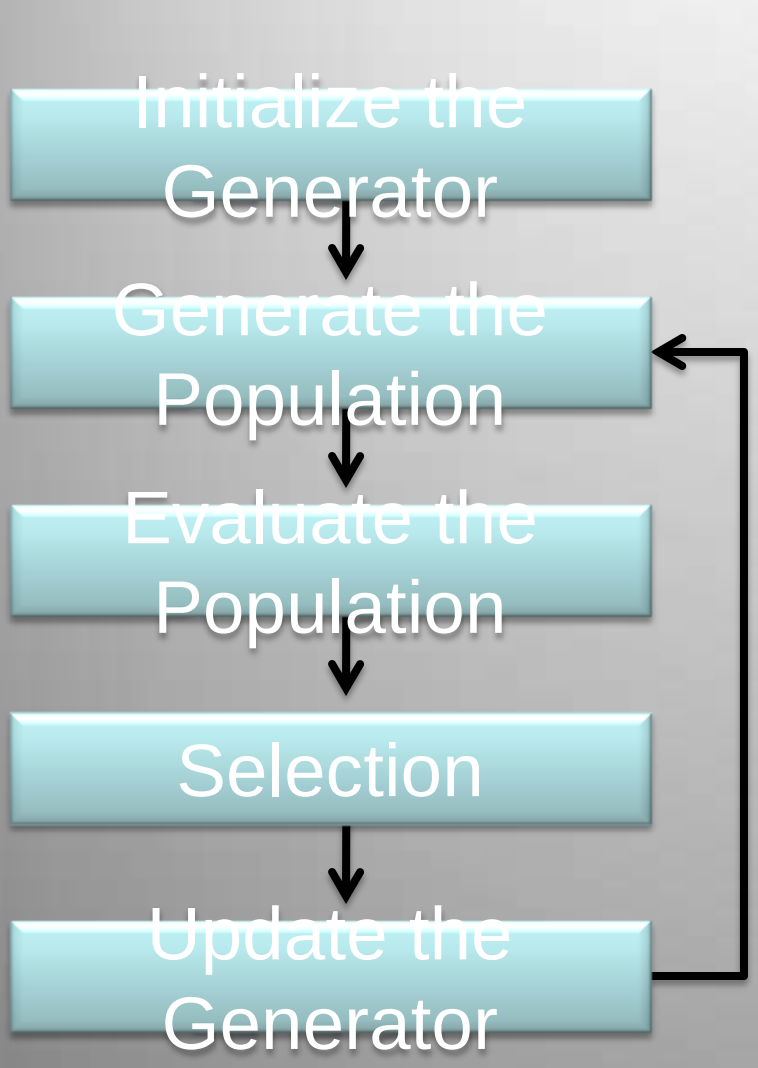
k=0.2

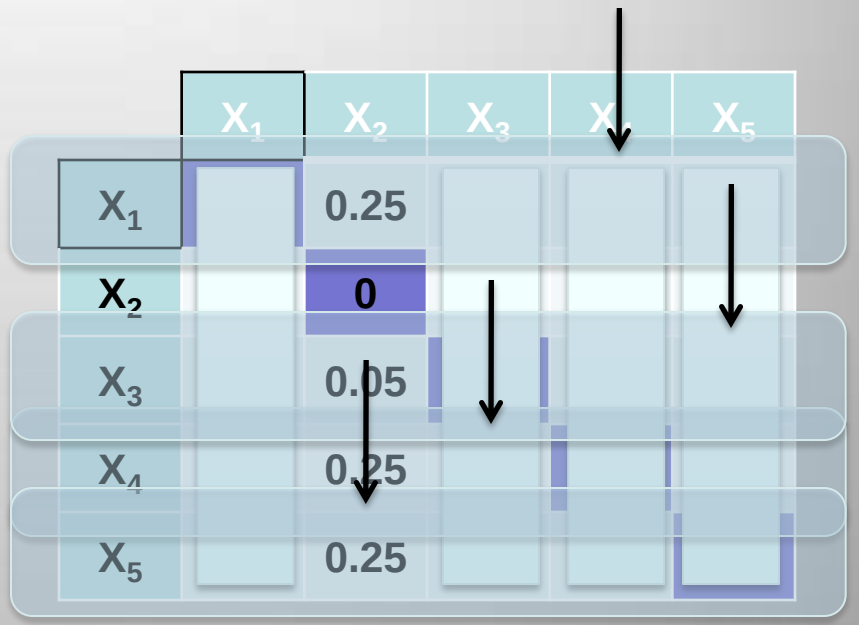
	X_1	X_2	X_3	X_4	X_5
X_1	0	0.25	0.05	0.45	0.25
X_2	0.25	0	0.45	0.05	0.25
X_3	0.45	0.05	0	0.25	0.25
X_4	0.25	0.25	0.25	0	0.25
X_5	0.25	0.25	0.25	0.25	0

Population					Fit
X_2	X_3	X_1	X_4	X_5	7
X_1	X_3	X_2	X_4	X_5	2

Discard

Coincidence Algorithm





X 1	X 4	X 3	X 5	X 2

Computational Cost and Space

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 $O(mn^2)$

TSP

TABLE I

TOUR LENGTH FOR THE GRÖSTEL24 PROBLEM

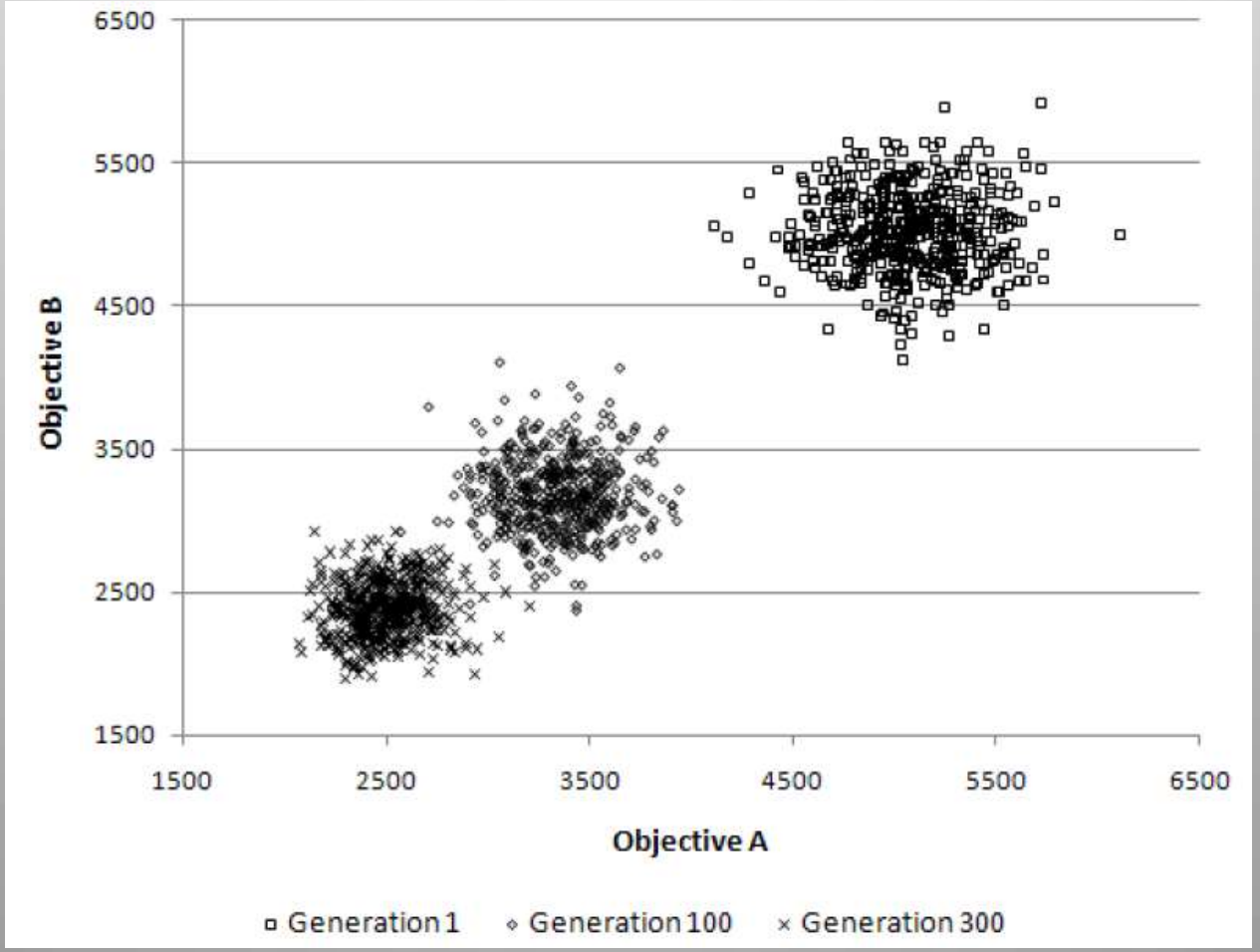
Algorithm	Population & Local Optimization							
	500-without		500-with		1000-without		1000-with	
	Best	Aver	Best	Aver	Best	Aver	Best	Aver
GA-ER*	1272	1272						
GA-OX2*	1300	1367						
UMDA	1339	1495	1272	1272	1329	1496	1272	1272
MIMIC	1391	1486	1272	1272	1328	1451	1272	1272
TREE	1413	1486	1272	1272	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

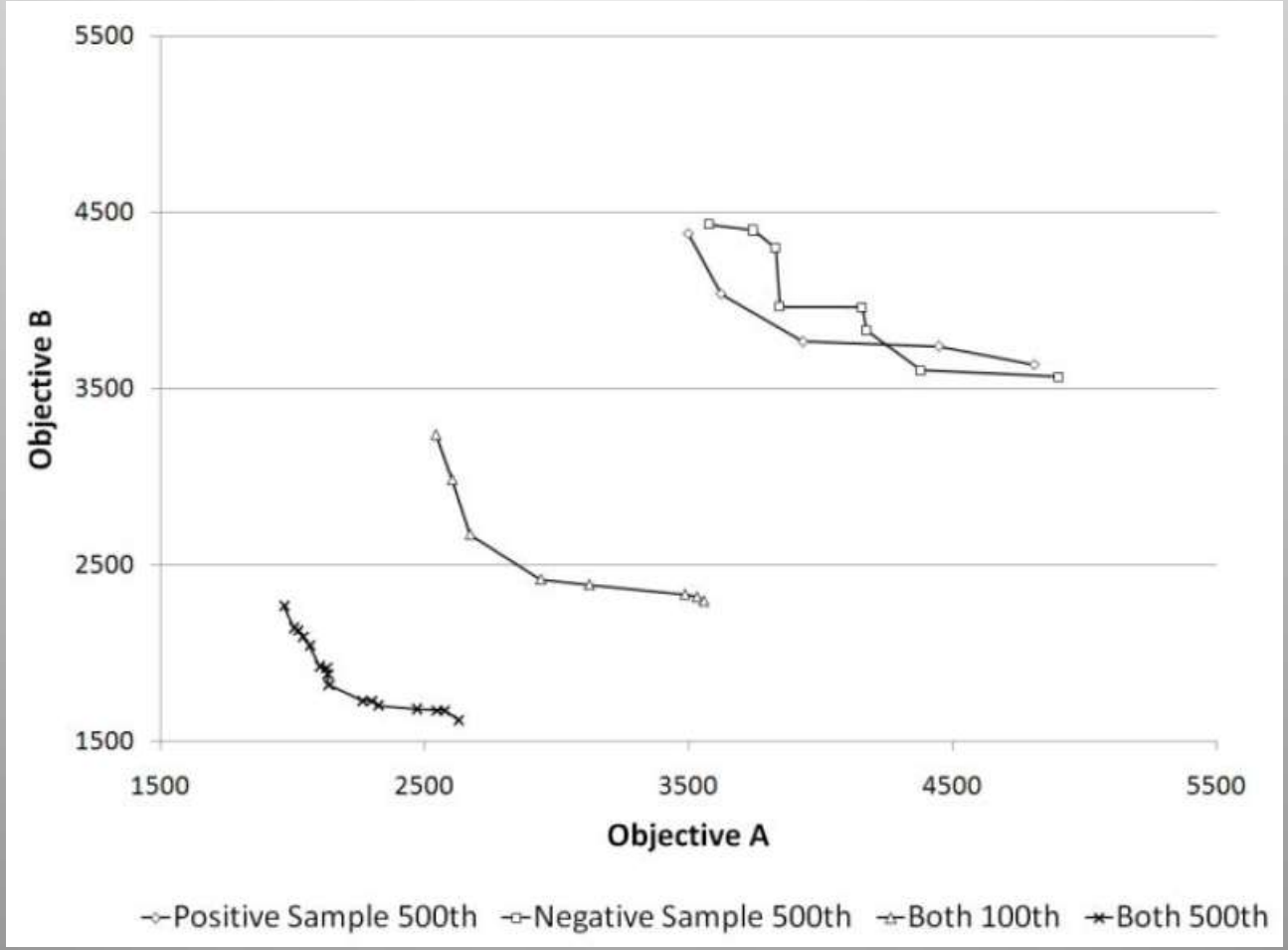
Optimum 1272

Multi-objective TSP

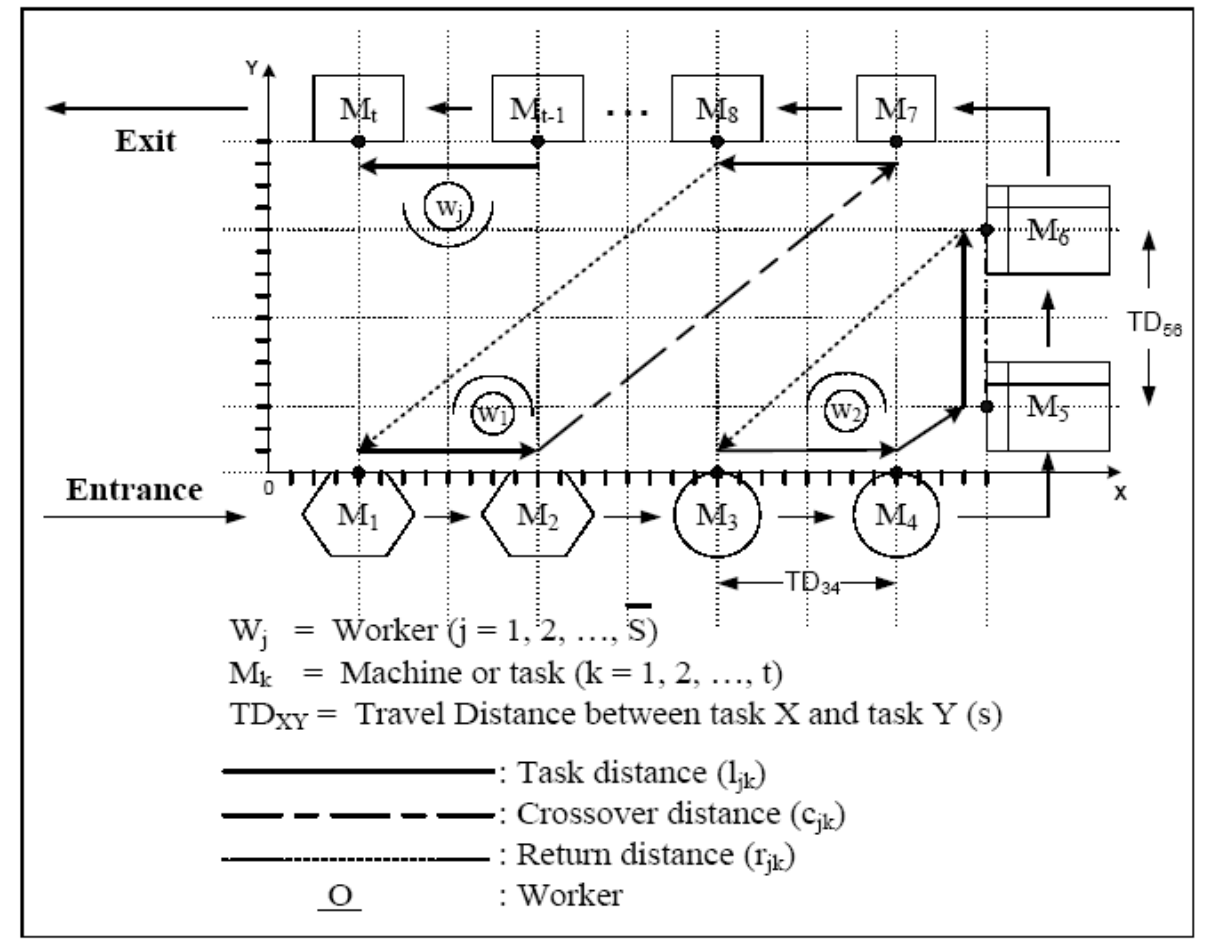


The population clouds in a random 100-city 2-obj TSP

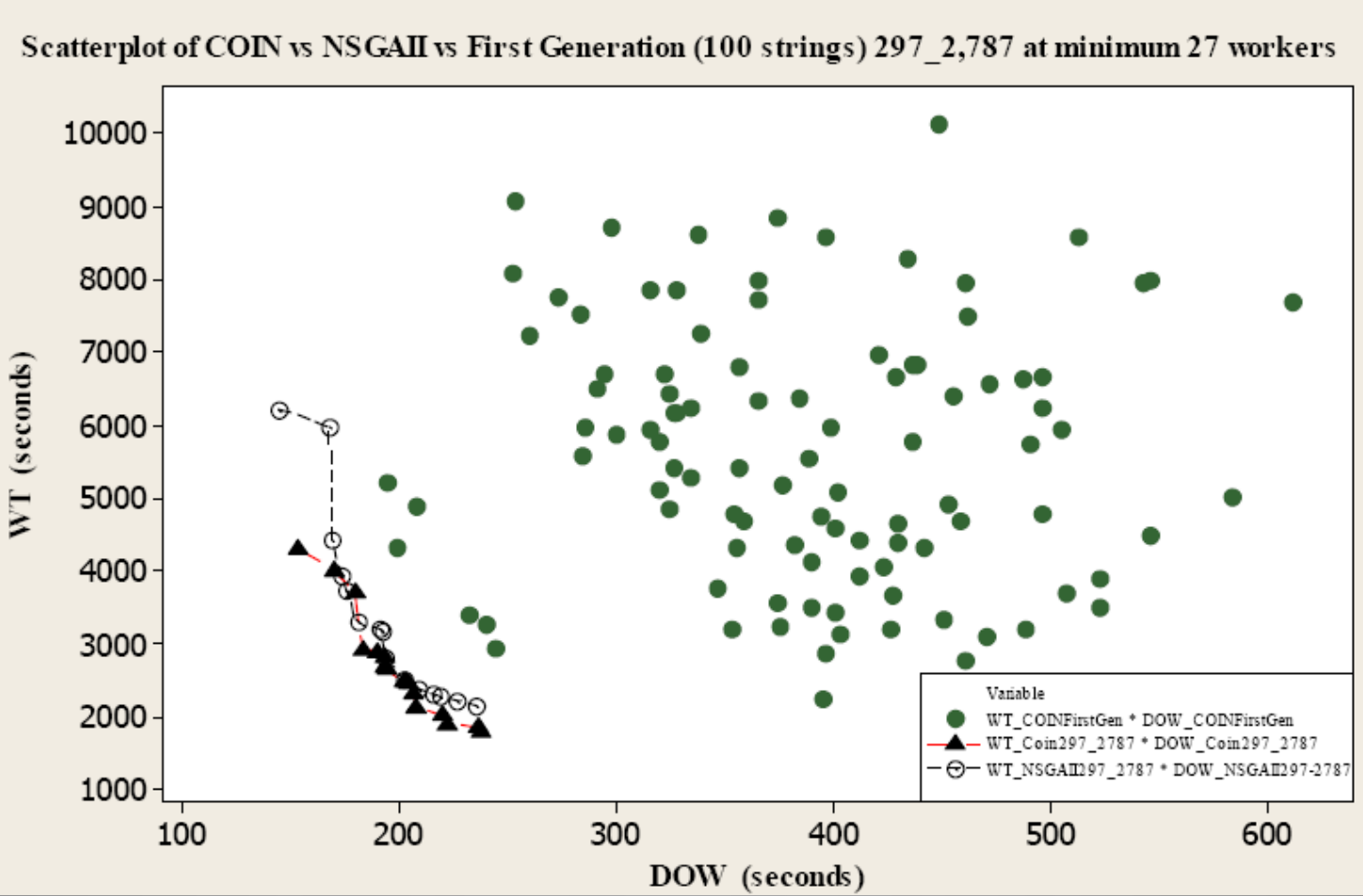
Role of Negative Correlation



U-shaped assembly line for *j* workers and *k* machines



Comparison for Scholl and Klein's 297 tasks at the cycle time of 2,787 time units



More Information

COIN homepage

- http://www.cp.eng.chula.ac.th/faculty/pjw/project/coin/index-coin.htm

My homepage

- http://www.cp.eng.chula.ac.th/faculty/pjw