DISTRIBUTED QUERY PROCESSING USING PARTICLE SWARM OPTIMIZATION

A dissertation submitted to the Jawaharlal Nehru University
in partial fulfillment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND TECHNOLOGY

BY

AMIT KUMAR

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

JULY 2012

Dedicated to my Parents

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

DECLARATION

This is to certify that the dissertation entitled **"Distributed Query Processing using Particle Swarm Optimization"** is being submitted to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Computer Science & Technology**, is a record of bonafide work carried out by me under the supervision of **Dr. T.V. Vijay Kumar**.

The matter embodied in the dissertation has not been submitted in part or full to any University or Institution for the award of any degree or diploma.

Amit Kumar
(Student)

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067

CERTIFICATE

This is to certify that this dissertation entitled **“Distributed Query Processing using Particle Swarm Optimization”** submitted by **Mr. Amit Kumar**, to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, for the award of degree of **Master of Technology** in **Computer Science & Technology**, is a research work carried out by him under the supervision of **Dr. T. V. Vijay Kumar**.

Dr. T. V. Vijay Kumar
(Supervisor)

Prof. Karmeshu
(Dean)

Acknowledgement

I take this opportunity to express my gratuitous expressions of gratitude towards faculty, seniors and friends who have helped me to complete this dissertation. Several individuals deserve a special mention for their contribution to this work.

I would like to thank my supervisor, **Dr. T.V. Vijay Kumar** for providing excellent guidance, indispensable ideas, incessant support and unflagging encouragement during the completion of this work. He has always been munificent with his time and suggestions. He understood me through my difficult times and always boosted my confidence. More importantly he demonstrated his faith in my dormant abilities. He has always helped me immeasurably in my professional socialization; has always kept me focused on imperative issues.

I owe a special debt to my friend **Rahul Singh** who has played a significant role in helping me to complete this work. Without his support, I may never have finished this dissertation. He has been a tremendous resource in erudite discussions about academic or social problems with his intelligent, clear-headed perspectives and well-reasoned opinions. He rattles off brilliant ideas with alarming frequency.

A special thank you note goes to **Mr. Akshay Kumar, Mr. Mohammed Haider, Mr. Santosh Kumar** and **Mr. Kumar Dilip** for their astute counsel and stimulation at critical decision-making points. They all have been great assets to me.

I would also like to thank all my classmates for their valuable suggestions and helpful comments.

I also gratefully acknowledge the support that I have received from the School while working on this dissertation. In particular, I thank JNU for supporting me with generous fellowships.

Amit Kumar

CONTENTS

CHAPTER 1

Introduction

A distributed database encompasses coherent data, disseminated over the sites of a computer network [CP84]. A Distributed Database Management System (DDBMS) deals with managing such distributed databases. It presents a simple and unified interface to the users so that they can access the databases as if the data is not distributed [OV91]. A DDBMS is illustrated in Figure 1.1 showing data distributed over databases connected by a network.

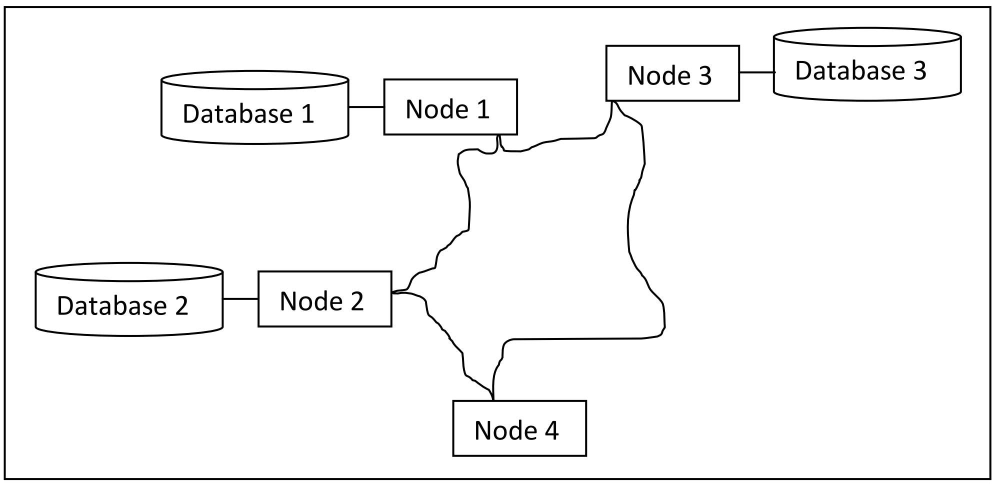

Figure 1.1: An example DDBMS

1.1 Distributed Query Processing

In query processing, the aim is to formulate algorithms that analyze queries and convert the queries into a set of data manipulation operations [OV91] as shown in figure 1.2

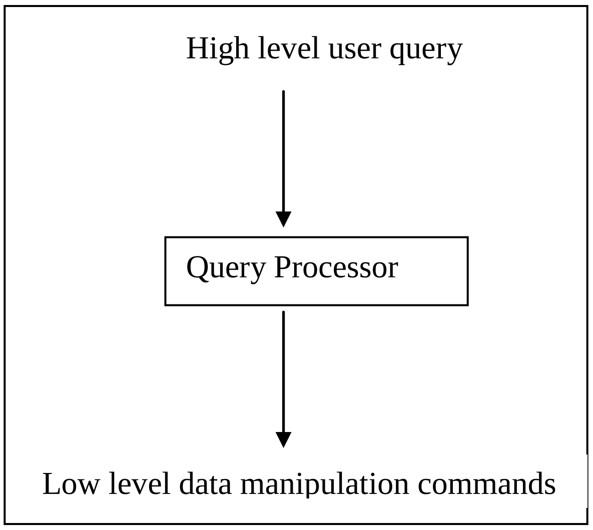

Figure 1.2: Query Processing

The query processing problem is much more complicated in distributed environments, as there are various parameters affecting the performance [AHH09]. In distributed environment, it is possible that the relations required in a query may be fragmented or replicated thereby leading to additional communication costs [OV91]. Furthermore, the query response time may become very high due to processing of distributed query at multiple sites. The performance of a DDBMS is determined by its ability to process queries in an effective and efficient manner [RM97]. Distributed query processing involves CPU, I/O and communication cost. However, it is the communication cost that constitutes the major cost of query processing [KYY82]. In order to answer the distributed queries, data is transmitted among the participating database sites, which incurs communication cost [YC84]. Hence, in order to reduce the communication cost, various strategies for executing a distributed query over the network are devised in distributed query optimization.

Processing distributed join queries typically involves three phases i.e. copy identification, reduction, and assembly [YC84]. In copy identification phase, files required by the query are selected for processing [YC84] [MLR90] [YC83]. In the reduction phase, local selection, projection and semijoins are used to reduce the size of data that needs to be transmitted in order to perform join operations [RM97]. In the assembly phase, files are processed to get the final output. In this phase reduced files may have to be transmitted to join sites every so often to accomplish join operations. Additionally the final output may have to be transmitted to the result site. Join sites and join order must be determined for this phase [RM97].

1.1.1 Related Work

Earlier research in distributed query processing has typically focused on only one of the three phases of query processing discussed above. The reduction phase has received the most attention. The objective in the reduction phase is to find a minimum cost semi join sequence that fully reduces the relations required by a query [RM97]. A relation required by a query is said to be fully reduced if all its rows, not satisfying the qualification of the query, have been removed prior to transmitting it to the join site [YC84] [YC83]. In the reduction phase it is assumed that the local processing cost is trivial likewise it is assumed that all join operations are performed at the result site [AHY83] [BC81] [BG+81] [HY79] [LW86] [PV88] [S86] [SW91] [YL89]. Using semi joins can reduce the amount of data that need to be transmitted; then again it can also drastically increase local processing costs. Semi joins effectively perform the join twice – once to reduce one of the relations and again to join the reduced relation with the other relation [RM97] as shown in Figure 1.3.

R_1	
A	**B**
a_1	b_1
a_2	b_1
a_2	b_2
a_2	b_4
a_3	b_4
a_4	b_7
a_4	b_9

R_2	
B	**C**
b_1	c_2
b_2	c_1
b_2	c_4
b_5	c_2
b_6	c_4
b_7	c_2
b_8	c_3

B	C
b_1	c_2
b_2	c_1
b_2	c_4
b_7	c_2

R_2 after semijoin $R_2 \ltimes R_1$ on attribute B

A	B	C
a_1	b_1	c_2
a_2	b_1	c_2
a_2	b_2	c_1
a_2	b_2	c_4
a_4	b_7	c_2

Resulting relation From $R_1 \bowtie R_2$

Figure 1.3: Examples of semijoin and join operations.

The use of semi joins must be considered more carefully as the local processing cost may become more significant compared to data communication costs [CY93][CY94][CY88][LM+85][LC85][ML86] [ME92].

Several approaches exist in literature that focuses on copy identification and assembly ignoring reduction. In these approaches, local processing costs are considered imperative, and joins are used as query processing technique [MR95] [MLR90]. These algorithms determine the optimum join order, join methods and join sites in order to minimize the sum of communication cost and local processing cost. According to [RM97] the sites at which joins are performed and the join-order have a crucial effect on the overall query processing cost. These needs to be included in query optimization to globally optimize distributed query processing. The fundamental assumption in these approaches is that data is stored non-redundantly. Some approaches use both semi joins and joins. These approaches identify beneficial semi joins, join order and join sites to reduce the local processing costs as well as communication costs [LW86]. In these approaches it is assumed that relations are pre-selected. These algorithms have a high computational complexity, which limits its applicability [YL89].

In distributed query processing, copy identification is an important issue as the cost of processing a query varies significantly with respect to the file copies used [RM97]. The number of possible solutions grows exponentially with respect to the number of files [RM97]. Thus optimal copy identification and the determination of an optimal query processing strategy are interdependent. The use of criteria such as minimization of sites containing required files is not likely to result in globally optimal solutions [YC84] [YC83].

1.2 The Problem

A large number of queries are posed against distributed databases spread across the globe. These queries need to be processed efficiently. For this purpose, optimal query processing strategies generating efficient query processing plans are devised. In distributed database systems, due to replication of relations at multiple sites, the relations required to answer a query may require access of data from many sites. This leads to exponential increase in the number of possible alternative query plans to process a query [IK90]. However, it is not computationally feasible to explore all possible query plans in such a large search space [IK90]. This problem in literature is referred to as a combinatorial optimization problem in distributed databases [JK84]. The query plan that gives rise to cost-effective query processing is considered necessary and should be generated for a given query [VSV11]. The problem, discussed in [VSV11], has been addressed in this dissertation. This problem can be illustrated with the help of following example:

Consider a query that accesses four relations R_1, R_2, R_3 and R_4, which are distributed across multiple sites. The relation-site matrix is shown in Figure 1.4.

Relations	**Sites**			
R_1	S_2	S_4	S_6	S_8
R_2	S_2	S_3	S_5	S_7
R_3	S_2	S_1	S_9	S_9
R_4	S_2	S_3	S_4	S_2

Figure 1.4: Relation-site matrix

The valid query plans are given in Figure 1.5.

8	5	2	2	R_1 in site S_8, R_2 in site S_5, R_3 in site S_2 and R_4 in site S_2
4	5	9	2	R_1 in site S_4, R_2 in site S_5, R_3 in site S_9 and R_4 in site S_2
2	2	2	3	R_1 in site S_2, R_2 in site S_2, R_3 in site S_2 and R_4 in site S_3
2	2	2	2	R_1 in site S_2, R_2 in site S_2, R_3 in site S_2 and R_4 in site S_2
5	5	2	2	R_1 in site S_5, R_2 in site S_5, R_3 in site S_2 and R_4 in site S_2

Figure 1.5: Query Plans

As the number of sites containing the relations accessed by the query increases, the number of possible valid query plans also increases. One way to generate query plans that lead to efficient query processing is by reducing the number of sites involved in query processing [VSV11]. As the number of distinct sites involved in processing the query decreases, the site-to-site communication cost decreases. Thus, the query plan should involve less number of sites. For the query plans, given in Figure 1.5, the first query plan involves 3 sites, the second query plan involves 4 sites, the third query plan involves 2 sites and the fourth query plan involves only 1 site. Accordingly, the fourth query plan is preferred over others, as it involves the least number of sites i.e. 1.

In case the numbers of sites involved in the query plans are equal, the query plan having sites with higher concentration of relations is more desirable [VSV11]. Since in this case the join operations between relations are performed at a single site. For the query plans in Figure 1.5, the third and the fifth query plans involve the same number of sites, i.e. 2. The third query plan has three relations $R_{1,}$ R_2 and R_3 in site S_2 and relation R_4 at site S_3 whereas the fifth query plan has two relations each in site S_5 and in site S_2. So, the third query plan has a higher concentration of relations at an individual site i.e. 3 and therefore should be preferred over the fifth query plan.

The above two aspects are used to define a 'close' query plan in [VSV11]. The query plan involving fewer sites, and having higher concentration of relations, is considered more 'close' and is preferred over the others. For the query plans in Figure 1.5, the ordering of query plans, based on descending order of closeness, is given in Figure 1.6.

2	2	2	2	R_1 in site S_2, R_2 in site S_2, R_3 in site S_2 and R_4 in site S_2
2	2	2	3	R_1 in site S_2, R_2 in site S_2, R_3 in site S_2 and R_4 in site S_3
5	5	2	2	R_1 in site S_5, R_2 in site S_5, R_3 in site S_2 and R_4 in site S_2
8	5	2	2	R_1 in site S_8, R_2 in site S_5, R_3 in site S_2 and R_4 in site S_2
4	5	9	2	R_1 in site S_4, R_2 in site S_5, R_3 in site S_9 and R_4 in site S_2

Figure 1.6: Query Plans ordered based on "close" Query Plans

The query plans higher in the order involve fewer sites and higher concentration and therefore should be generated before query plans that are lower in the order involving larger number of sites.

Based on the two aspects discussed above, a cost function, that computes the cost of proximity of data relevant for answering a user query, is defined in [VSV11]. This cost, referred to as Query Processing Cost (QPC), is given below:

$$QPC = \sum_{i=1}^{M} \frac{S_i}{N}\left(1 - \frac{S_i}{N}\right)$$

Where M is the number of sites accessed by the query plan S_i is the number of times the i^{th} site is used in query plan, N is the number of relations accessed by the query.

The QPC varies between zero and (N-1)/N. Zero indicates that all the relations accessed by the queries, reside at the same site. (N-1)/N indicates that each of the relations, accessed by the query, is in different sites. The query plans having less QPC are considered “close” and therefore are generated before the ones having higher QPC.

1.3 Aim

The query plan generation problem, based on the above heuristic, has been solved using Genetic Algorithms in [VSV11]. In this dissertation, an attempt has been made to solve this query plan generation problem using particle swarm optimization. The dissertation aims to address this problem in the following manner:

(i) A query plan generation problem is formulated as a single objective optimization problem where the objective is to minimize the Query Processing Cost, as defined above. This problem is solved using Set based Comprehensive Learning Particle Swarm Optimization (S-CLPSO). The performance of the S-CLPSO based approach is compared with the query plan generation approach based on Genetic algorithms.

(ii) The query processing cost (QPC), as discussed above, defines “close” query plans as those that involve fewer sites and higher concentration of relations in sites. The former can be formulated as the site communication cost (SCC) and the latter can be formulated as the relation concentration gain (RCG). These formulations, which are motivated by [PV02(c)], are given below:

Objective 1: The first objective considered is the minimization of total communication cost. It is based on the number of sites required to process a user query, lesser the number of sites involved in query processing, lesser will be the communication between the sites. As a result, query processing will be efficient. So if s is the number of sites being used and m is the number of communications then this objective can be calculated by the following expression:

$$Min\ SCC = m \times s$$

Where s is the number of sites being used and m is the number of communications

Objective 2: Another objective considered is that if there are more than one query plans having the minimum number of required sites, the query plan having sites with greater concentration of relations provides efficient results and shall accordingly be preferred over the others. So if n is the number of relations in the query and c_i is the count of sites and arranged in decreasing order then this objective can be calculated by the following expression:

$$Max\ RCG = \sum_{i=1}^{s} (n - i + 1)c_i$$

Where n is the number of relations in the query, c_i is the count of sites arranged in decreasing order and s is the number of sites involved.

Thus, minimizing QPC comprises of minimizing SCC and maximizing RCG. So, the single objective query plan generation problem is formulated as a bi-objective query plan generation problem with the two objectives namely minimizing SCC and maximizing RCG. This bi-objective problem is solved using Set based Comprehensive Learning Parallel Particle Swarm Optimization (S-CLPPSO). The performance of the S-CLPPSO based approach is compared with the single-objective query plan generation approach based on S-CLPSO.

1.4 Organization of the Dissertation

The dissertation is organized as follows: Chapter 2 discusses query plan generation using single-objective particle swarm optimization (PSO). The bi-objective query plan generation problem is solved using multi-objective particle swarm optimization (MOPSO) technique in chapter 3. Chapter 4 is conclusion.

CHAPTER 2

Distributed Query Plan Generation Using PSO

In nature, a large number of insects and other small organisms are generally organized in hierarchies, e.g. ants, bees and fish etc. In these organisms although each individual agent has limited responses, the agents all together exhibit fascinating behavior and obvious traits of intelligence. For example, fish maintain a greater mutual distance when swimming carefree, while they come together in very dense groups in the presence of predators [U4]. In order to preserve the personal integrity of each member of the group, the members of the group respond collectively against the external threats. The swarm is able to change its current form rapidly by breaking into smaller parts and then reuniting again when there is no danger. This observed behavior of natural systems has stimulated scientific curiosity regarding the underlying rules that produce this behavior. Systems, where such collective phenomena occur, prepare the ground for the development of swarm intelligence [U7].

2.1 Swarm Intelligence

Swarm intelligence is a branch of artificial intelligence that studies the collective behavior and emergent properties of complex, self-organized, decentralized systems with social structure [U16]. Although each agent has a very limited action space with no central control, the aggregated behavior of the whole swarm exhibits traits of intelligence i.e. an ability to react to environmental changes and decision-making capacities [U5]. Notwithstanding their physical or structural differences, such systems share common properties based on **five** basic principles of swarm intelligence, which are discussed next.

2.1.1 Basic Principles of Swarm Intelligence

The five basic principles of swarm intelligence are: proximity, quality, diverse responses, stability and adaptability [M94]. **Proximity** is the ability to perform space and time computation. The group should be able to do elementary space and time computations. Since space and time translate into energy expenditure, the group should have some capability to calculate the benefit of a particular response to the environment in these terms [SK86]. **Quality** is the ability to respond to environmental quality factors. The group should be able to respond not only to time and space considerations, but also to the quality factors, e.g. quality of foodstuffs or safety of location. **Diverse Responses** are the ability to deliver a multiplicity of different responses. The group should not allocate all of its resource along extremely narrow lines. It should seek to allocate its resources along many modes as assurance against the abrupt change in anyone of them due to environmental fluctuations. **Stability** is the ability to preserve robust behaviors under mild environmental changes. The group should not change its behavior from one mode to another with every fluctuation of the environment. Such changes consume energy without producing a useful return for the investment. **Adaptability** is the ability to change behavior when it is dictated by external factors. When the rewards for changing a behavioral mode are expected to be worth the investment in energy, the group should be able to change its behavioral mode.

2.1.2 Swarm Intelligence Techniques

Three main swarm intelligence optimization algorithms are: Stochastic Diffusion Search, Ant Colony optimization and Particle Swarm Optimization [B89] [D92] [EK95].

2.1.2.1 Stochastic Diffusion Search (SDS)

It is an agent-based probabilistic global search and optimization technique [B89]. It is particularly suitable to problems in which the objective function can be decomposed into multiple independent partial-functions [U6]. In this technique, each agent maintains a hypothesis that is iteratively tested by evaluating a randomly selected partial objective function parameterized by the agent's current hypothesis [U18]. Agents share hypotheses via a one-to-one communication approach. A positive feedback mechanism ensures that a population of agents eventually becomes stable around the global-best solution.

2.1.2.2 Ant Colony Optimization (ACO)

It is a novel metaheuristic, which is inspired by the foraging behavior of real ants, for solving combinatorial or other optimization problem [D92]. When real ants search for food, in the beginning they search the area surrounding their nest in an erratic manner. Once an ant finds food source, during the return trip, this ant lays down a chemical substance called pheromone on the ground. The deposited pheromone guides other ants to the food source. This indirect communication between the ants, via the pheromone trail, is known as stigmergy [U15]. This stigmergy facilitates them to locate the shortest path between their nest and food source. Artificial ants, imitating the real ants, locate optimal solutions by exploring and exploiting search space representing all possible solutions and record their positions and the quality of their solutions in order to achieve better results in subsequent iterations [U2].

2.1.2.3 Particle Swarm Optimization

Particle swarm optimization (PSO) is a population based stochastic optimization technique designed for continuous nonlinear optimization problem [U12]. It is based on the simulation of the social behavior of birds within a flock. In an experiment, given in

[EK95], an attempt to graphically simulate the elegant and erratic choreography of a bird flock was carried out [EK95]. The aim was to determine patterns that govern the ability of birds to fly synchronously and to abruptly change direction with a regrouping in an optimal formation [SM+08]. From the initial idea, the concept developed into a simple and efficient Optimization technique.
This dissertation focuses on solving distributed query plan generation problem using particle swarm optimization, which is discussed in detail next.

2.2 Particle Swarm Optimization

In Particle Swarm Optimization (PSO), particles are simply the agents that fly through the search space and simultaneously record the best position that they have hitherto come across. This value is identified as personal best or *pbest* and is possibly communicated at times. Another best value that is recorded by the PSO is the best value attained up to now by any particle in the swarm. This value is called global best or *gbest*.

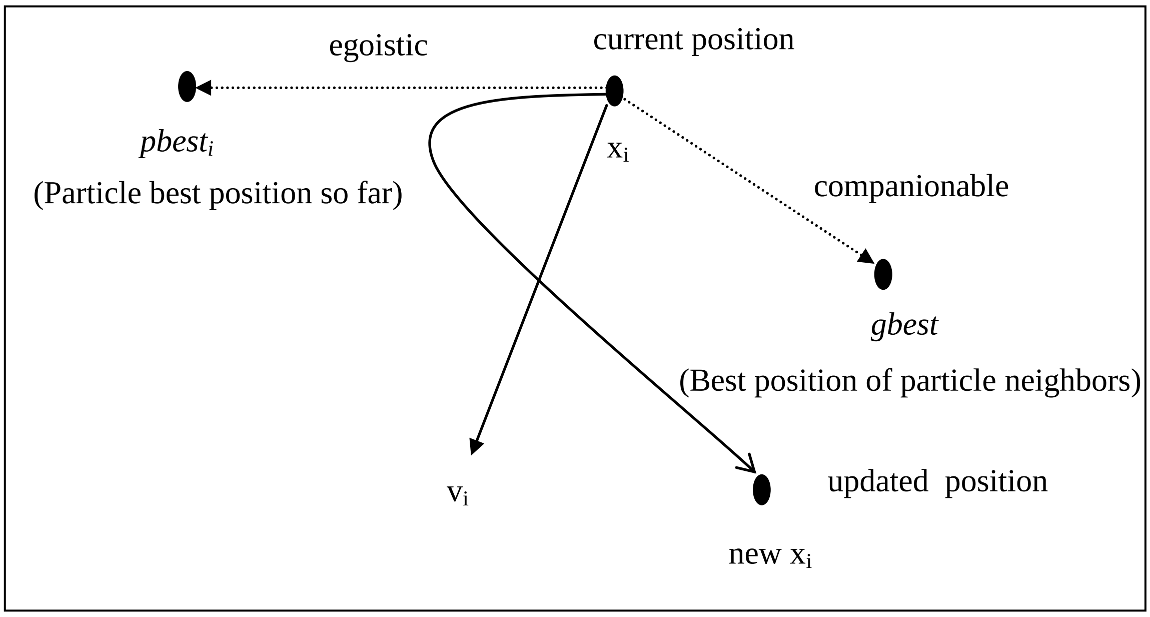

Figure 2.1: Basic Concept of PSO [U17]

Figure 2.1 represents the basic concept of PSO which is to accelerate each particle toward its *pbest* and the *gbest* locations, with a random weighted acceleration at each time step in order to update its position [U11].

The original version of PSO [EK95] is defined by the following equations (1) and (2) as:

$$v_{id}(t+1) = v_{id}(t) + c_1 r_1 \left(p_{id}(t) - x_{id}(t)\right) + c_2 r_2 \left(p_{gd}(t) - x_{id}(t)\right) \qquad (1)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \qquad (2)$$

i = 1, 2…, N,   d = 1, 2,…, n

Where $\boldsymbol{i}$ is the particle, t denotes the iteration counter, r_1 and r_2 are random variables uniformly distributed within [0,1] , c_1 and c_2 are weighting factors also called cognitive and social parameters respectively , g represents the index of the particle with the best fitness, and d is the dth dimension. $v_{id}(t)$ denotes velocity of the particle at time't'. $p_{id}(t)$ denotes personal best position of the particle at time 't'. $x_{id}(t)$ is the position of the particle at time 't'. $p_{gd}(t)$ denotes the global best position of the particle at time 't'. $x_{id}(t+1)$ denotes the position of the particle at time 't+1'. $v_{id}(t+1)$ denotes the velocity of the particle at time't+1'. The update of position $x_{id}(t+1)$ and velocity $v_{id}(t+1)$ of ith particle in the swarm [PV88]] is shown in Figure 2.2.

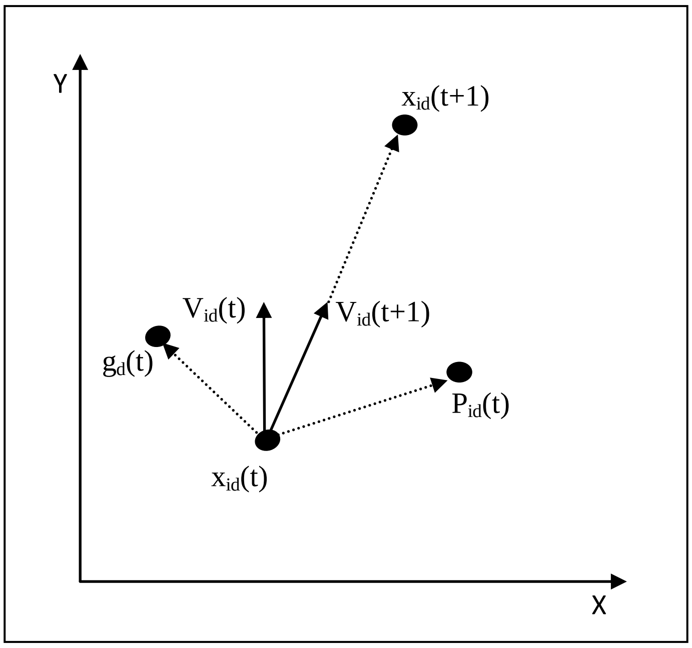

Figure 2.2: Position and velocity update of ith particle in the swarm [U10]

In every iteration, after the update and evaluation of particle's position and velocity is completed, the *pbest* and *gbest* positions (memory) are also updated. The flow chart describing the PSO algorithm is shown in Figure 2.3.

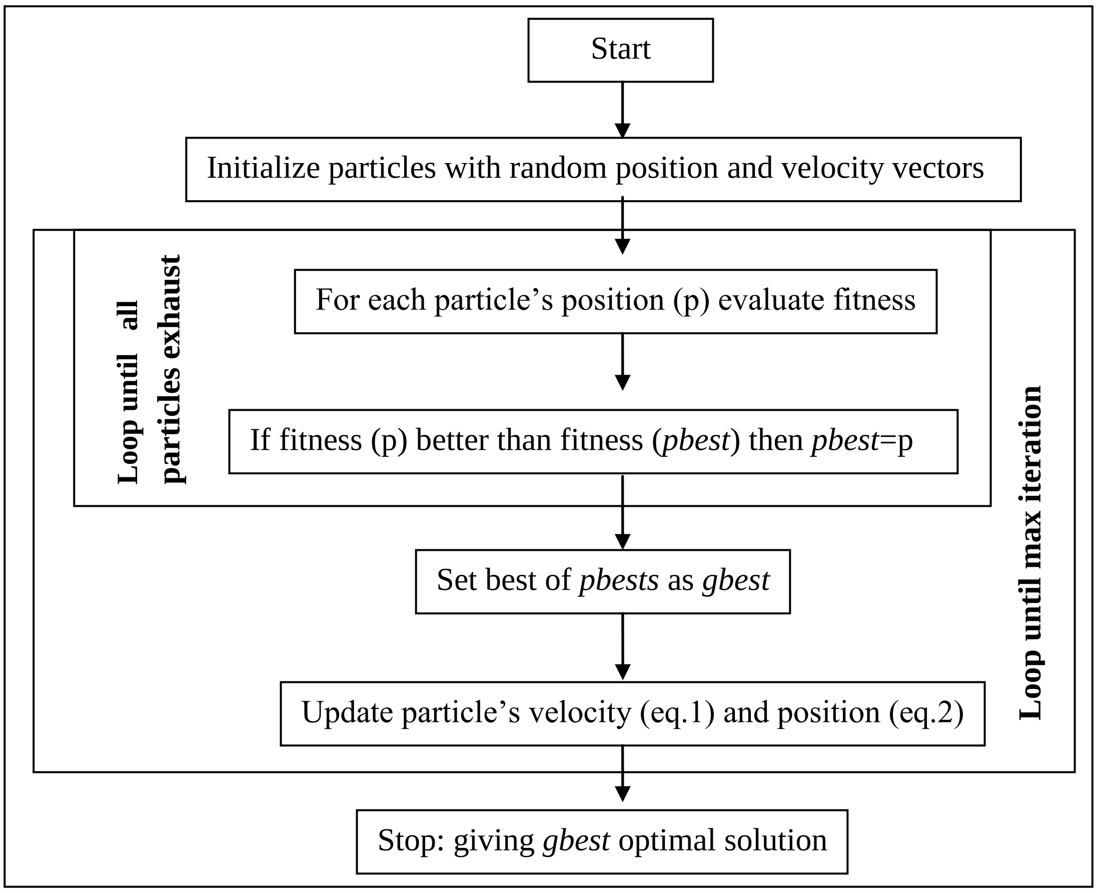
Start
Initialize particles with random position and velocity vectors
Loop until all particles exhaust
For each particle's position (p) evaluate fitness
If fitness (p) better than fitness (pbest) then pbest=p
Set best of pbests as gbest
Update particle's velocity (eq.1) and position (eq.2)
Loop until max iteration
Stop: giving gbest optimal solution

Figure 2.3: Flow chart depicting the General PSO Algorithm [U19]

2.3 Refinements in PSO

Early PSO variants performed satisfactorily for simple optimization problems. However, their crucial deficiencies were revealed as soon as they were applied on harder problems with large search spaces and a multitude of local minima.

2.3.1 The Concept of Inertia Weight

The first significant issue was the swarm explosion effect. This deficiency was due to uncontrollable increase in the magnitude of velocities. For this purpose, a new parameter, ω called inertia weight, was introduced in equation (1), resulting in a new PSO variant [ES98] defined by the following equations (3) and (4) as:

$$v_{id}(t+1) = \omega v_{id}(t) + c_1 r_1 \left(p_{id}(t) - x_{id}(t)\right) + c_2 r_2 \left(p_{gd}(t) - x_{id}(t)\right) \qquad (3)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \qquad (4)$$

i = 1, 2,…, N,   d = 1, 2,…, n

The inertia weight parameter can be varied as per the requirement for global and local search. A large inertia weight is more suitable for global search, and a small inertia weight ensures local search [SE98]. The inertia weight ω is selected such that the effect of $v_{id}(t)$ fades away gradually over iterations.

2.3.2 The Concept of Neighborhood

The global information exchange scheme allows each particle to know instantly the overall best position after each iteration. This might lead to pre-mature convergence. For this purpose, the concept of neighborhood was introduced with the main idea to limit the information exchange amongst the neighbor and not amongst all particles in the swarm [K99] [MKN03]. Each particle assumes a set of particles to be in its neighborhood and in each iteration, it communicates its best position only to the neighboring particles, instead of the whole swarm. So (global best) p_{gd} is changed to (local best) p_{ld} in equation (1) resulting in a new PSO variant [K99] [MKN03] defined by the following equations (5) and (6) as:

$$v_{id}(t+1) = \omega v_{id}(t) + c_1 r_1 \left(p_{id}(t) - x_{id}(t)\right) + c_2 r_2 \left(p_{ld}(t) - x_{id}(t)\right) \qquad (5)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \qquad (6)$$

i = 1, 2,…, N,   d = 1, 2,…, n

The scheme for determining the neighbors of each particle is called neighborhood topology [K99][MKN03]. In case of complex problem, PSO considering small neighborhood performs better, while PSO considering a large neighborhood is more useful on simple problems [ZSD10]. Also, topology can change with time instead of remaining fixed throughout the execution. Such dynamic topologies have been used in multiobjective optimization problems [HE02]. Nevertheless, the vast majority of local best (lbest) models in literature are based on ring or star topology, as shown in Figure 2.4.

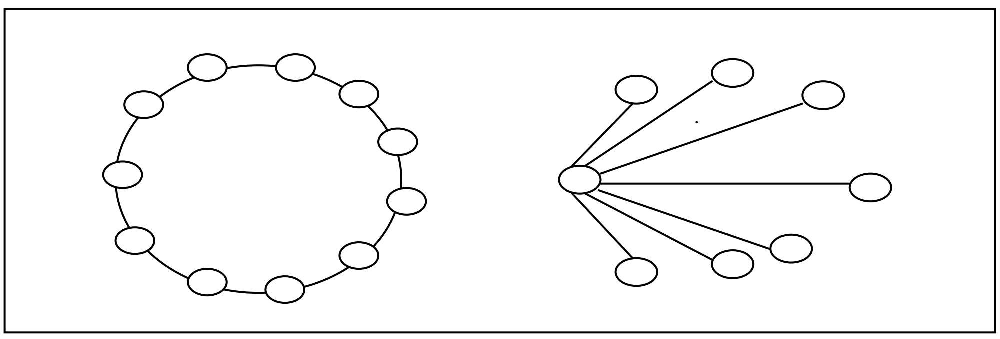

Figure 2.4: Common neighborhood topologies of PSO: ring (left) and star (right) [U13]

2.3.3 The Concept of Constriction Coefficient

Another method for controlling the velocities of particles is to use another parameter called constriction coefficient or constriction factor (χ) [CK02]. This modified version is defined by the following equations (7) and (8) as:

$$v_{id}(t+1) = \chi\left[\omega v_{id}(t) + c_1 r_1 \left(p_{id}(t) - x_{id}(t)\right) + c_2 r_2 \left(p_{gd}(t) - x_{id}(t)\right)\right] \qquad (7)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \qquad (8)$$

i = 1, 2,…, N, d = 1, 2,…, n

Where χ is a parameter called constriction coefficient or constriction factor, while the rest of the parameters remain the same as in the case of previously described PSO models.

$$\chi = \frac{2}{\left|2-\emptyset-\sqrt{(\emptyset^2-4\emptyset)}\right|}, \text{ Where } \varphi = c_1 + c_2, \text{ and } \varphi > 4$$

2.3.4 Hybrid PSO Algorithms

There are numerous variants of Hybrid PSO algorithms proposed in the literature that utilize basic mechanism of PSO and the natural selection mechanism, crossover, and mutation which is usually considered in Evolutionary Computing methods such as Genetic Algorithm [LRK01]. In [PPV07], a memetic algorithm based hybrid PSO approach that uses the random walk with directional exploitation local search method was proposed.

2.4 Further Refinements in PSO

Premature convergence in solving multimodal problems with large search spaces is the main drawback in most of the variants of PSO [RMN11]. In the original PSO [EK95], each particle learns from its *pbest* and *gbest* simultaneously [BH07]. The problem with this approach is that all particles in the swarm learn from the *gbest* even if the current *gbest* is far from the global optimum [PBP09]. In such circumstances, particles may simply be attracted and confined into a local optimum if the search environment is complex with various local solutions. To overcome this, three novel learning strategies were proposed in [LQ+04] to improve the performance of original PSO [EK95]. The three versions are: elite learning PSO (ELPSO), multi-exemplars learning PSO (MLPSO), and comprehensive learning PSO (CLPSO). The distributed query plan generation approach presented in this chapter is based on CLPSO discussed next.

2.4.1 Comprehensive Learning Particle Swarm Optimizer (CLPSO)

In CLPSO [LS+06], for each particle, any one of the particle in the swarm can be used as an exemplar to be learned from i.e. any one of the particles' *pbest*, including it, is used to update the velocity of a particular particle [LS+06]. Each dimension of a particle can also choose to learn from a different exemplar. With this novel learning strategy the particles have more choice in terms of exemplars to learn from. The particles can fly through a large search space. In this strategy, better quality solutions are generated effectively by using the information in the swarm.

The velocity and position updating equation in CLPSO [LS+06], for a d-dimensional problem are defined by the following equations (9) and (10) as:

$$V_i^d = \omega * V_i^d + c * rand_i^d * \left(pbest_{f_i(d)}^d - X_i^d\right) \qquad (9)$$

$$X_i^d = X_i^d + V_i^d \qquad (10)$$

Where $f_i(d)$ defines which particle's *pbest* the i^{th} particle should follow on the d^{th}dimension. $pbest_{f_i(d)}^d$ can be the corresponding dimension of any particle's *pbest* including its own *pbest*, and the decision depends on the learning probability. Different initial learning probability values for different particles are set at the beginning of searching and are kept stable throughout the whole searching process.

For each dimension of particle, a random number is generated. If this random number is larger than learning probability of the particle then the corresponding dimension will learn from its own *pbest* otherwise it randomly chooses two particles out of the population, excluding itself, This is followed by comparing the fitness values of their *pbest* of the chosen two particles' and selecting the one with a better fitness values of *pbest* as an exemplar for the dimension. If a particle is an exemplar of itself on all dimensions then a dimension is chosen randomly to learn from the dimension of some other randomly chosen particle's *pbest.* New exemplars are chosen for a particle when it fails to improve itself over a pre-specified generation.

2.4.2 CLPSO's Search Behavior

The differences between the search behavior of the original PSO [EK95] and the CLPSO [LS06] were discussed in [LS+06]. In CLPSO, instead of using particle's own *pbest* and *gbest* as the exemplars, all particles' *pbest* can be considered as exemplar to guide particle's flying direction. These operations increase the particles' initial diversity and enable the swarm to overcome premature convergence problem.

Further, in the original PSO, for a certain dimension, if the *pbest* and *gbest* are on opposite sides of the particle's current position as shown in Figure 2.5, the *pbest* and *gbest* may make the particle oscillate. However, the *gbest* is more likely to provide a larger momentum, as $|gbest - X|$ is likely to be larger than the $|pbest - X|$. Hence, the *gbest* may influence the particle to move in its direction even if it is in a local optimum region far from the global optimum [LS+06].

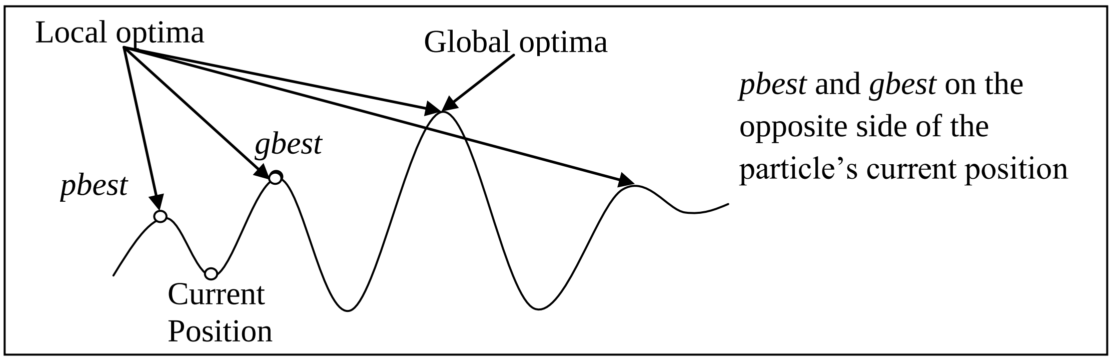
Local optima
Global optima
pbest and gbest on the opposite side of the particle's current position
gbest
pbest
Current Position

Figure 2.5: Oscillation of particle between *gbest* and *pbest*

If *pbest* and *gbest* are on the same side of the particle's current position and points to the same local optima (as shown in Figure 2.6), the particle will move in that direction and it may be impossible to jump out of the local optimum area once its falls into the same local optimum region where the *gbest* is [LS+06]. However, in CLPSO, the particle can fly in other directions by learning from other particles' *pbest* when the particle's *pbest* and *gbest* fall into the same local optimum region. Hence, CLPSO strategy has the ability to jump out of local optimum via the cooperative behavior of the whole swarm.

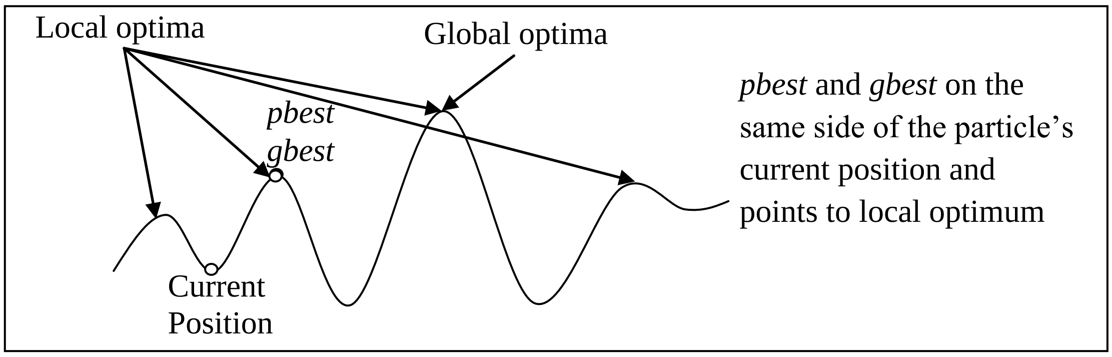
Local optima
Global optima
pbest and gbest on the same side of the particle's current position and points to local optimum
pbest
gbest
Current Position

Figure 2.6: Entrapment of particle in local optima

2.5 PSO method for Discrete Optimization Problems

The original PSO is simple and efficient. It has been successful in solving a number of continuous optimization problems [EK95]. However it is not applicable for a discrete search i.e. if the position of each particle is bound to a discrete set of values. In order to extend PSO to solve discrete optimization problem, a number of discrete particle swarm optimization algorithms have been proposed in literature. The first discrete version was proposed in [KE97]. It was based on binary coding scheme. The bottleneck with this approach was that the binary coding scheme has applications in limited types of optimization problems in the discrete space. In [CZ+10], various discrete PSO (DPSO) algorithms were classified into four types. The first type is the swap-operator-based DPSO algorithm proposed in [C04]. In this algorithm, the position of a particle is defined as a permutation of numbers. Velocity is any operator which, when applied to a position during an iteration, gives another position. Thus, it is a permutation of elements i.e. a list of transpositions. In [WH+03], a similar approach was applied for solving travelling salesman problem. The second type of DPSO algorithms are those which use space transformation techniques [SAA02] [SH06] [PW+04(a)]. In these algorithms, the position is defined as a real vector, and thus a space transformation technique is used to convert the position into its corresponding solution. The third types of DPSO algorithms define the position and velocity as a fuzzy matrix [PW+04(b)] [LTL07] [WW07]. These algorithms require a defuzzification method to decode the fuzzy matrix into a feasible solution to the problem. The fourth type comprises hybrid approaches. In these approaches, the pure PSO algorithm is integrated with some other meta-heuristics [WW+07] [AMR05]. In addition to the above approaches, there exist approaches that use some problem dependent local search techniques with standard PSO algorithms to solve specific problems.

In [CZ+10], Set-based Particle Swarm Optimization (S-PSO) algorithm was proposed. It was based on the concept of set theory and probability theory. This algorithm provides a more generalized framework and is applicable to a varied class of discrete optimization problems. In S-PSO, the velocity and position updating rules are similar to that of original PSO [EK95] except that all the related arithmetic operators used in these equations are redefined on crisp sets and sets with probabilities. The parameters used in the earlier PSO, the acceleration coefficients and the inertia weight play a similar role in S-PSO. In addition to the original PSO, different improved variants of the original PSO can be extended to their corresponding discrete versions following the representation scheme in S-PSO [CZ+10].

In this dissertation, the discrete version of the CLPSO algorithm, i.e. the set-based CLPSO is used to solve distributed query plan generation problem, discussed in Chapter 1, the set-based CLPSO is discussed next.

2.5.1 The Set-Based CLPSO

The set-based CLPSO algorithm is based on a problem of finding a subset from a universal set, which satisfies some constraints and optimizes a problem specific objective function. It defines the problem as defined in [LK73]. The set-based CLPSO (S-CLPSO) algorithm uses a representation scheme similar to set based representation scheme in [CZ+10]. S-CLPSO, as in [LK73], represents each candidate solution as a crisp subset X out of the universal set E. The universal set E can be divided into n dimensions, i.e. $E = E^1 \cup E^2 \cup \ldots \cup E^n$. A candidate solution to the problem X ⊆ E can also be divided into n dimensions, i.e. $X = X^1 \cup X^2 \cup \ldots \cup X^n$, where $X^j \subseteq E^j$. X satisfies the constraints Ω. The objective of the problem is to find a feasible solution X^* that optimizes *f*.

Velocity Updating: The velocity updating rule in S-CLPSO is the same as in **[**LS+06], i.e.

$$V_i^j \leftarrow \omega V_i^j + cr^j \left(PBest_{fi(j)}^j - X_i^j\right)$$

However positions, velocities and all related arithmetic operators in the above equations are redefined in the discrete space according to [CZ+10].

Position: Position is defined, according to [CZ+10], as a feasible solution to the problem. The position of the ith particle is X_i ($X_i \subseteq E$).The position is composed of n dimensions [CZ+10] as given under

$$Xi = Xi^1 \cup Xi^2 \cup \ldots \cup Xi^n \text{and } X^j \subseteq E^j (j=1, 2\ldots n)$$

Velocity: Velocity is defined as a set with possibilities. Given a crisp set E, a set with possibilities V defined on E is given by [CZ+10]

V = {e/p (e) |e ∈ E}, each element e ∈ E has a possibility p (e) ∈ [0, 1] in V.

In the jth dimension,

V_i^j= {e/p (e) |e ∈ E^j} is a set with possibilities defined on E^j.

Coefficient ×Velocity: The product of a coefficient c ($c \geq 0$) and velocity i.e. a set with possibilities V = {e/p (e) |e ∈ E} is defined as [CZ+10]

$$cV = \{e/p'(e) \mid e \in E\},$$

$$p'(e)=\left\{\begin{matrix} 1, & if\ c \times p(e) > 1 \\ c \times p(e), & otherwise \end{matrix}\right\}$$

Position – Position: The difference between two positions is defined by using the traditional definition of the minus operator between two crisp sets [CZ+10]. Given two crisp sets A and B, the relative complement A–B of B in A is given by [CZ+10]

$$A - B = \{e \mid e \in A \text{ and } e \notin B\}$$

Coefficient × (Position − Position): The multiplication operator between a coefficient c ($c \geq 0$) and a crisp set E′ (Position−Position) is defined as [CZ+10]

$$cE' = \{e/p'(e) \mid e \in E\},$$

$$p'(e)=\left\{\begin{matrix} 1, & if\ e \in E' and\ c > 1 \\ c, & if\ e \in E' and\ 0 \leq c \leq 1 \\ 0, & if\ e \notin E' \end{matrix}\right\}$$

Velocity + Velocity: The plus operator between two sets V_1 = {e/p_1 (e) |e ∈ E} and V_2 = {e/p_2 (e) |e ∈ E} with possibilities is defined as [CZ+10]

$$V_1 + V_2 = \{e/\max(p_1(e), p_2(e)) \mid e \in E\}$$

When the velocity of the particle V_i is updated, the particle adjusts its current position X_i to build a new position X_i'.

Position Updating: A new method was defined in [CZ+10] to update the position of a particle after its velocity has been updated. For this purpose, a particle learns from some elements of the updated velocity. First the set with probability V_i is converted into a crisp set. In each iteration, a random number α ∈ (0, 1) is generated for each particle. If the probability p (e) for each element e in the jth dimension is not smaller than α, element e is retained in the crisp set, i.e.

$$cut_\alpha(V_i^j) = \{e \mid e/p(e) \in V_i^j \text{ and } p(e) \geq \alpha\}$$

Now the particle *i* learns from the elements in $cut_\alpha(V_i^j)$ to build a new position. If the construction of new position X_i^j is not finished and there is no available element in $cut_\alpha(V_i^j)$, particle *i* reuses the elements in the previous X_i^j to build new X_i^j. The constraints Ω must be taken into account during the construction.

2.6 Query Plan Generation

S-CLPSO algorithm is used to solve the Distributed query plan generation problem. The algorithm considers a relation-site matrix that represents all the possible sites where a relation is available. For a given query, the relations accessed by the query are considered. Using the site-relation matrix, the sites where the relations accessed by the query reside are identified. Many possible query plans or combinations of site-relation may exist and each such combination represents a particle, which is represented as an ordered pair of relation-site combination. The universal set E consists of relations and all possible ordered pairs of each of them with the sites where they reside. Each query plan X_i is a subset of the universal set E, that is, X ⊆ E. X can also be divided into n dimensions, i.e. $X = X^1 \cup X^2 \cup \ldots \cup X^n$, where $X^j \subseteq E^j$. X is a feasible query plan only if it contains all the relations accessed by a query and each relation is selected from one of the sites from amongst all the sites where it resides in. The velocity of a particle (query plan) is the relation-site ordered pair and the randomly associated probability with it. The query plan generation algorithm based on S-CLPSO is given in Figure 2.7.

For the given user query, the algorithm first generates a universal relation set E for the relations accessed by the query (Step1). Next, the initial population of particles along with their velocities is randomly generated using the site-relation matrix (Step2).
The fitness of each particle (query plan) is computed using the (Query Processing Cost (QPC)) function given in [VSV11] (Step3). Initially the *pbest* value of any particle (query plan) is initialized to current position (Step4). Now, for each particle of the population ps, learning probability P_c is computed (Step6). Next, the velocity of all the particles of the population is updated (Step7). In order to update the velocity of a particle, for each of dimension of the particle (query plan), a random number is generated. If the random number is larger than its learning probability P_c^i then its corresponding dimension will learn from its own *pbest* otherwise it will learn from another particle's *pbest*. In the latter case, two particles are chosen randomly, excluding the particle whose velocity is being updated. The QPC values of their pbest are compared and the one with lower *pbest* value is considered as exemplar for the given particle. These steps are repeated for all dimensions of the given particle. After the

velocity has been updated, the position of the particle is accordingly updated. Next, QPC of the updated particle (query plan) is computed (step 8).The *pbest* value and topKQueryPlan are accordingly updated (step 9, Step10).These steps are repeated until a pre-specified number of generations are completed or no improvement is observed over a pre-specified number of generations (Step11). At the end, the top-K query plans are produced as output (Step12).

```
Input:      rsm: relation-site matrix
            ps : Population Size
            max_iter: Maximum number of iterations
            ω: Inertia weight // linearly decreasing from 0.9 to 0.4
            c: Cognitive acceleration constant (2.0)
Output:     TopkQueryPlan - Top K query plan
Method:
Step1:   Obtain the universal set E based on the available relation site matrix rsm.
Step2:   Generate initial particles (query plans) and their associated velocities randomly
           from the available relation site matrix equal to the Population size, ps.
Step3:   Calculate the fitness (query plan cost) of each particle.
             f=$QPC = \sum_{i=1}^{M} \frac{S_i}{N}\left(1 - \frac{S_i}{N}\right)$
         Where M is the number of sites accessed by the query plan,
             $S_i$ is the number of times the $i^{th}$ site is used in query plan,
             N is the number of relations accessed by the query.
Step4:   Set $pbest_i = X_i$ for all $1 \le i \le ps(swarm\ size)$
Step5:   For each particle of the swarm do steps 6, 7, 8, 9 and 10.
Step6:   Calculate learning probability ($Pc_i$)for the ith particle as:
             $Pc_i = 0.05 + 0.45 * \frac{\left(exp\left(\frac{10(i-1)}{ps-1}\right) - 1\right)}{(exp(10) - 1)}$
         Where, ps is the total number of particles in the swarm.
Step7:   For each dimension of the particle do steps a and b
           (a) Generate a random number ( $rand_i$)
           (b) IF   $rand_i > Pc_i$
                 Update position and velocity using $pbest_i$ ;
               ELSE
                 Choose two particles (p and q) randomly;
                 Compare the fitness values of their pbest and find the winner particle (say p);
                 Use the winner's pbest ($pbest_p$) as exemplar for the chosen dimension
                 Update position and velocity using $pbest_p$ ;
Step8:   IF a particle is an exemplar of itself on all dimensions
           Randomly choose one dimension to learn from the dimension of some other
           randomly chosen particle's pbest.
Step9:   Calculate the query plan cost of the updated particle
Step10:  Update the pbest of the particle
Step11:  Update Top K query plan
Step12:  IF (iteration < max_iter and not stagnated) GOTO Step 6.
Step13:  Return Top K query plan as TopkQueryPlan
```

Figure 2.7: Query Plan Generation Algorithm using S-CLPSO

2.7 An Example

Input:

A relation-site matrix (rsm) that represents eight relations R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8 distributed among eight sites S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8 is shown in Figure 2.8.

Relations\Sites	**S_1**	**S_2**	**S_3**	**S_4**	**S_5**	**S_6**	**S_7**	**S_8**
R_1	1	0	1	1	1	0	1	1
R_2	1	1	0	0	1	1	0	0
R_3	0	1	1	0	1	1	1	1
R_4	1	1	1	0	1	0	1	0
R_5	1	1	0	1	1	1	1	1
R_6	0	0	1	1	1	1	1	1
R_7	0	1	1	1	1	1	1	0
R_8	1	1	1	1	1	1	0	0

Figure 2.8: Relation-Site Matrix

Consider a query that accesses the relations R_3, R_4, R_7 and R_8

Let

Population Size (ps) =5,

Maximum number of iteration (max_iter) =20,

Inertial Weight ω=linearly decreasing from 0.9 to 0.4,

Cognitive acceleration constant c=2.0

Step 1: Using rsm, the universal set E is given by:

$$E = \bigcup_{i=1}^{8} E^i$$

Where

$$E^1 = \{(1,1),(1,3),(1,4),(1,5),(1,7),(1,8)\}$$

$$E^2 = \{(2,1),(2,1),(2,5),(2,6)\}$$

$$E^3 = \{(3,2),(3,3),(3,5),(3,6),(3,7),(3,8)\}$$

$$E^4 = \{(4,1),(4,2),(4,3),(4,5),(4,7)\}$$

$$E^5 = \{(5,1),(5,2),(5,4),(5,5),(5,6),(5,7),(5,8)\}$$

$$E^6 = \{(6,3),(6,4),(6,5),(6,6),(6,7),(6,8)\}$$

$$E^7 = \{(7,2),(7,3),(7,4),(7,5),(7,6),(7,7),(7,9)\}$$

$$E^8 = \{(8,1),(8,2),(8,3),(8,4),(8,5),(8,6)\}$$

Step 2: The randomly generated initial particles (query plans) along with their associated velocities are given in Figure 2.9.

Particle (i)	Position X_i	Velocity V_i
1	{(3,2),(4,3),(7,2),(8,3)}	{(3,2) /.2176,(4,1) /.0815,(7,4) /.0519,(8,1)
2	{(3,5),(4,2),(7,2),(8,2)}	{(3,5) /.3906,(4,2) /.0086,(7,7) /.1176,(8,4)
3	{(3,3),(4,7),(7,6),(8,6)}	{(3,5) /.6670,(4,3) /.0498,(7,5) /.4397,(8,2)
4	{(3,2),(4,7),(7,4),(8,3)}	{(3,3) /.9206,(4,5) /.3756,(7,5) /.6369,(8,6)
5	{(3,6),(4,7),(7,6),(8,4)}	{(3,2) /.0878,(4,2) /.9573,(7,4) /.3035,(8,2)

Figure 2.9: Initial particles and their velocities

Step 3: The fitness defined as the query processing cost of each particle is computed, using the formula of QPC [50] given below, is shown in Figure 2.10

Particles (i)	$f_i = QPC_i = \sum_{i=1}^{M} \frac{S_i}{N}\left(1 - \frac{S_i}{N}\right)$
1	0.5000
2	0.3750
3	0.6250
4	0.7500
5	0.6250

Figure 2.10: Initial QPC of particles

Step 4: Initially for each of the five particles (query plans) X_1, X_2, X_3, X_4 and X_5

$pbest_i = X_i$, where i=1 to 5

Step 5: Velocity update for the first particle (query plan) X_1 is computed as:

Learning Probability (Pc_1)for the particle X_1 is computed using the equation:

$$Pc_i = 0.05 + 0.45 * \frac{\left(exp\left(\frac{10(i-1)}{ps-1}\right) - 1\right)}{(exp(10) - 1)}$$

Pc_1 = 0.0500

For each dimension $j(1 \le j \le 4)$ a random number is generated and compared with the learning probability Pc_1 to choose the particles amongst X_1, X_2, X_3, X_4 and $X_5$from whose *pbest* X_1 has to learn from.

Suppose for *j*=1, random number generated is 0.3900, which is greater than Pc_1. Thus, the first dimension would learn from *pbest* of particle X_1 and thus X_1 will learn from its own *pbest* i.e. $f_1(1)$= 1

$$pbest^1_{f_1(1)} = pbest^1_1$$

$$\left(pbest^1_1 - x^1_1\right) = \{(3, 2)\text{-}(3, 2\} = [\emptyset\}$$

Suppose for j=2, random number generated is 0.0342, which is less than Pc_1. Thus two particles X_5 and X_2 are randomly generated. Since the fitness of X_2 i.e. QPC_2 is 0.3750 is less than QPC_5 0.6250 of X_5, particle X_2 is selected i.e. $f_1(2)$= 2

$$pbest^2_{f_1(2)} = pbest^2_2$$

$$\left(pbest^2_2 - x^2_1\right) = \{(4, 2)\text{-}(4, 3)\} = \{(4,2)\}$$

Suppose for j=3, random number generated is 0.4941, which is greater than Pc_1. Thus, the third dimension would learn from *pbest* of particle $X_1$and thus X_1will learn from its own *pbest* i.e.$f_1(3)$= 1

$$pbest^3_1 = pbest^3_1$$

$$\left(pbest^3_1 - x^3_1\right) = \{(7, 2)\text{-}(7, 2)\} = \{\emptyset\}$$

Suppose for j=4, random number generated is 0.2602, which is greater than Pc_1. Thus, the fourth dimension would learn from *pbest* of particle $X_1$and thus X_1will learn from its own *pbest* i.e. $f_1(4)$= 1

$$pbest^4_1 = pbest^4_1$$

$$\left(pbest^4_1 - x^4_1\right) = \{(8, 3)\text{-}(8, 3)\} = \{\emptyset\}$$

Next, the velocity of the particle $X_1$is updated using the velocity update equation:

$$v^j_i \leftarrow \omega v^j_i + cr^j \left(pbest^j_{f_i(j)} - x^j_i\right)$$

For this, the inertia component value of particle X_1 for each dimension is computed using the following rule:

The product of a coefficient c ($c \geq 0$) and velocity i.e. a set with possibilities V = {e/p (e) |e ∈ E} is defined as:

$$cV = \{e/p'(e) \mid e \in E\},$$

$$p'(e)=\begin{Bmatrix} 1, & if c \times p(e) > \\ c \times p(e), & otherwise \end{Bmatrix}$$

The inertia component value for each dimension is shown in Figure 2.11

ω	v^j_1	ωv^j_1
0.9	$v^1_1 = \{(3,2)/.2176\}$	$\omega v^1_1 = \{(3,2)\ /.1958\}$
0.9	$v^2_1 = \{(4,1)/.0815\}$	$\omega v^2_1 = \{(4,1)\ /.0734\}$
0.9	$v^3_1 = \{(7,4)/.0519\}$	$\omega v^3_1 = \{(7,4)\ /.0467\}$
0.9	$v^4_1 = \{(8,1)\ /.0587\}$	$\omega v^4_1 = \{(8,1)\ /.0528\}$

Figure 2.10: Updated inertia component

Thus the inertia of the particle X_1 is

$$\omega V_1 = \{(3,2)\ /.1958, (4,1)\ /.0734, (7,4)\ /.0467, (8,1)\ /.0528\}$$

Next, cognitive component values of particle $X_1$for each dimension are computed using the following rule:

The multiplication operator between a coefficient c ($c \geq 0$) and a crisp set E′ (Position−Position) is defined as:

$$cE' = \{e/p'(e) | e \in E\},$$

$$p'(e) = \begin{cases} 1, & if e \in E' and c > 1 \\ c, & if e \in E' and\ 0 \leq c \leq 1 \\ 0, & if e \notin E' \end{cases}$$

The cognitive component computation for each dimension of particle X_1 is shown in Figure 2.12.

c	r^j	$\left(pbest^j_{f_i(j)} - x^j_i\right)$	$c\ \boxed{?}\ ^{\boxed{?}}\left(pbest^j_{f_i(j)} - x^j_i\right)$
2.0	0.3245	$\{\emptyset\}$	$\{\emptyset\}$
2.0	0.4665	$\{(4,2)\}$	$\{(4,2)/0.9331$
2.0	0.0564	$\{\emptyset\}$	$\{\emptyset\}$
2.0	0.9323	$\{\emptyset\}$	$\{\emptyset\}$

Figure 2.11: Updated cognitive component

Thus the cognitive component of the particle is

$$c * rand * \left(pbest_{f_1} - x_1\right) = \{(4, 2)\ /.9331\}$$

Now, new updated velocity is computed using the following rule:

The plus operator between two sets $V_1 = \{e/p_1(e)\ |e \in E\}$ and $V_2 = \{e/p_2(e)\ |e \in E\}$ with possibilities is defined as:

$$V_1 + V_2 = \{e/\max(p_1(e), p_2(e))\ |e \in E\}$$

$$V_1' = \omega V_1 + c * rand * \left(pbest_{f_1} - x_1\right)$$

$$= \{(3,2)\ /.1958, (4,1)\ /.0734, (7,4)\ /.0467, (8,1)\ /.0528\} + \{(4, 2)\ /.9331\}$$

$$= \{(3, 2)\ /0.1958, (4, 2)\ /0.9331, (7, 4)\ /0.0467, (8, 1)\ /0.0528\}$$

Next, the position of the particle is updated using the updated velocity. The current position $X_1 = \{(3, 2), (4, 3), (7, 2), (8, 3)\}$ is updated to a new positionX_1' in the following manner

$$X_i' \leftarrow \text{position updating } (X_i, V_i)$$

First, the set with possibilities V_i is converted into a crisp set. For each dimension, a random number $\alpha \in (0, 1)$ is generated for each particle. For each element *e* in the jth dimension, if it's corresponding possibility p (e) in Vi^j is not smaller than α, element e is reserved in a crisp set, that is:

$cut_\alpha(V_i^j)$ = {e| e/p (e) $\in V_i^j$ and p (e) ≥ α}.

The crisp set for V_1 is shown in Figure 2.13

j	α	**p(e)**	**Comparison between α and p(e)**
1	0.7356	0.1958	p(e) ≤ α
2	0.0421	0.9331	p(e) ≥ α
3	0.1259	0.0467	p(e) ≤ α
4	0.6421	0.0528	p(e) ≤ α

Figure 2.12: Crisp Set for V_1

Elements *e* reserved in a crisp set: $cut_\alpha(V_1)$ = {(4, 2)}. So, element {(4, 2)} would be used for updating the current position {(3, 2), (4, 3), (7, 2), (8, 3)} of particle X_1.

The new updated Position would have the relation R_4 accessed from site S_2 i.e.

$X_1' =$ {(3, 2), (4, 2), (7, 2), (8, 3)}

The fitness value (QPC) of X_1' is $f_1' = 0.3750$

Similarly, the velocity and position for other three particles are updated. The updated position and velocity of particles X_1, X_2, X_3, X_4 and X_5 are given in Figure 2.14.

i	**Updated Position (X_i')**	**Updated Velocity (V_i')**
1	{(3, 2), (4, 2), (7, 2), (8, 3)}	{(3, 2)/0.1958,(4,2)/0.9331,(7, 4)/0.0467,(8,1)/0.0528}
2	{(3, 5), (4, 2), (7, 2), (8, 6)}	{(3, 5)/0.3515,(4,2)/0.0078,(7, 7)/0.1058,(8,6)/0.4737}
3	{(3, 5), (4, 7), (7, 5), (8, 2)}	{(3, 5)/0.6003,(4,3)/0.0448,(7, 5)/0.3957,(8,2)/0.4945}
4	{(3, 3), (4, 5), (7, 5), (8, 3)}	{(3, 3)/0.8286,(4,5)/0.3380,(7, 5)/0.5732,(8,6)/0.5249}
5	{(3, 6), (4, 7), (7, 6), (8, 4)}	{(3, 2)/0.0790,(4,2)/0.8615,(7, 4)/0.2732,(8,2)/0.4646}

Figure 2.13: Updated Position and Velocity

The updated QPC of the particles X_1, X_2, X_3, X_4 and X_5 are shown in Figure 2.15

i	**Updated QPC (f_i')**
1	0.6250
2	0.3750
3	0.6250
4	0.5000
5	0.6250

Figure 2.14: Updated QPC

In a similar manner, the position and velocities are updated of the particles X_1, X_2, X_3, X_4 and X_5 for a pre-specified number of iterations. The top-4 query plans generated after 20 iterations is shown in Figure 2.16

Query Plans	**QPC**
{(3, 2), (4, 2), (7, 2), (8, 2)}	0.0000
{(3, 2), (4, 2), (7, 2), (8, 3)}	0.3750
{(3, 6), (4, 7), (7, 6), (8, 6)}	0.3750
{(3, 3), (4, 5), (7, 5), (8, 3)}	0.5000

Figure 2.15: Top-4 query plans

2.8 Experimental Results

The GA based query plan generation algorithm and S-CLPSO based query plan generation algorithm are implemented in MATLAB 7.4 in Windows XP environment. The two algorithms were compared by conducting experiments on an Intel based 2 GHz PC having 1 GB RAM. The comparisons were carried out on parameters like number of relations, average query processing cost (QPC), top-K query plans and number of iterations.

First, line graphs were plotted to compare S-CLPSO and GA based algorithms on Average QPC against the number of iterations for selecting top-10 query plans. These graphs for the number of relations n=6, 8, 10, 12 and 14 are shown in figures 2.16, 2.17, 2.18, 2.19 and 2.20 respectively. Line graphs for different pairs of crossover and mutation probabilities (GA(P_c, P_m)) were plotted.

It can be observed from the graphs that the GA based algorithm (crossover probability 0.6 and mutation probability 0.05), in case of 6 and 8 relations, is able to generate Top-10 query plans having almost equal average QPC. Whereas, for higher number of relations, i.e. for 10, 12, 14 relations the S-CLPSO based algorithm is able to generate top-10 query plans with relatively lower average QPC. So, it can be said that, as the number of relations in the query increases, the S-CLPSO based algorithm, in comparison to GA based algorithm, is able to generate relatively better query plans with respect to the cost of query processing.

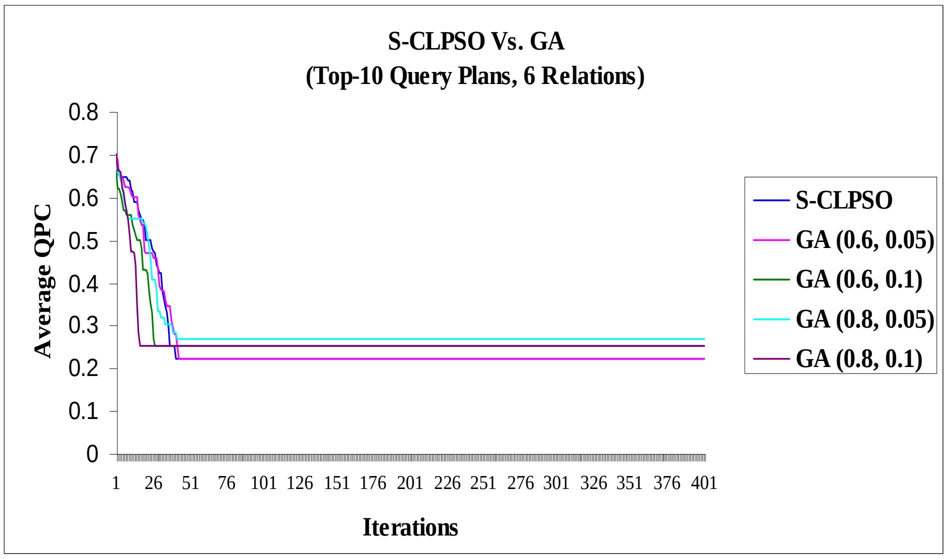

Figure 2.16: S-CLPSO vs. GA – Average QPC vs. Iterations (Top-10 Query Plans, 6 Relations)

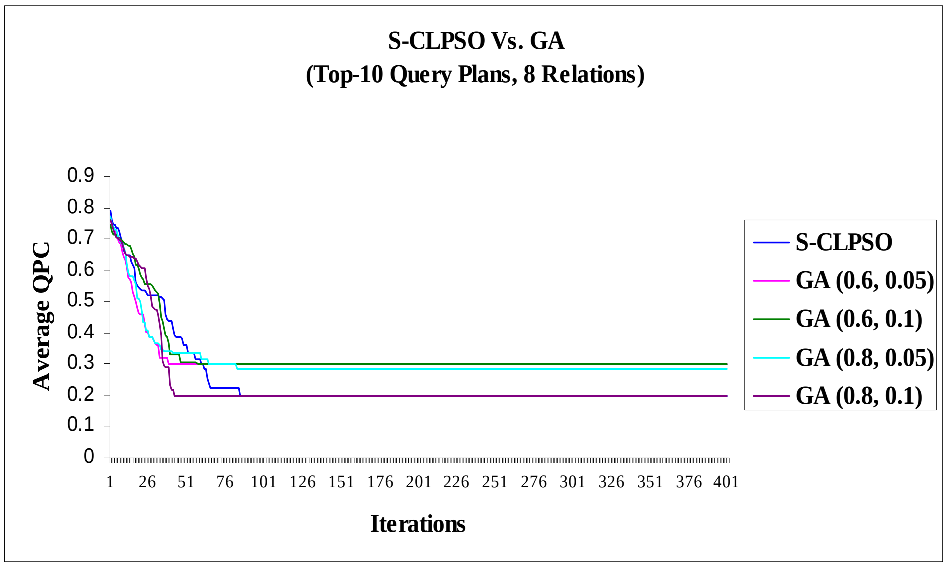

Figure 2.17: S-CLPSO vs. GA – Average QPC vs. Iterations (Top-10 Query Plans, 8 Relations)

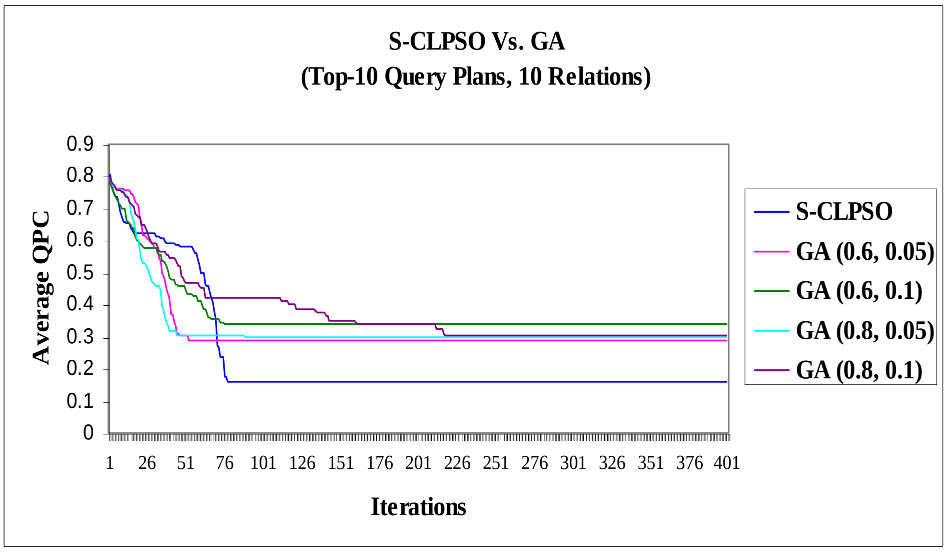

Figure 2.18: S-CLPSO vs. GA – Average QPC vs. Iterations
(Top-10 Query Plans, 10 Relations)

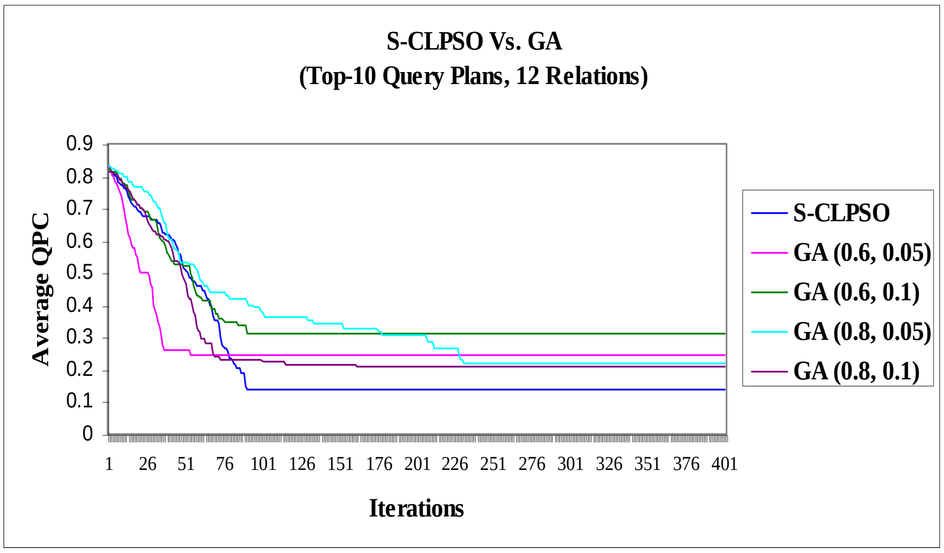

Figure 2.19: S-CLPSO vs. GA – Average QPC vs. Iterations
(Top-10 Query Plans, 12 Relations)

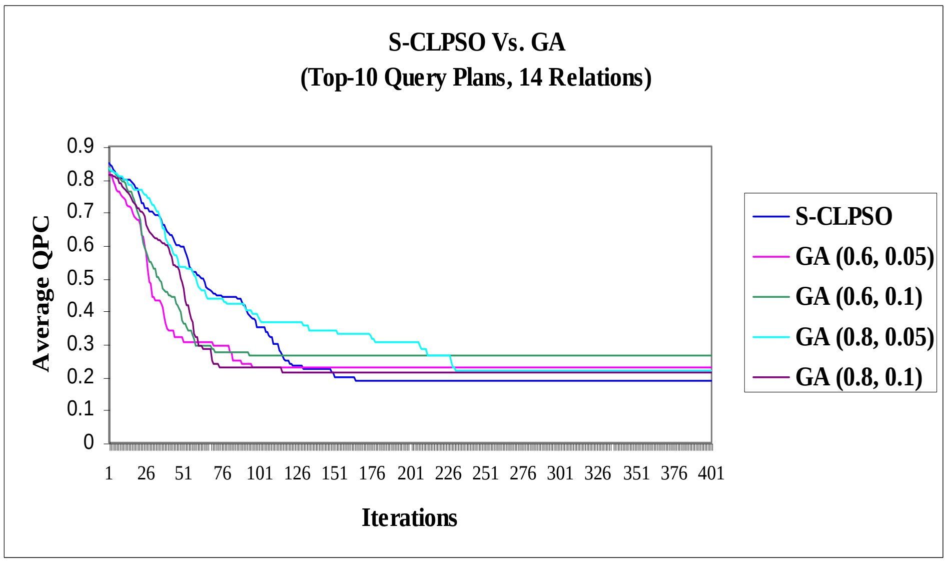

Figure 2.20: S-CLPSO vs. GA – Average QPC vs. Iterations (Top-10 Query Plans, 14 Relations)

Next, graphs were plotted to compare S-CLPSO and GA based algorithms on Average QPC value for selecting Top-K query plans(K=6, 8, 10, 12, 14) generated after 400 iterations. These graphs, plotted for relations 6, 8, 10, 12 and 14, are shown in figures 2.21, 2.22, 2.23, 2.24 and 2.25 respectively. These graphs show that S-CLPSO generate Top-K query plans with almost equal average QPC for 6 and 8 relations but is able to generate Top-K query plans with relatively lower average QPC for 10, 12 and 14 relations. Thus, it can be said that for higher number of relations, S-CLPSO is able to generate good quality Top-K plans with relatively lower average QPC.

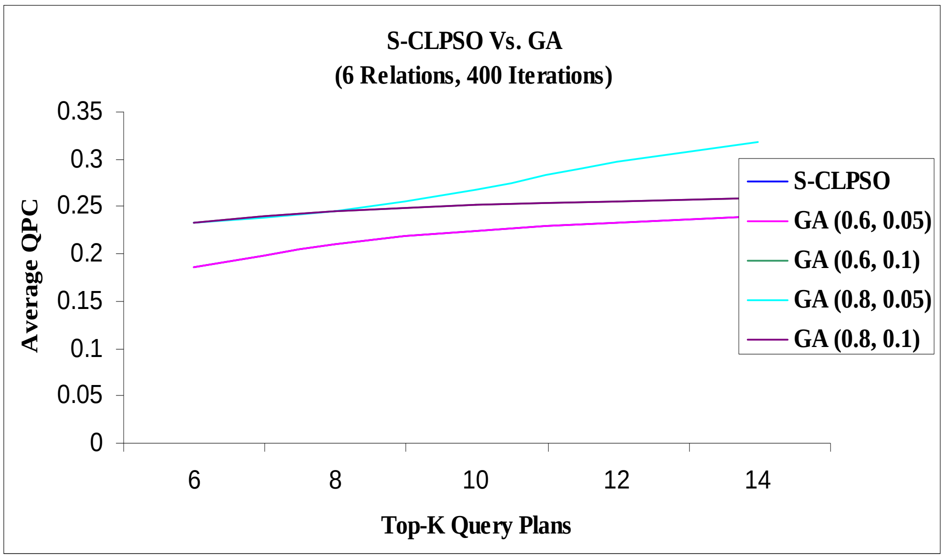

Figure 2.21: S-CLPSO vs. GA – Average QPC vs. Top-K Query Plans (6 Relations, 400 iterations)

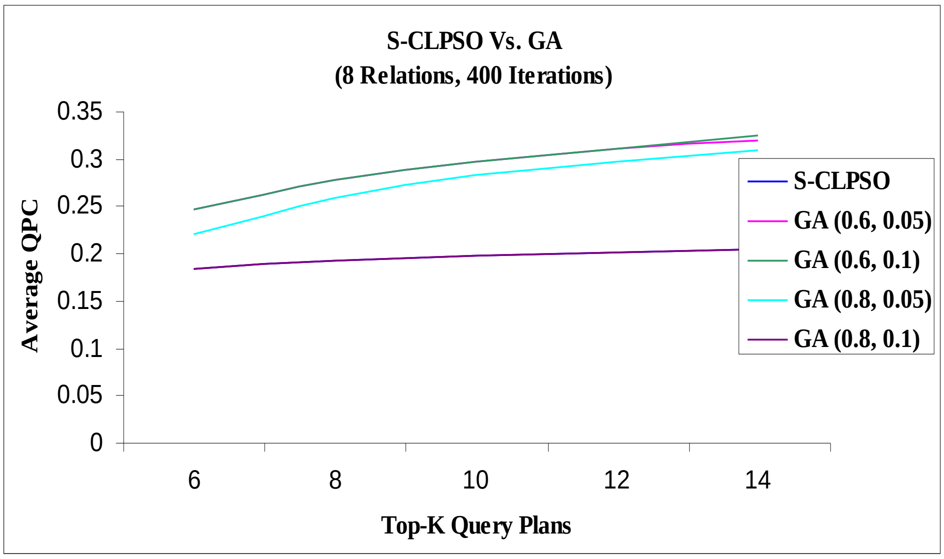

Figure 2.22: S-CLPSO vs. GA – Average QPC vs. Top-K Query Plans (8 Relations, 400 iterations)

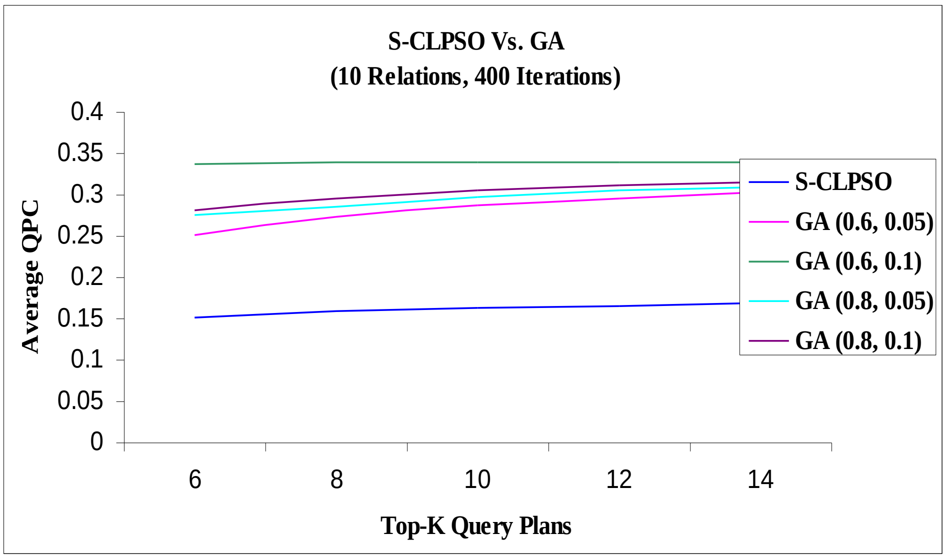

Figure 2.23: S-CLPSO vs. GA – Average QPC vs. Top-K Query Plans (10 Relations, 400 iterations)

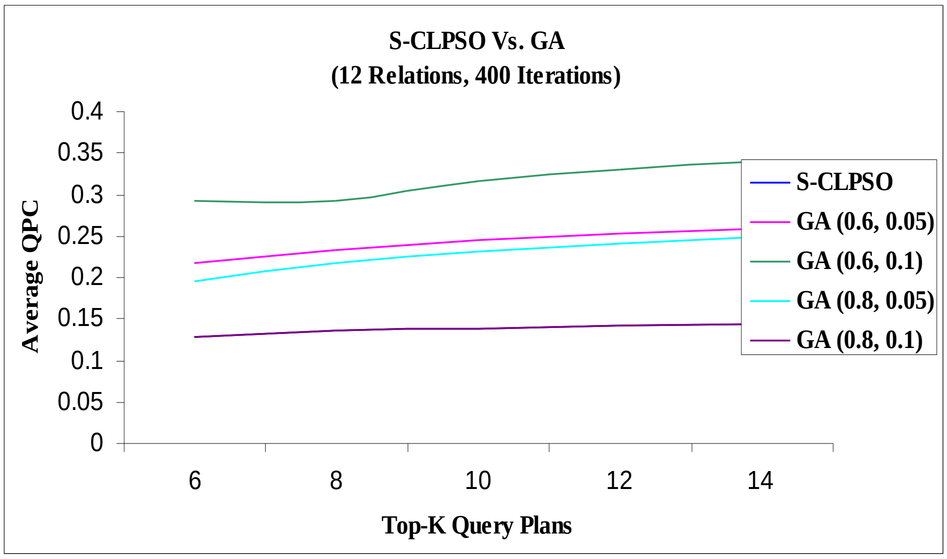

Figure 2.24: S-CLPSO vs. GA – Average QPC vs. Top-K Query Plans (12 Relations, 400 iterations)

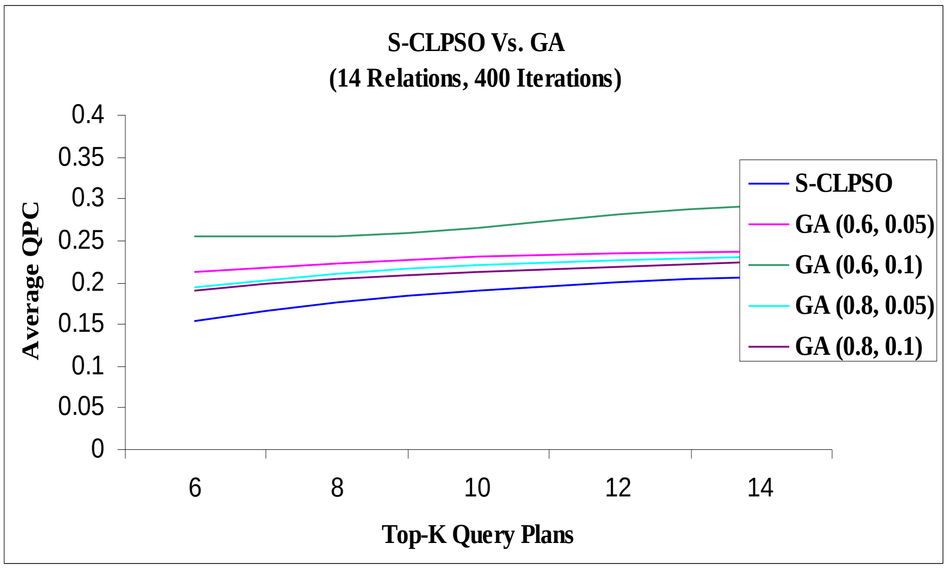

Figure 2.25: S-CLPSO vs. GA – Average QPC vs. Top-K Query Plans (14 Relations, 400 iterations)

CHAPTER 3

Distributed Query Plan Generation Using MOPSO

In this day and age, ubiquitously, it is observed that it is atypical for any problem to involve only a single value or objective. In order to find any holistic solution to the problem, there is a need to optimize various objectives or parameters [U8]. Maximizing profit and minimizing the cost of a product, maximizing performance and minimizing fuel consumption of a vehicle and minimizing weight while maximizing the strength of a particular component are some of the few examples of multi-objective optimization problems [U3]. In today's world, Multi-objective optimization problems can be found by and large in various fields, e.g. product and process design, automobile design, aircraft design and finance etc [U3].

3.1 Multi-Objective Optimization

Multi-Objective optimization is defined in [U9] as "the process of simultaneously optimizing two or more conflicting and /or incommensurable objectives subject to certain constraints". In these problems, optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives e.g. maximizing profit and minimizing the cost of a product [U3].

If it is assumed that the objectives are to be minimized, the Multi-Objective optimization problem can be expressed mathematically as [U3]:

$$\min_{x}[\mu_1(x), \mu_2(x), \dots, \mu_n(x)]^T$$

s.t.

$$g(x) \leq 0$$

$$h(x) = 0$$

$$x_l \leq x \leq x_u$$

Where μ_i is the i^{th} objective function, g and h are the inequality and equality constraints, respectively, and *x* is the vector of optimization or decision variables.
The objective functions, $\mu_i(x)$ may be conflicting with each other and thus the detection of a single global minimum is impossible. So, instead of achieving a unique solution to the problem, the solution would be a set of Pareto optimal points [U9]. Solutions are said to be Pareto optimal if no objective can be improved without sacrificing at least another objective [SAA02]. Suppose $u = (u_1, u_{2\,,} \dots, u_k)^T$ and $v = (v_1, v_2, \dots, v_k)^T$ are two k-dimensional vectors then the following are defined as [PV10]:
Pareto Dominance: - The vector u is said to dominate vector v, if and only if the following holds:

$$u_i \leq v_i, for\ all\ i = 1,2, \dots, k, and, u_i < v_i, for\ at\ least\ one\ component\ i.$$

This property is known as Pareto dominance [PV10].
Pareto Optimality: - A solution, x ∈ A, of the multi-objective problem is said to be Pareto optimal, if and only if there is no other solution, y ∈ A, such that f(y) dominates f(x). Alternatively, it can be said that x is non-dominated with respect to A. Here $A \subset R^n$ is n-dimensional search space [PV10].
Pareto optimal set:-The set of all Pareto optimal solutions is called Pareto optimal set, P*[PV10].

Pareto Front:-The set of vector function values of all Pareto optimal solutions is called Pareto Front [PV10].

$$PF^* = \{ f(x) : x \in P^* \}$$

Pareto optimal surface or the Pareto Front can be represented graphically as shown in figure 3.1 [U14]. In this figure, *Pareto optimality* in the bi-objective case is illustrated. Here, points *A* and *B* are non-dominated solutions residing on the Pareto front. Neither is better than the other. Point *A* has a smaller value of f_2 than point *B*, but a larger value of f_1. Likewise, point *B* has a smaller value of f_1 than point *A*, but a larger value of f_2. Solution *A* and solution *B* are not dominated by any other solution on the Pareto front, or Pareto-optimal surface. There exists no solution which has a better value with respect to both the objective functions f_1 and f_2 [U14].

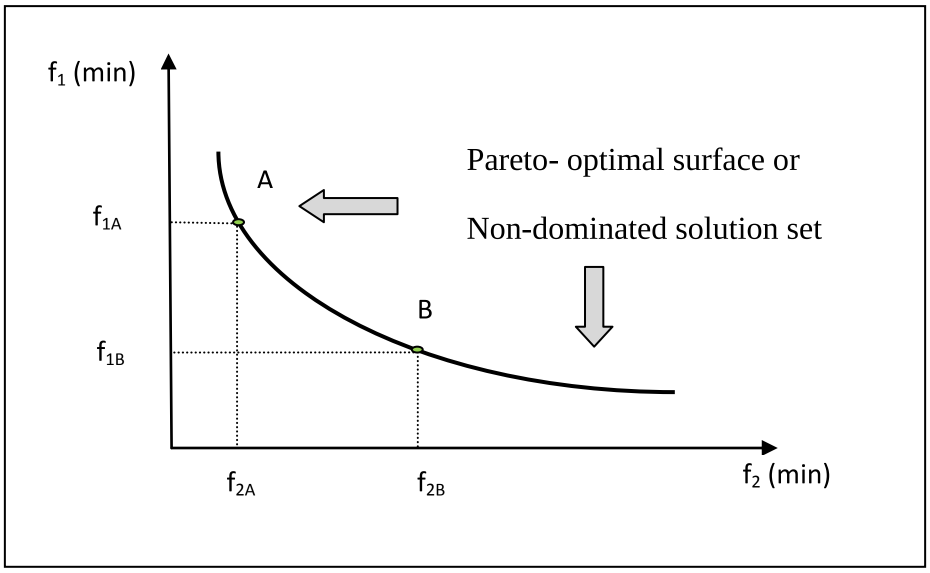

Figure 3.1: Pareto-optimal surface for a bi-objective problem [U14]

Multi-Objective optimization aims at "maximizing the number of elements of the Pareto optimal set found, minimizing the distance of the Pareto front produced by the algorithm with respect to the true (global) Pareto front and maximizing the spread of solutions found, so as to have a distribution of vectors as smooth and uniform as possible"[ZDT00]. Additionally, the goal is to achieve and monitor convergence towards true Pareto-front in order to avoid local convergence. The Pareto optimal fronts for bi-objective problems are illustrated in Figure 3.2.

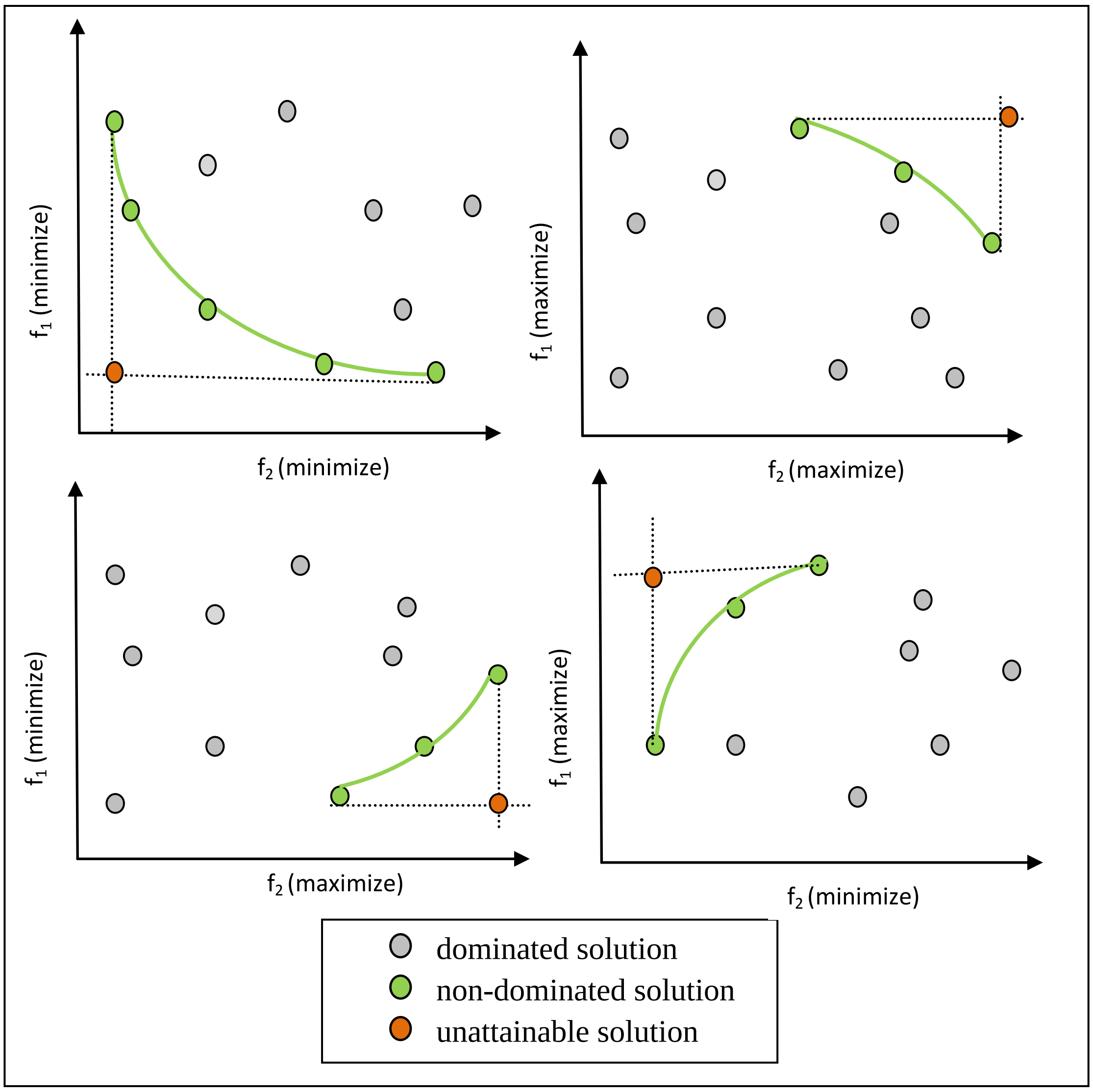

Figure 3.2: Examples of Pareto Optimal sets with Two Objective Functions

3.2 Particle Swarm Optimization for Multi-Objective Problems

It is generally observed that particle swarm optimization has a high speed of convergence when applied to a single-objective problem. This particular feature has been a motivation behind it being used for solving multi-objective problems [KE01]. Multi-objective particle swarm optimization (PSO) can be divided into two categories [RC06 (a)] namely PSO Variants that exploit each objective function separately and variants based on Pareto Dominance. The former is referred here as Non-Pareto based approaches and the latter is referred to as Pareto based approaches. These are discussed next.

3.2.1 Non-Pareto Based Approaches

This category consists of two types of approaches [PV10] namely approaches that combine all objective functions in a single one, referred here as Aggregated Approaches, and approaches that consider each objective function in turn for the evaluation of particles, referred here as Non-Aggregated Approaches. These approaches are discussed next:

3.2.1.1 Aggregated Approaches

These are the approaches that combine all the objectives of the problem into a single objective. In other words, the multi-objective problem is converted into a single-objective problem [RC06 (a)].

$$F(x) = \sum_{i=1}^{k} w_i f_i(x),$$

Where w_i are non-negative weights, such that:

$$\sum_{i=1}^{k} w_i = 1.$$

If weights are fixed during a run then it is called conventional weighted aggregation (CWA). This algorithm has to be applied repeatedly with different weight settings to detect a desirable number of non-dominated solutions with only a single solution attained per run. Moreover, CWA is unable to detect solutions in concave regions of the Pareto front. In order to resolve these limitations dynamically adjusted weights, such as bang-bang weighted aggregation (BWA) were suggested in [JOS01]. For bi-objective problems, the weights in BWA are adapted as follows:

$$w_1(t) = sign\big(sin(2\pi t/a)\big), \qquad w_2(t) = 1 - w_1(t),$$

Where a is user-defined adaptation frequency, and t stands for the iteration number.

The sign function used in BWA causes the weights to change abruptly so that the algorithm continues to move towards the Pareto front. Another approach proposed in [JOS01] is the dynamic weighted aggregation (DWA), the weights in DWA are adapted as follows:

$$w_1(t) = |sin(2\pi t/a)|, \qquad w_2(t) = 1 - w_1(t).$$

DWA performs better in comparison to BWA in problems involving convex Pareto fronts. However, in case of problems involving concave Pareto fronts, the performance of both the techniques is more or less the same. Another multi-objective particle swarm optimization algorithm based on weighted aggregation approach was proposed in [PV02 (b)] [PV02 (a)] to solve bi-objective problems. This algorithm uses all three types of aggregating functions: CWA, BWA, and DWA. In [BMR04] a similar approach that uses linear aggregation functions was proposed. In this approach the whole swarm in divided into various sub-swarms, each of the sub-swarm uses a different set of weights. The best particle of each sub-swarm is used as a leader to guide other members of the sub-swarm. In [MCL04], another aggregating approach based on dynamically modified weights was proposed.

3.2.1.2 Non-Aggregated Approaches

In these approaches, each particle is evaluated only with one objective function at a time, and the best positions are determined following the standard single-objective PSO rules, using the corresponding objective function. These can be further categorized as lexicographic ordering approaches and sub-population based approaches.

Lexicographic Ordering Approaches: These approaches are based on ranking schemes that determine the importance of each objective function and rank the objectives accordingly. In order to obtain the optimum solution, the objective functions are minimized one by one. The most important objective function is minimized at the outset and then minimization procedure continues according to the assigned order of importance of the objectives [M99]. However, this approach is sensitive to the ordering of the objectives and is useful only when there are few objective functions [C99]. In [HE02] another similar ordering scheme was proposed where each objective function is optimized separately. The best positions of the particles are stored as non-dominated solutions. In this scheme external archive is not used. However, in a later version of this approach [HES03], an external archive was incorporated.

Sub-Population Based Approaches: These approaches use various sub-populations, which exchange information among themselves. In [PTV04], a vector evaluated PSO (VEPSO) scheme was proposed. This scheme employs one swarm per objective function and evaluation is carried out with this objective function while best positions of one swarm are used to update velocities of another swarm with a different objective function. In [CT04], a scheme similar to VEPSO was proposed called multi-species PSO. This scheme uses various sub-swarms, where each such sub-swarm is evaluated with one objective function. Information of best particles in a swarm is communicated to neighboring sub-swarms.

3.2.2 Pareto Based Approaches

In these approaches the concept of Pareto dominance plays an important role. In order to guide the particles, some elite particles that are non-dominated solutions with respect to the swarm are used. Furthermore some other decisive factors such as swarm diversity and Pareto front spread are also taken into account. In [MC99], an algorithm based on Pareto dominance was presented in an unpublished document. In this approach, both an individual and a group search are performed simultaneously. The major drawback of this algorithm was that it did not adopt any scheme to maintain diversity [MC99]. In [RL02] a new algorithm was proposed that uses Pareto dominance and combines concepts of evolutionary techniques with the particle swarm optimization technique. This approach uses crowding distance to maintain diversity and a multilevel sieve to handle constraints. In [FS02], another approach was presented that made use of an unconstrained elite archive to store the non-dominated individuals found along the search process. This archive interacts with the primary population in order to define local guides. Similar schemes were put forward in [CL02] [CPL04]. In [CPL04], the proposed MOPSO stored the non-dominated solutions in an archive also referred to as repository. In addition, the search space is divided in hypercubes where each hypercube is assigned a fitness value, inversely proportional to the number of particles in it. Roulette wheel selection is then used to select a hypercube and a leader from it. In [MT03], a sigma method was recommended in which the best local guides for each particle are adopted to improve the convergence and diversity of a MOPSO approach. The use of the sigma values increases the selection pressure of the algorithm. A hybrid approach called non-dominated sorting particle swarm optimizer was given in [L03]. This approach incorporates the main mechanisms of the NSGA-II [DP+02] in a PSO algorithm. In [PC04], a new approach was presented that uses the concept of Pareto dominance to decide the flight direction of a particle. This algorithm divides the population of particles into several sub-swarms by using clustering techniques so as to provide a better distribution of solutions in decision variable space. In each sub-swarm, a PSO algorithm is executed by using own set of leaders. These different sub-swarms exchange information. In this approach, no external archive is used since the migration of leaders among the sub-swarms ensures elitism. In [RC05], another approach based on Pareto dominance that uses nearest neighbor density estimator in order to select leaders for the particles was proposed. As opposed to other typical approaches, this algorithm makes use of two external archives. One external archive is used to store the leaders currently used for performing the flight and another for storing the final solutions. The concept of dominance is used to select the particles that will remain in the archive of final solutions. In order to retain a fixed number of non-dominated solutions (leaders) in the archive, the density estimator factor is used. In [RN05], multi-objective particle swarm optimization based on Crowding Distance (MOPSO-CD) was proposed. This approach uses crowding distance to facilitate the selection of global best particle. It is also used to delete non-dominated solutions from the external archive. Mutation is utilized in order to maintain diversity of non-dominated solutions in the archive. Several other approaches of multi-objective particle swarm optimization have been reported in [RC06 (a)] [RC06 (b)] [PV08] [ZZ10]. In [ZZ10], a parallel particle swarm optimization (PPSO) algorithm was presented to solve the multi-objective optimization problems.

Parallel Particle Swarm Optimization (PPSO) algorithm exploits the intrinsic parallel characteristics of the PSO algorithm to solve multi-objective problems. The basic idea is to exploit as many swarms as the number of objectives in the multi-objective problem. Each of these swarms use the same evolutionary mechanism and simultaneously optimizes objectives assigned to them [ZZ10]. These different swarms communicate and share information among them through an external archive, which stores the non-dominated solutions found by different swarms. The velocity is updated using the following equation [ZZ10]:

$$V_{id}^{m} = \omega V_{id}^{m} + c_1 r_{1d}(P_{id}^{m} - X_{id}^{m}) + c_2 r_{2d}(G_{d}^{m} - X_{id}^{m}) + c_3 r_{3d}(A_{id}^{m} - X_{id}^{m})$$

Where d is the dimension, and the position update is as:

$$X_{id}^{m} = X_{id}^{m} + V_{id}^{m}$$

In the velocity update equation, the term $c_3 r_{3d}(A_{id}^{m} - X_{id}^{m})$ represents the sharing information from the archive that influences the particle to fly along the Pareto front. The term A_i^m is a non-dominated solution selected by the i^{th} particle in the m^{th}swarm randomly picked from the archive.

The achieve *A is* initially empty and is updated in every generation. There is a limit on the number of non-dominated solutions the archive can store i.e. the archive has a maximum size (Max_Arch). At any particular iteration, the *pbest* of each particle in each swarm and the solutions of the archive from the previous iteration are used to select Max_Arch number of non-dominated solutions. In order to select this fixed number of non-dominated solutions, non-dominated sorting algorithm [DP+02] is used. PPSO algorithm achieves the goal of multi-objective optimizations by ensuring population diversity, so that a good number of non-dominated solutions can be obtained. It avoids the difficulty associated with fitness assignment [DP+02]. Due to sharing of search information through the external archive, the swarms are not attracted to the margin of Pareto front [ZZ10]. According to [ZZ10], since PPSO uses each swarm to optimize a single objective, any standard or improved PSO [ZZ+09] [ZZ+10] can be used to solve a single objective problem. In this dissertation, Set Based Comprehensive Learning PSO (S-CLPPSO) algorithm has been used to solve the multi-objective distributed query plan generation problem given in Chapter 1. This algorithm is discussed next.

3.2.2.1 S-CLPPSO

S-CLPPSO algorithm uses as many swarms as the number of objectives in the multi-objective problem. Each swarm optimizes only one of the objectives using S-CLPSO. These swarms work in parallel. An external archive is used to store the non-dominated solutions found by different swarms in each iteration. The swarms share search information and communicate with each other through the external archive. The evolutionary mechanisms in each swarm are similar, just like a standard S-CLPSO.

In every generation, for the m^{th}swarm, the velocity of each particle *i* is updated as:

$$V_{id}^{m} = \omega\, V_{id}^{m} + c_1 r_{1d}^{m}\left(pbest_{f_i(d)}^{m} - X_{id}^{m}\right) + c_3 r_{3d}\left(A_{id}^{m} - X_{id}^{m}\right)$$

Where d is the dimension and the term $c_3 r_{3d}\left(A_{id}^{m} - X_{id}^{m}\right)$ represents the share information from the archive. A_i^m is a non-dominated solution, which is randomly selected by the i^{th} particle in the m^{th}swarm, stored in the archive. The representation scheme and the interpretation of all other operators are same as in S-CLPSO discussed in chapter 2. Subsequently, the updated velocity is used to update the position of the particle. The archive A, which is initially empty, is updated after each iteration in order to keep a fixed number of non-dominated solutions (Max_Arch). The archive is updated in the following manner. First, a set S is initialized as empty. Then the *pbest* of each particle in each swarm is added into the set S. Then all the solutions in the old archive A are added into the set S. Later, non-dominated sorting strategy [DP+02] is performed on solutions in the set S to find Max_Arch solutions, which are then stored in the archive. The query plan generation algorithm, presented in this dissertation, is based on S-CLPPSO. This algorithm is discussed next.

3.3 Query Plan Generation

The multi-objective problem comprises of optimizing the two objectives namely Minimizing Site Communication Cost (SCC) and Maximizing the Relation Concentration Gain (RCG) as discussed in chapter 1. In this case, two objectives have to be optimized, so two swarms S_1 and S_2 are used. Each swarm optimizes only one of the objectives using S-CLPSO. These swarms work in parallel. An external archive is used to store the non-dominated solutions found by both the swarms in each iteration. The two swarms share search information and communicate with each other through the external archive.

The algorithm considers a relation-site matrix that comprises of relations and their respective sites. For a given query, the relations accessed by the query are considered. Using the relation-site matrix, sites where the relations accessed by the query reside are identified. Many possible query plans or combinations of site-relation may exist and each such combination represents a particle, which is represented as an ordered pair of relation-site combination. The universal set E consists of relations and all possible ordered pairs of each of them with the sites where they reside. Each query plan X_i is a subset of the universal set E, that is, X ⊆ E. X can also be divided into n dimensions, i.e. $X = X^1 \cup X^2 \cup ... \cup X^n$, where $X^j \subseteq E^j$. X is a feasible query plan only if it contains all the relations accessed by a query and each relation is selected from one of the sites from amongst all the sites where it resides in. The velocity of a particle (query plan) is the relation-site ordered pair and the randomly associated probability associated with it. The query plan generation algorithm based on S-CLPPSO is given in Figure 3.3

```
Input:
    rsm: relation-site matrix
    ps : Population size in each of the two swarms
    max_iter: Maximum number of iterations
    ω: Inertia weight // linearly decreasing from 0.9 to 0.4
    $c_1$: Cognitive acceleration constant (2.0)
    $c_3$: Inter swarm communication constant (2.0)
Output:
    TopkQueryPlan - Top K query plan
Method:
Step1:  Obtain the universal set E based on the available relation site matrix rsm.
Step2:  For each swarm generate initial particles (query plans) and their associated velocities randomly
        from the available relation site matrix equal to the Population size, ps.
Step3:  For the two swarms, compute the objective function values:
        Swarm 1
        $SCC = m \times s$
            Where s is the number of sites being used and m is the number of communications
        Swarm 2
        $RCG = \sum_{i=1}^{s}(n - i + 1)c_i$
            Where n is the number of relations in the query and $c_i$ is the count of sites arranged in
            decreasing order.
Step4:  For each swarm, set $pbest_i = X_i$ for all $1 \leq i \leq ps(swarm\ size)$
Step5:  Set external archive size (EXA_size) =k;
        Initialize EXA= { };
Step6:  For each swarm do step 7
Step7:  For each particle of the swarm do steps 8, 9, 10, 11 and 12.
Step8:  Compute learning probability ($Pc_i$) for the ith particle as:
        $$Pc_i = 0.05 + 0.45 * \frac{\left(exp\left(\frac{10(i-1)}{ps-1}\right) - 1\right)}{(exp(10) - 1)}$$
        Where, ps is the total number of particles in the swarm.
Step9:  For each dimension of the particle do steps a and b
          (a) Generate a random number ( $rand_i$)
          (b) IF  $rand_i > Pc_i$
                  Update position and velocity using $pbest_i$ ;
              ELSE
                  Choose two particles (p and q) randomly;
                  Compare the fitness values of their pbest and find the winner particle (say p);
                  Use the winner's pbest ($pbest_p$) as exemplar for the chosen dimension
                  Update position and velocity using $pbest_p$ ;
Step10: If a particle is an exemplar of itself on all dimensions
            Randomly choose one dimension to learn from the dimension of some other randomly
            chosen particle's pbest.
Step11: Compute the respective cost of the updated particle
Step12: Update the pbest of the particle
Step13: Update the external archive (EXA)
Step14: IF (iteration < max_iter AND not stagnated)
                GOTO Step 6.
Step15: Return Top K query plan as TopkQueryPlan
```

Figure 3.3: Query Plan Generation Algorithm using S-CLPPSO

For the given user query, the algorithm first generates a universal relation set E for the relations accessed by the query (Step1). Next, the initial population of particles along with their velocities is randomly generated using the site-relation matrix in both the swarms. (Step2).

The cost of each particle (query plan), with respect to the objective of the swarm, is computed using the respective objective functions (SCC or RCG) given in chapter 1(Step3). Initially the *pbest* value of any particle (query plan) is initialized to current position in both the swarm (Step4). The size of the external archive is set to be *k* (where *k* denotes the number of top query plans to be generated) and the archive is initialized as empty set (Step 5). Now, in both the swarms, for each particle of the population ps, learning probability P_c is computed, Thereafter the velocity and position of all the particles of the population are updated (Step 6, 7, 8, 9 and 10). Next, the respective cost value of the updated particle (query plan) is computed in both the swarms (step 11). The *pbest* values of the particles in both the swarms are also updated (Step12). The external archive is also updated using non-dominated sorting (Step 13). These steps are repeated until a pre-specified number of iterations are completed or no improvement is observed over a pre-specified number of iterations (Step14). At the end, the top-K query plans are produced as output from the external archive (Step15).

3.4 An Example

Input:

A relation-site matrix (rsm) that represents eight relations R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8 distributed among eight sites S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8 is shown in Figure 3.4.

Relations\Sites	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
R_1	1	0	1	1	1	0	1	1
R_2	1	1	0	0	1	1	0	0
R_3	0	1	1	0	1	1	1	1
R_4	1	1	1	0	1	0	1	0
R_5	1	1	0	1	1	1	1	1
R_6	0	0	1	1	1	1	1	1
R_7	0	1	1	1	1	1	1	0
R_8	1	1	1	1	1	1	0	0

Figure 3.4: Relation-Site Matrix

Consider a query that accesses relations R_3, R_4, R_7 and R_8. The objective is to generate top-4 query plans.

Let

Population Size in each of the two swarms (ps) =5,

Maximum number of iteration (max_iter) =20,

Inertial Weight ω=linearly decreasing from 0.9 to 0.4,

Cognitive acceleration constant c_1=2.0

Inter swarm communication constant c_2 =2.0

Step 1: Using matrix rsm, the universal set E is given by:

$$E = \bigcup_{i=1}^{8} E^i$$

Where

$$E^1 = \{(1,1),(1,3),(1,4),(1,5),(1,7),(1,8)\}$$

$$E^2 = \{(2,1),(2,1),(2,5),(2,6)\}$$

$$E^3 = \{(3,2),(3,3),(3,5),(3,6),(3,7),(3,8)\}$$

$$E^4 = \{(4,1),(4,2),(4,3),(4,5),(4,7)\}$$

$$E^5 = \{(5,1),(5,2),(5,4),(5,5),(5,6),(5,7),(5,8)\}$$

$$E^6 = \{(6,3),(6,4),(6,5),(6,6),(6,7),(6,8)\}$$

$$E^7 = \{(7,2),(7,3),(7,4),(7,5),(7,6),(7,7),(7,9)\}$$

$$E^8 = \{(8,1),(8,2),(8,3),(8,4),(8,5),(8,6)\}$$

Step 2: The randomly generated initial particles (query plans) along with their associated velocities for swarm1 and swarm2 are given in Figure 3.5 and in Figure 3.6 respectively.

Particle (i)	Position X_i	Velocity V_i
1	{(3,5),(4,5),(7,4),(8,6)}	{(3,6) /.5097,(4,1) /.7418,(7,3) /.4612,(8,6) /.4151}
2	{(3,7),(4,7),(7,5),(8,2)}	{(3,6) /.6320,(4,5) /.7626,(7,2) /.8225,(8,3) /.9805}
3	{(3,2),(4,3),(7,4),(8,2)}	{(3,5) /.3935,(4,7) /.0632,(7,6) /.8355,(8,6) /.5607}
4	{(3,7),(4,1),(7,3),(8,4)}	{(3,2) /.4397,(4,3) /.7157,(7,7) /.5093,(8,6) /.5296}
5	{(3,5),(4,3),(7,5),(8,4)}	{(3,2) /.8641,(4,5) /.4011,(7,6) /.6800,(8,5) /.3404}

Figure 3.5: Initial particles and their velocities in Swarm 1

Particle (i)	Position X_i	Velocity V_i
1	{(3,7),(4,7),(7,6),(8,5)}	{(3,7) /.9403,(4,5) /.0664,(7,6) /.0121,(8,6) /.1747}
2	{(3,2),(4,7),(7,4),(8,2)}	{(3,3) /.3335,(4,3) /.2677,(7,2) /.5165,(8,2) /.6427}
3	{(3,6),(4,5),(7,5),(8,2)}	{(3,6) /.6206,(4,5) /.0334,(7,5) /.9865,(8,2) /.3556}
4	{(3,3),(4,7),(7,4),(8,4)}	{(3,3) /.9796,(4,7) /.2852,(7,3) /.6300,(8,2) /.6187}
5	{(3,2),(4,1),(7,3),(8,2)}	{(3,7) /.1266,(4,3) /.4364,(7,6) /.3924,(8,2) /.7773}

Figure 3.6: Initial particles and their velocities in Swarm 2

Step 3: The objective function values for the two swarms are given in Figure 3.7 and Figure 3.8 respectively

Particles (i)	$SCC = m \times s$
1	6
2	6
3	6
4	12
5	6

Figure 3.7: fitness values of particles in Swarm1

Particles (i)	$RCG = \sum_{i=1}^{s}(n-i+1)c_i$
1	13
2	13
3	13
4	13
5	13

Figure 3.8: fitness values of particles in Swarm2

Step 4: For each particle in each of the two swarms

$$pbest_i = X_i \text{ for all } 1 \le i \le ps(swarm\ size)$$

Step5: External archive size (EXA_size) =4

Initially the external archive is empty i.e. EXA= { }

Step 6: Next, the velocity and position of each particle in each of the two swarms is updated using the following rule:

$$V_{id}^{m} = \omega V_{id}^{m} + c_1 r_{1d}^{m}\left(pbest_{f_i(d)}^{m} - X_{id}^{m}\right) + c_3 r_{3d}\left(A_{id}^{m} - X_{id}^{m}\right)$$

Since the external archive is empty initially, the above equation is modified to:

$$v_i^j \leftarrow \omega v_i^j + cr^j\left(pbest_{f_i(j)}^j - x_i^j\right)$$

First, a random number $r \in [0, 1]$ is generated. If r is larger than parameter P_c then $f_i(j)$ = i. Otherwise, the algorithm applies the tournament selection to randomly select two particles. The particle with a better fitness value is selected as $f_i(j)$.In this sense, $pbest^{j}_{f_{i(j)}}$can be the corresponding dimension of any particle's *pbest* position. S-CLPSO algorithm is applied to particles in swarm1 and particles in swarm2 in the same manner as was applied in solving the single objective problem in Chapter 2. The updated position, updated velocity and the respective cost value of particles in swarm1 and swarm2 are given in Figure 3.9 and 3.10 respectively.

i	Updated Position (X_i')	Updated Velocity (V_i')	Updated SCC
1	{(3, 5), (4, 1), (7, 5), (8, 6)}	{(3, 6)/0.8463,(4,1)/0.0598,(7, 5)/0.2379,(8,6)/0.1572}	6
2	{(3, 7), (4, 7), (7, 4), (8, 3)}	{(3, 6)/0.3001,(4,5)/0.2409,(7, 4)/1.000,(8,3)/0.5785}	6
3	{(3, 2), (4, 3), (7, 6), (8, 4)}	{(3, 5)/0.5585,(4,7)/0.0301,(7, 6)/0.8879,(8,4)/1.000}	12
4	{(3, 2), (4, 1), (7, 4), (8, 4)}	{(3, 2)/0.8816,(4,3)/0.2567,(7, 4)/1.000,(8,6)/0.5568}	6
5	{(3, 5), (4, 5), (7, 6), (8, 6)}	{(3, 2)/0.1139,(4,2)/0.3927,(7, 4)/0.3532,(8,2)/0.6996}	2

Figure 3.9: Updated position and velocities after first iteration in Swarm 1

i	Updated Position (X_i')	Updated Velocity (V_i')	Updated RCG
1	{(3, 7), (4, 7), (7, 6), (8, 5)}	{(3, 7)/0.9403,(4,5)/0.0664,(7, 3)/0.0121,(8,6)/0.1747}	13
2	{(3, 2), (4, 7), (7, 4), (8, 2)}	{(3, 3)/0.3335,(4,3)/0.2677,(7, 6)/0.5135,(8,2)/0.6427}	13
3	{(3, 6), (4, 5), (7, 5), (8, 2)}	{(3, 6)/0.6206,(4,5)/0.0334,(7, 5)/0.9865,(8,2)/0.3556}	13
4	{(3, 3), (4, 7), (7, 4), (8, 2)}	{(3, 3)/0.9796,(4,7)/0.2852,(7, 5)/0.6300,(8,2)/0.6187}	10
5	{(3, 2), (4, 3), (7, 3), (8, 2)}	{(3, 2)/0.1266,(4,2)/0.4364,(7, 4)/0.3924,(8,2)/0.7773}	14

Figure 3.10: Updated position and velocities after first iteration in Swarm 2

After first iteration the non-dominated particles in the External Archive are:

EXA= {{(3,5),(4,5),(7, 6),(8,6)}, {(3,2),(4,3),(7,3),(8,2)},
{(3,5),(4,1),(7,5),(8,6)}, {(3,7),(4,7),(7,4),(8,3)}}

Since after the first iteration the external archive contains four non-dominated solutions, the velocity and position of each particle in each of the two swarms is updated using the following rule:

$$V^{m}_{id} = \omega V^{m}_{id} + c_1 r^{m}_{1d}\left(pbest^{m}_{f_i(d)} - X^{m}_{id}\right) + c_3\, r_{3d}\left(A^{m}_{id} - X^{m}_{id}\right)$$

Iteration 2: Swarm1: First Particle

Position after first iteration

$X_1' = \{(3,5), (4,1), (7,5), (8,6)\}$

Velocity after first iteration

$V_1' = \{(3, 6)/0.8463, (4, 1)/0.0598, (7, 5)/0.2379, (8, 6)/0.1572\}$

$pbest_1 = \{(3,5), (4,5), (7,4), (8,6)\}$

Next, the velocity of the first particle X_1' is updated using the velocity update equation:

$$V_{id}^m = \underbrace{\omega V_{id}^m}_{\text{Inertia Component}} + \underbrace{c_1 r_{1d}^m \left(pbest_{f_i(d)}^m - X_{id}^m\right)}_{\text{Cognitive Component}} + \underbrace{c_3 r_{3d}\left(A_{id}^m - X_{id}^m\right)}_{\text{External Archive Component}}$$

Computation of Inertia component

The inertia component value of particle X_1' for each dimension is computed using the following rule:

The product of a coefficient c ($c \geq 0$) and velocity i.e. a set with possibilities V = {e/p (e) |e ∈ E} is defined as:

$$cV = \{e/p'(e) \mid e \in E\},$$

$$p'(e) = \begin{cases} 1, & if c \times p(e) > 1 \\ c \times p(e), & otherwise \end{cases}$$

The value of the inertia weight ω for the second iteration is calculated using the following rule:

$$\omega' = \omega_{max} - (\omega_{max} - \omega_{min}) \times \left(\frac{cureent\ iteration}{maximum\ iteration}\right)$$

$$\omega' = 0.9 - (0.9 - 0.4) \times \left(\frac{2}{20}\right) = 0.85$$

The inertia component value for each dimension is shown in Figure 3.11

ω'	v_1^j	$\omega' v_1^j$
0.85	$v_1^1 = \{(3,6)/.8463\}$	$\omega' v_1^1 = \{(3,6)\ /.7193\}$
0.85	$v_1^2 = \{(4,1)/.0598\}$	$\omega' v_1^2 = \{(4,1)\ /.0508\}$
0.85	$v_1^3 = \{(7,5)/.2379\}$	$\omega' v_1^3 = \{(7,5)\ /.2022\}$
0.85	$v_1^4 = \{(8,6)\ /.1572\}$	$\omega' v_1^4 = \{(8,6)\ /.1336\}$

Figure 3.11: Updated inertia component

Thus the inertia of the particle X_1' is

$$\omega' V_1' = \{(3,6)\ /.7193, (4,1)\ /.0508, (7,5)\ /.2022, (8,1)\ /.1336\}$$

Computation of Cognitive component

In order to compute the cognitive component values Learning Probability (Pc_1) for the particle X$_1'$ is computed using the equation:

$$Pc_i = 0.05 + 0.45 * \frac{\left(exp\left(\frac{10(i-1)}{ps-1}\right) - 1\right)}{(exp(10) - 1)}$$

Pc_1 = 0.0500

For each dimension $j(1 \leq j \leq 4)$, a random number is generated and compared with the learning probability Pc_1 to choose particles amongst X$_1'$ X$_2'$ X$_3'$ X$_4'$ and X$_5'$ from whose *pbest* X$_1'$ has to learn from.

Suppose for *j*=1, random number generated is 0.9542, which is greater thanPc_1. Thus, the first dimension would learn from *pbest* of particle X$_1'$ and thus X$_1'$ will learn from its own *pbest* i.e. f_1(1)= 1

$$pbest^1_{f_1(1)} = pbest^1_1$$

$$\left(pbest^1_1 - x^1_1\right) = \{(3, 5)\text{-}(3, 5\} = \{\emptyset\}$$

Suppose for *j*=2, random number generated is 0.6502, which is greater thanPc_1. Thus, the second dimension would learn from *pbest* of particle X$_1'$ and thus X$_1'$ will learn from its own *pbest* i.e. f_1(1)= 1

$$pbest^1_{f_1(1)} = pbest^1_1$$

$$\left(pbest^1_1 - x^1_1\right) = \{(4, 5)\text{-}(3, 1\} = \{(4,5)\}$$

Suppose for j=3, random number generated is 0.0301, which is less than Pc$_1$. Thus two particles X$_2'$ and X$_4'$ are randomly chosen. The fitness values of *pbest* of both the particles are shown in Figure 3.12

Particles	***pbest*$_i$**	**SCC$_i$**
X$_2'$	{(3,7),(4,7),(7,5),(8,2)}	6
X$_4'$	{(3,2),(4,3),(7,4),(8,2)	6

Figure 3.12: fitness values of *pbest* of particles

Since the fitness value of both the particles is same, any one of the two particles can be selected as an exemplar. Suppose particle X$_2'$ is selected i.e. f_1(3)= 2

$$pbest^3_{f_1(3)} = pbest^3_2$$

$$\left(pbest^3_2 - x^3_1\right) = \{(7, 5)\text{-}(7, 5)\} = \{\emptyset\}$$

Suppose for j=4, random number generated is 0.2602, which is greater than $Pc_{1.}$ Thus, the fourth dimension would learn from *pbest* of particle X₁′and thus X₁′ will learn from its own *pbest* i.e. $f_1(4)$= 1

$$pbest_1^4 = pbest_1^4$$

$$\left(pbest_1^4 - x_1^4\right) = \{(8, 6)\text{-}(8, 6)\} = \{\emptyset\}$$

Next, cognitive component values of particle X₁′ for each dimension are computed using the following rule:

The multiplication operator between a coefficient c (c ≥ 0) and a crisp set E′ (Position−Position) is defined as:

$$cE' = \{e/p'(e) | e \in E\},$$

$$p'(e) = \begin{Bmatrix} 1, & if e \in E' and c > 1 \\ c, & if e \in E' and\ 0 \leq c \leq 1 \\ 0, & if e \notin E' \end{Bmatrix}$$

The cognitive component computation for each dimension of particle X₁′ is shown in Figure 3.13.

c_1	r^j	$\left(pbest_{fi(j)}^j - x_i^j\right)$	$cr^j\left(pbest_{fi(j)}^j - x_i^j\right)$
2.0	0.3235	{∅}	{∅}
2.0	0.6665	{(4,5)}	{(4,5)/1.000}
2.0	0.0264	{∅}	{∅}
2.0	0.2323	{∅}	{∅}

Figure 3.13: Updated cognitive component

Thus the cognitive component of the particle is

$$c_1 r_1\left(pbest_{f_1} - x_1\right) = \{(4, 5)\ /1.0000\}$$

Computation of External Archive component

In order to compute the External Archive component for each dimension j $(1 \leq j \leq 4)$, the particle chooses an exemplar *m* from the external archive and that particular dimension learns from chosen element.

After the first iteration the external archive contains the following four non-dominated solutions as shown in figure 3.14:

Particle (i)	Positions
1	{(3,5),(4,5),(7, 6),(8,6)}
2	{(3,2),(4,3),(7,3),(8,2)}
3	{(3,5),(4,5),(7,4),(8,6)}
4	{(3,7),(4,7),(7,5),(8,2)}

Figure 3.14: external archive

Current position, X_1' = {(3, 5), (4, 1), (7, 5), (8, 6)}

The External archive component computation for each dimension of particle X_1' is shown in Figure 3.15.

j	c_3	r_3	m	A_1^m	$\left(A_{ij}^m - X_{ij}^m\right)$	$c_3\, r_3\left(A_{ij}^m - X_{ij}^m\right)$
1	2.0	0.3263	4	{(3,7),(4,7),(7,5),(8,2)}	{(3,7)} −{(3,5)}={(3,7)}	{(3,7)/0.6526}
2	2.0	0.2626	2	{(3,2),(4,3),(7,3),(8,2)}	{(4,3)} −{(4,1)}={(4,3)}	{(4,3)/0.5252}
3	2.0	0.2162	4	{(3,7),(4,7),(7,5),(8,2)}	{(7,5)} −{(7,5)}={∅}	{∅}
4	2.0	0.9749	1	{(3,5),(4,5),(7,6),(8,6)}	{(8,6)} −{(8,6)}= {∅}	{∅}

Figure 3.15: updated External archive components

Thus, the External archive component of the particle is:

$$c_3 r_3\left(A_i^m - X_i^m\right) = \{(3, 7)/0.6526, (4, 3)/0.5252\}$$

Now, new updated velocity is computed using the following rule:

The plus operator between two sets V_1 = {e/p1 (e) |e ∈ E} and V_2 = {e/p2 (e) |e ∈ E} with possibilities is defined as:

$$V_1 + V_2 = \{e/\max(p1(e), p2(e)) \mid e \in E\}$$

Using the three components computed above, the updated velocity after the second iteration is computed as:

$$V_1^{''} = \omega V_1^{'} + c_1 r_1\left(pbest_{f_1} - x_1\right) + c_3 r_3\left(A_i^m - X_i^m\right)$$

$$\omega V_1' = \{(3,6)\ /.7193, (4,1)\ /.0508, (7,5)\ /.2022, (8,1)\ /.1336\}$$

$$c_1 r_1\left(pbest_{f_1} - x_1\right) = \{(4, 5)\ /1.0000\}$$

$$c_3 r_3\left(A_i^m - X_i^m\right) = \{(3, 7)/0.6526, (4, 3)/0.5252\}$$

Thus, the updated velocity of the particle after the second iteration is:

$$V_1^{''} = \{(3,6)\ /.7193, (4,5)\ /1.000, (7,5)\ /.2022, (8,1)\ /.1336\}$$

Next, the position of the particle is updated using the updated velocity. The current position, X_1' = {(3, 5), (4, 1), (7, 5), (8, 6)} is updated to a new position $X_1^{''}$ in the following manner:

First, the set with possibilities $V_1^{''}$ is converted into a crisp set. For each dimension, a random number α ∈ (0, 1) is generated for each particle. For each element e in the jth dimension, if it's corresponding possibility p (e) in Vi^j is not smaller than α, element e is reserved in a crisp set, that is:

$$cut_\alpha(V_i^j) = \{e|\ e/p\ (e) \in V_i^j \text{ and } p\ (e) \geq \alpha\}.$$

The crisp set for V_1 is shown in Figure 3.16

j	α	**p(e)**	**Comparison between α and p(e)**
1	0.1356	0.7193	p(e) ≥ α
2	0.9421	1.000	p(e) ≥ α
3	0.7259	0.2022	p(e) ≤ α
4	0.4421	0.1336	p(e) ≤ α

Figure 3.16: Crisp Set for V_1''

Elements e reserved in a crisp set cut_α(V_1) = {(3, 6), (4, 5)}. So, elements {(3, 6), (4, 5)} would be used for updating the current position {(3, 5), (4, 1), (7, 5), (8, 6)} of particle X_1'. The new updated Position would have the relation R_3 accessed from site S_6 and R_4 accessed from site S_5 i.e. $X_1^{''}$ = {(3, 6), (4, 5), (7, 5), (8, 6). Similarly, the velocity and position for other particles are also updated in both the swarms. The above procedure continues till a pre-specified twenty iterations. The non-dominated solutions from the external archive are reported as the top-4 query plans.

Query Plans
{(3, 5), (4, 5), (7, 5), (8, 5)}
{(3, 6), (4, 5), (7, 6), (8, 6)}
{(3, 5), (4, 3), (7, 5), (8, 5)}
{(3, 5), (4, 5), (7, 3), (8, 5)}

Figure 3.17: Top-4 query plans

3.5 Experimental Results

The S-CLPSO based query plan generation algorithm and S-CLPPSO based query plan generation algorithm are implemented in MATLAB 7.4 in Windows XP environment. The two algorithms were compared by conducting experiments on an Intel based 2 GHz PC having 1 GB RAM. The comparisons were carried out on parameters like number of relations, average query processing cost (QPC), top-K query plans and number of iterations.

First, line graphs were plotted to compare S-CLPSO and S-CLPPSO based algorithms on Average QPC against the number of iterations for selecting top-10 query plans. These graphs for the number of relations n=6, 8, 10, 12 and 14 are shown in figures 3.18, 3.19, 3.20, 3.21 and 3.22 respectively.

It can be observed from the graphs that the S-CLPSO based algorithm, in case of 6, 8 and 10 relations, is able to generate Top-10 query plans having almost equal average QPC. Whereas, for higher number of relations, i.e. for 12 and 14 relations, the S-CLPPSO based algorithm is able to generate top-10 query plans with relatively lower average QPC. So, it can be said that, as the number of relations in the query increases, the S-CLPPSO based algorithm, in comparison to S-CLPSO based algorithm, is able to generate relatively better query plans with respect to the cost of query processing.

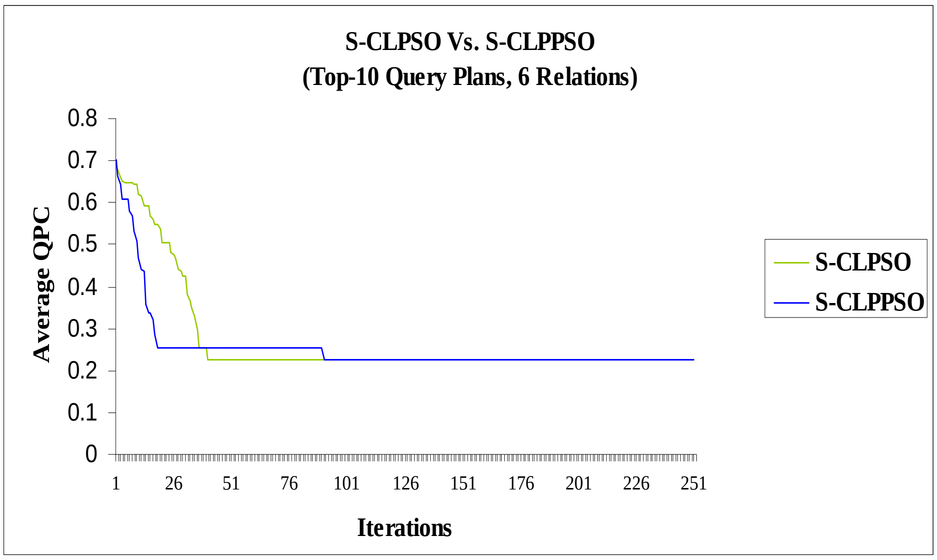

Figure 3.18: S-CLPSO vs. S-CLPPSO – Average QPC vs. Iterations (Top-10 Query Plans, 6 Relations)

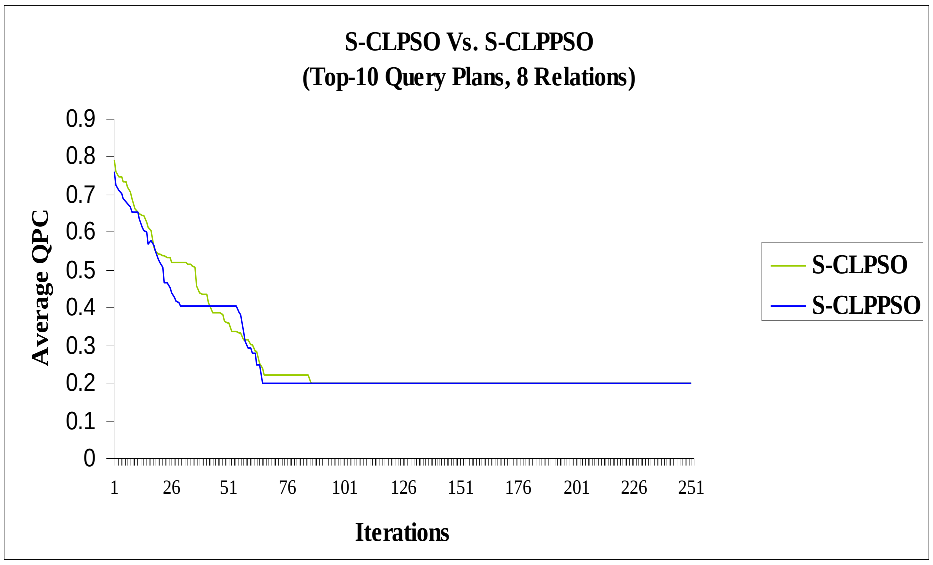

Figure 3.19: S-CLPSO vs. S-CLPPSO – Average QPC vs. Iterations
(Top-10 Query Plans, 8 Relations)

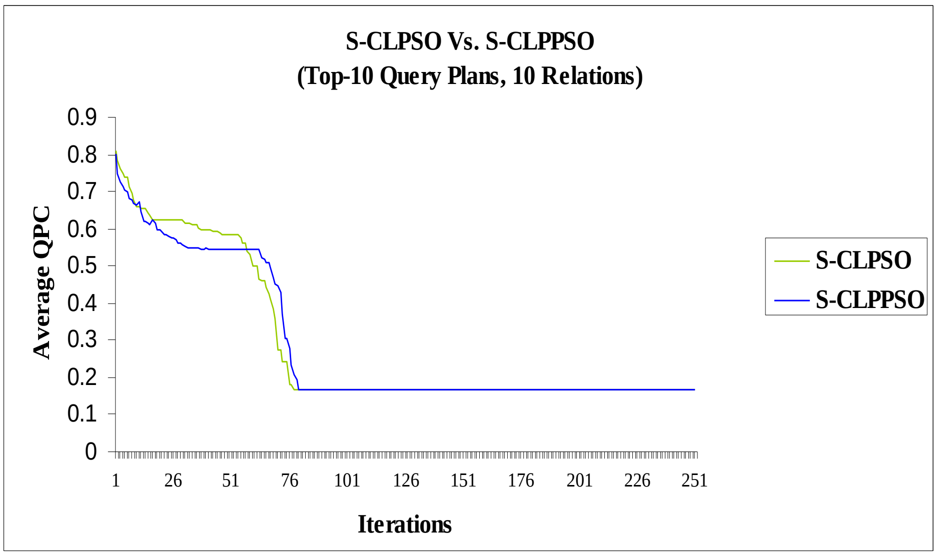

Figure 3.20: S-CLPSO vs. S-CLPPSO – Average QPC vs. Iterations
(Top-10 Query Plans, 10 Relations)

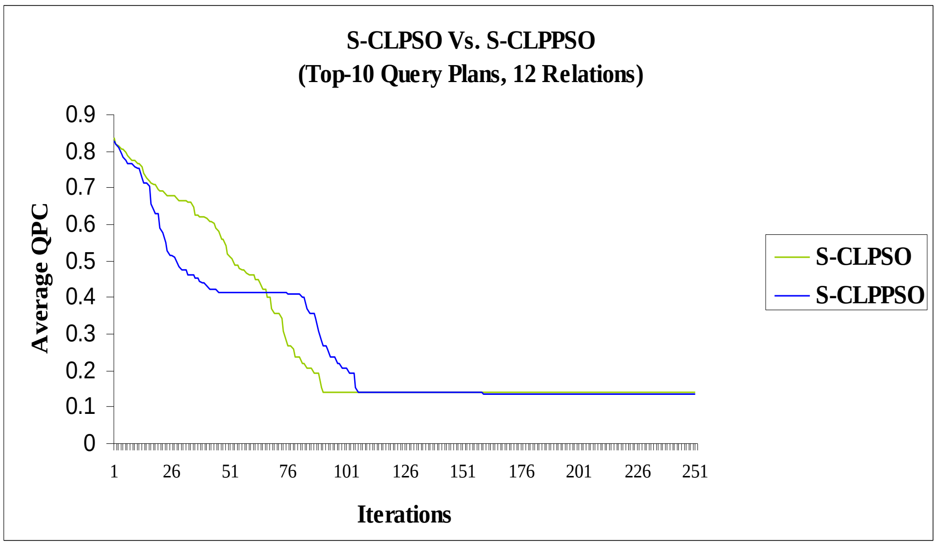

Figure 3.21: S-CLPSO vs. S-CLPPSO – Average QPC vs. Iterations (Top-10 Query Plans, 12 Relations)

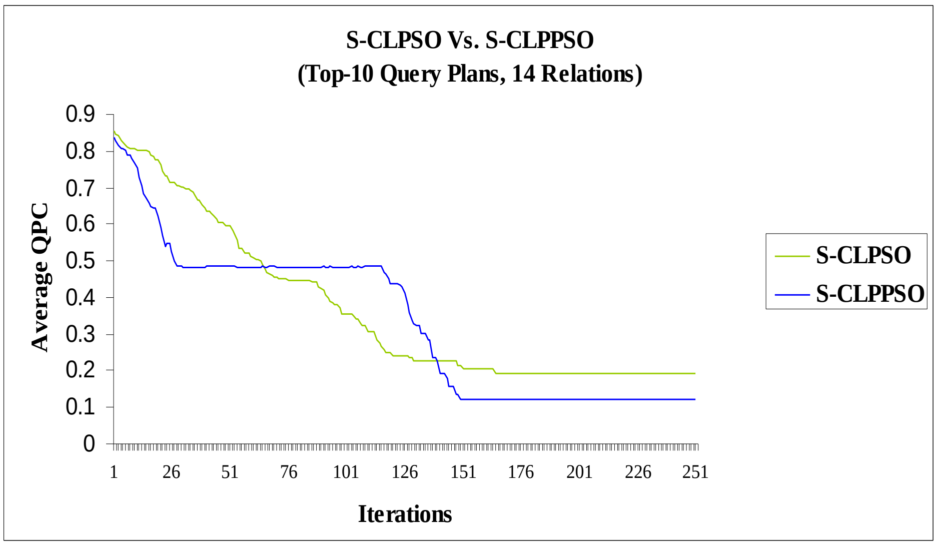

Figure 3.22: S-CLPSO vs. S-CLPPSO – Average QPC vs. Iterations (Top-10 Query Plans, 14 Relations)

Next, graphs were plotted to compare S-CLPSO and S-CLPPSO based algorithms on Average QPC value for selecting Top-K query plans(K=6, 8, 10, 12, 14) generated after 400 iterations. These graphs, plotted for relations 6, 8, 10, 12 and 14, are shown in figures 3.23, 3.24, 3.25, 3.26 and 3.27 respectively.

These graphs show that S-CLPSO and S-CLPPSO generate Top-K query plans with almost equal average QPC for 6, 8 and 10 relations. Whereas, S-CLPPSO, in comparison S-CLPSO, is able to generate Top-K query plans with relatively lower average QPC for 12 and 14 relations. Thus, it can be said that for higher number of relations, S-CLPPSO is able to generate good quality Top-K plans with relatively lower average QPC.

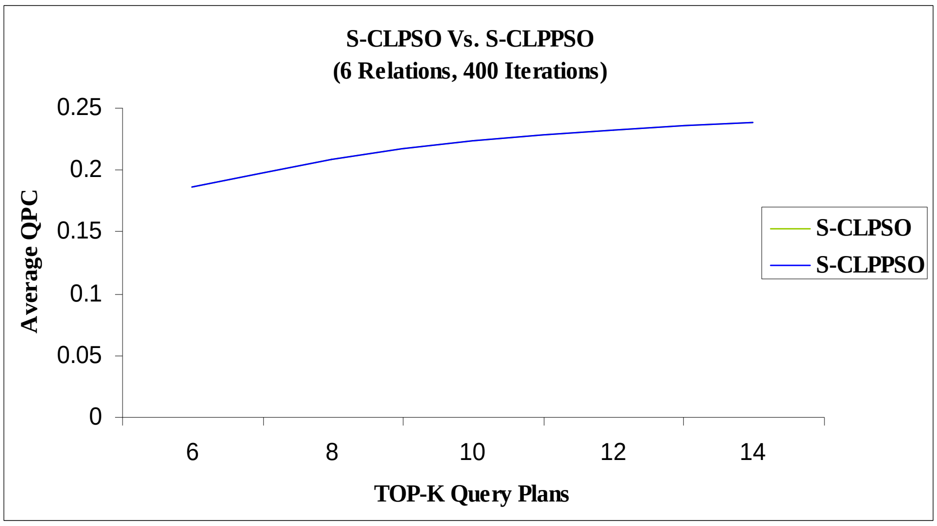

Figure 3.23: S-CLPSO vs. S-CLPPSO – Average QPC vs. Top-K Query Plans (6 Relations, 400 iterations)

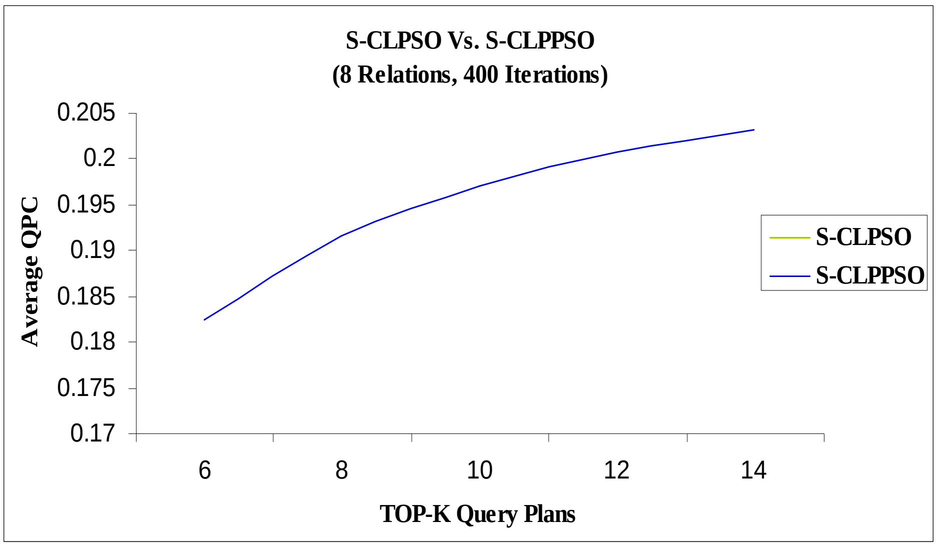

Figure 3.24: S-CLPSO vs. S-CLPPSO – Average QPC vs. Top-K Query Plans (8 Relations, 400 iterations)

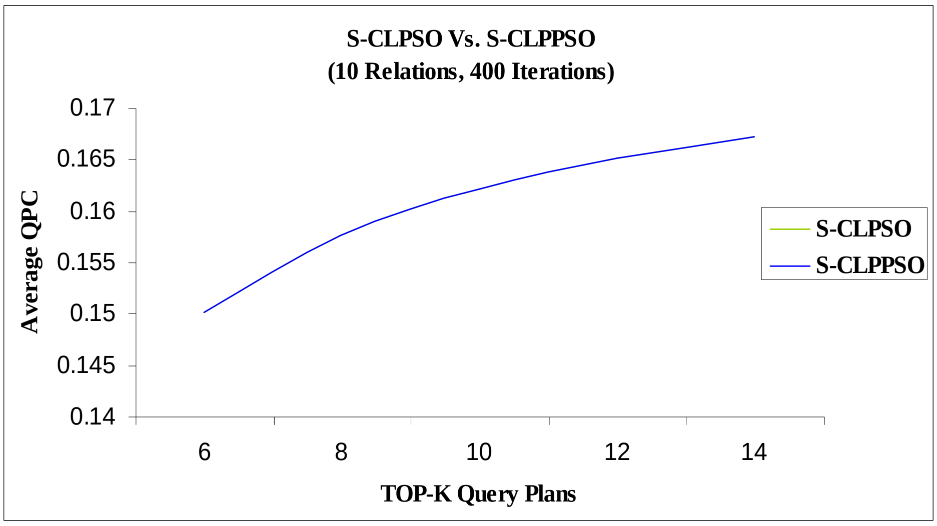

Figure 3.25: S-CLPSO vs. S-CLPPSO – Average QPC vs. Top-K Query Plans (10 Relations, 400 iterations)

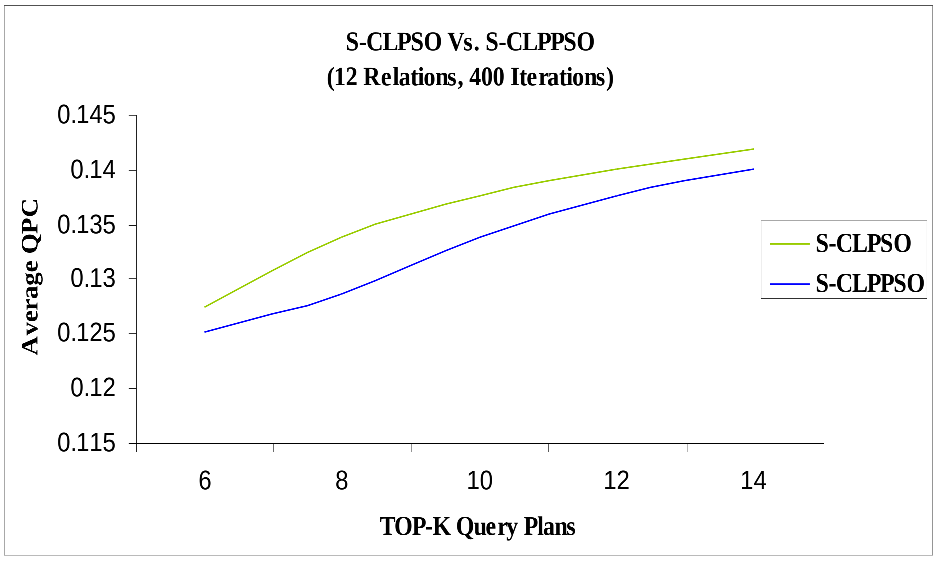

Figure 3.26: S-CLPSO vs. S-CLPPSO – Average QPC vs. Top-K Query Plans (12 Relations, 400 iterations)

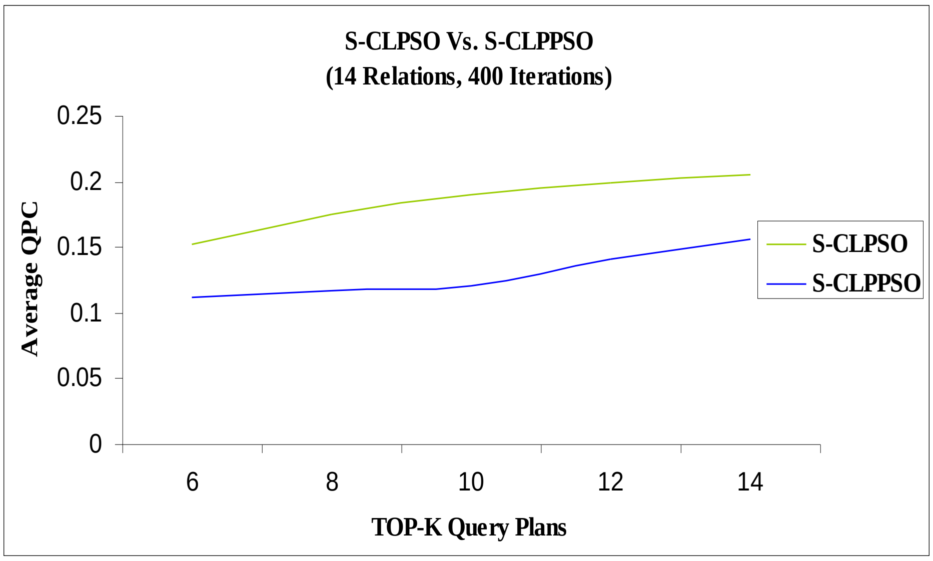

Figure 3.27: S-CLPSO vs. S-CLPPSO – Average QPC vs. Top-K Query Plans (14 Relations, 400 iterations)

CHAPTER 4

Conclusion

The aim of distributed query processing is to process distributed queries in an efficient manner. This would require devising an effective and efficient distributed query processing strategy that would minimize the total query processing cost. This is possible if the query plans constructed using the strategy involve fewer numbers of sites for processing thereby incurring less communication overhead. This dissertation addresses the distributed query plan generation problem, which has been solved using GA in [VSV11]. In this dissertation, particle swarm optimization is used to solve this problem.

First, the distributed query plan generation problem is solved using single-objective particle swarm optimization with the objective to minimize the query processing cost (QPC), as defined in [VSV11]. Set-based Comprehensive Learning Particle Swarm optimization technique is used to generate Top-K query plans for a distributed query. Experiment based comparison of this algorithm with GA based distributed query plan generation algorithm [VSV11] is carried out. The experimental results show that for higher number of relations the S-CLPSO based algorithm is able to generate relatively better quality top-K query plans.

Next, the single-objective distributed query plan generation problem is formulated as a bi-objective distributed query plan generation problem. The two objectives being minimization of site communication cost (SCC) and maximization of relation concentration gain (RCG). These are motivated by the cost functions in [PV02]. This problem is solved using multi-objective particle swarm optimization technique S-CLPPSO. The experiment based comparison of S-CLPPSO with the S-CLPSO based distributed query plan generation algorithm shows that the S-CLPPSO algorithm is able to generate comparatively better quality query plans for higher number of relations in the user query.

It can therefore be concluded that the S-CLPPSO based algorithm performs relatively better amongst the three algorithms with S-CLPSO algorithm having a slight edge over the GA based algorithm.

References

[AMR05] Afshinmanesh F., Marandi A., Rahimi-Kian A.: A novel binary particle swarm optimization method using artificial immune system. In Proc. IEEE Int. Conf. Comput. Tool (EUROCON), 2005.

[AHH09] Alom B.M., Henskens F., Hannaford M.: Query Processing and Optimization in Distributed Database Systems. IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.9, September 2009.

[AHY83] Apers P.M., Hevner A.R., Yao S.B.: Optimization algorithms for distributed queries, IEEE Transactions on Software Engineering SE-9(1983)57–68, 1983.

[BMR04] Baumgartner U., Magele C., Renhart W.: Pareto optimality and particle swarm optimization. IEEE Transactions on Magnetics, 40(2), 1172–1175, 2004.

[BC81] Bernstein P.A., Chiu D.W.: Using semi-joins to solve relational queries, Journal of the ACM 28(1981)25–40, 1981.

[BG+81] Bernstein P.A., Goodman N., Wong E., Reeve C.L., Rothnie J.B.: Query processing in a system for distributed databases (SDD-1), ACM Transactions on Database Systems 6, 1981.

[B89] Bishop, J. M.: Stochastic searching network. In Proceedings of the 1st IEE Conference on Artificial Neural Networks, London, UK (pp. 329-331), 1989.

[BH07] Borji A., Hamid M.: Evolving a Fuzzy Rule-Base for Image Segmentation. International Journal of Electrical and Computer Engineering 2:7 2007.

[CP84] Ceri S., Pelagati G. Distributed Database: Principles and Systems. McGraw Hill, 1984.

[CY94] Chen M.S., Yu P.S.: A graph theoretical approach to determine a join reducer sequence in distributed query processing, IEEE Transactions on Knowledge and Data Engineering 6, 1994.

[CY93] Chen M.S., Yu P.S.: Combining join and semijoin operations for distributed query processing. IEEE Transactions on Knowledge and Data Engineering 5, 1993.

[CZ+10] Chen W., Zhang J., Chung H., Zhong W., Wu W., Shi Y.: A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems. IEEE Trans. Evolutionary Computation, vol. 14, no. 2, pp. 278-300 , Apr 2010.

[CT04] Chow C.K., Tsui H.T.: Autonomous agent response learning by a multi-species particle swarm optimization. In Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC'04), Portland (OR), USA (pp. 778-785), 2004.

[CK02] Clerc M., Kennedy J.: The particle swarm - explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation, 6(1), 2002.

[C04] Clerc M.: Discrete particle swarm optimization. In New Optimization Techniques in Engineering. New York: Springer-Verlag, 2004.

[CPL04] Coello Coello C. A., Pulido T. G., Lechuga S. M.: Handling multiple objectives with particle swarm optimization. IEEE Transactions on Evolutionary Computation, 8(3), 256–279, 2004.

[CL02] Coello Coello C.A., Lechuga M.S.: MOPSO: A proposal for multiple objective particle swarm optimization. In Congress on Evolutionary Computation (CEC'2002), volume 2, pages 1051-1056, Piscataway, New Jersey, May 2002.

[C99] Coello Coello C.A.: A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge and Information Systems. An International Journal, 1(3):269–308, August 1999.

[CY88] Cornell D.W., Yu P.S.: Site assignment for relations and join operations in the distributed transaction processing environment, Proceedings of the 4th International Conference on Data Engineering, 1988.

[DP+02] Deb K., Pratap A., Agarwal S., Meyarivan T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2):182–197, 2002.

[D92] Dorigo, M.: Optimization, learning and natural algorithms. Ph.D. thesis, Politecnico di Milano, Italy, 1992.

[ES98] Eberhart R. C., Shi Y.: Comparison between genetic algorithms and particle swarm optimization. In: W. Porto, N. Saravanan, D. Waagen, & A.E. Eiben (Eds.), Evolutionary Programming VII (pp. 611-616). Berlin: Springer, 1998.

[EK95] Eberhart R.C., Kennedy J.:A New Optimizer Using Particle Swarm Theory, 6th International Symposium on Micro Machine and Human Science, Nagoya, Japan, p. 39-43, 1995.

[FS02] Fieldsend J. E., Singh S.: A multi-objective algorithm based upon particle swarm optimization, an efficient data structure and turbulence. In Proceedings of the 2002 UK Workshop on Computational Intelligence (UKCI-02), Birmingham, UK (pp. 34-44), 2002.

[HY79] Hevner A.R., Yao S.B.: Query processing in distributed database systems, IEEE Transactions on Software Engineering SE-5, 1979.

[HES03] Hu X., Eberhart R. C., Shi Y.: Particle swarm with extended memory for multiobjective optimization. In Proceedings of the 2003 IEEE Swarm Intelligence Symposium (SIS'03), Indianapolis (IN), USA (pp. 193-197), 2003.

[HE02] Hu X., Eberhart, R. C.: Multiobjective optimization using dynamic neighborhood particle swarm optimization. In Proceedings of the 2002 IEEE Congress on Evolutionary Computation, Honolulu (HI), USA (pp. 1677-1681), 2002.

[IK90] Ioannidis Y.E., Kang Y.C.: Randomized algorithms for optimizing large join queries, ACM 1990.

[JK84] Jarke M., Koch J.: Query optimization in database systems. ACM Computing Surveys, volume 16, no. 2, pp. 111-152, June 1984.

[JOS01] Jin Y., Olhofer M., Sendhoff B.: Evolutionary dynamic weighted aggregation for multiobjective optimization: Why does it work and how? In Proceedings of the 2001 Genetic and Evolutionary Computation Conference (GECCO'01), San Francisco (CA), USA, pp. 1042-1049, 2001.

[KYY82] Kambayashi Y., Yoshikawa M., Yajima S.: Query Processing for Distributed databases using Generalized Semi-Joins, ACM, 1982.

[KE97] Kennedy J., Eberhart R.: A discrete binary version of the particle swarm optimization. A. Proceeding of the conference on System, Man, and Cybernetics [C], NJ,USA: IEEE Service Center, 1997.

[KE01] Kennedy J., Eberhart R.C.: Swarm Intelligence. Morgan Kaufmann Publishers, San Francisco, California, 2001.

[K99] Kennedy J.: Small worlds and mega minds: effects of neighborhood topology on particle swarm performance. In Proceedings of 1999 IEEE Congress on Evolutionary Computation, Washington (DC), USA (pp. 1931-1938), 1999.

[K00] Kosmann D.: The State of the Art in Distributed Query Processing. ACM Computing Surveys, Vol. 32, No. 4, pp. 422-469, December 2000.

[LW86] Lafortune S., Wong E.: A state transition model for distributed query processing. ACM Transactions on Database Systems 11, 1986.

[L03] Li X.: A non-dominated sorting particle swarm optimizer for multi-objective optimization. Lecture Notes in Computer Science, 2723, 37–48, 2003.

[LQ+04] Liang J.J., Qin A.K., Suganthan P.N., Baskar S.: Particle swarm optimization algorithms with novel learning strategies, In Proc. Int. Conf. Systems, Man, Cybernetics, The Netherlands, Oct. 2004.

[LS+06] Liang J.J., Suganthan P.N., Qin A.K., Baskar S.: Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions.Accepted by IEEE Transactions on Evolutionary Computation, to appear in June 2006.

[LS06] Liang J.J., Suganthan P.N.: Adaptive Comprehensive Learning Particle Swarm Optimizer with History Learning. In Proceedings of the 6th International Conference on Simulated Evolution and Learning, vol. 4247, pp. 213-220, 2006.

[LTL07] Liao C.J., Tseng C.T., Luarn P.: A discrete version of particle swarm optimization flowshop scheduling problems. Comput. Operations Res., vol. 34, no. 10, pp. 3099–3111, Oct. 2007.

[LK73] Lin S., Kernighan B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations Res., vol. 21, no. 2, pp. 498–516, Mar.–Apr. 1973.

[LM+85] Lohman G.M., Mohan C., Haas L.M., Daniels D., Lindsay B.G., Selinger P.G., Wilms P.F.: Query Processing in R*. ;Query Processing in Database Systems, 1985.

[LRK01] Lovbjerg M., Rasmussen T.K., Krink T.: Hybrid particle swarm optimizer with breeding and subpopulations. In: Proceedings of the Third Genetic and Evolution Composition Conference (GECCO-2001), vol. 1. Morgan Kaufman, pp. 469–476, 2001.

[LC85] Lu H., Carey M.J.: Some experimental results on distributed join algorithms in a local network, Proceedings of the 10th International Conference on VLDB, Stockholm, 1985.

[ML86] Mackert L.F., Lohman G.M.: R* optimizer validation and performance evaluation for distributed queries, Proceedings of the 12th International Conference on VLDB, Kyoto, 1986.

[MCL04] Mahfouf M., Chen M.Y., Linkens D. A.: Adaptive weighted particle swarm optimization for multi-objective optimal design of alloy steels. [Berlin: Springer.]. Lecture Notes in Computer Science, 3242, 762–771, 2004.

[MR95] March S.T., Rho S.: Allocating data and operations to nodes in distributed database design, IEEE Transactions on Knowledge and Data Engineering 7, 1995.

[MLR90] Martin T.P., Lam K.H., Russell J.L.: Evaluation of site selection algorithms for distributed query processing, Computer Journal 33(1990)61–70, 1990.

[MKN03] Mendes R., Kennedy J., Neves J.: Watch thy neighbor or how the swarm can learn from its environment. In Proceedings of the 2003 IEEE Swarm Intelligence Symposium, Indianapolis (IN), USA (pp. 88-94), 2003.

[M99] Miettinen K.: Nonlinear Multiobjective Optimization. Kluwer Academic Publishers, Boston, Massachusetts, 1999.

[M94] Millonas M. M.: Swarms, phase transitions, and collective intelligence. In C.G. Langton (Ed.), Artificial Life III (pp. 417-445), 1994.

[ME92] Mishra P., Eich M.H.: Join processing in relational databases, ACM Computing Surveys 24 ,1992.

[MC99] Moore J., Chapman R.: Application of particle swarm to multi-objective optimization, Department of Computer Science and Software Engineering, Auburn University, 1999.

[MT03] Mostaghim S., Teich J.: The role of ε-dominance in multi-objective particle swarm optimization methods. In Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC’03), Canberra, Australia (pp. 1764-1771), 2003.

[OV91] Ozsu M.T., Valduriez P.: Principles of Distributed Database Systems .2nd edition, Prentice Hall, Englewood Cliffs, N.J., 1991.

[PW+04 (a)] Pang W., Wang K.P., Zhou C.G., Dong L.J., Liu M., Zhang H.Y., Wang J.Y. : Modified particle swarm optimization based on space and transformation for solving traveling salesman problem.In Proc. 3rd Int. Conf. Mach. Learning Cybern., 2004.

[PW04 (b)] Pang W., Wang K.P., Zhou C.G., Dong L.J.: Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In Proc. 4th Int. Conf. Comput. Information Technol. (CIT), 2004.

[PV02 (a)] Parsopoulos K. E., Vrahatis M. N.: Particle swarm optimization method in multiobjective problems. In Proceedins of the 2002 ACM Symposium on Applied Computing (SAC’02), Madrid, Spain (pp. 603-607), 2002.

[PV02(b)] Parsopoulos K. E., Vrahatis, M. N.: Recent approaches to global optimization problems through particle swarm optimization. Natural Computing, 2002.

[PV02(c)] Parimala N and T.V. Vijay Kumar, Querying Multidatabase Systems Using SIQL, In the proceedings of the 5th International conference on Flexible Query Answering Systems, FQAS-2002, Copenhagen, Denmark, October 27-29, 2002, Lecture Notes in Artificial Intelligence (LNAI-2522), Springer Verlag, pp. 301-313

[PTV04] Parsopoulos K.E., Tasoulis D.K., Vrahatis M.N.: Multiobjective Optimization Using Parallel Vector Evaluated Particle Swarm Optimization, Proceedings of the IASTED International Conference on Artificial Intelligence and Applications (AIA 2004), Innsbruck, Austria, pp. 823-828, 2004.

[PV08] Parsopoulos K.E., Vrahatis M.N.: Multiobjective Particle Swarm Optimization Approaches, Multi-Objective Optimization in Computational Intelligence: Theory and Practice, by Lam Thu Bui, Sameer Alam (Eds.), Chapter 2, pp. 20-42, IGI Global, 2008, (ISBN-10: 1599044986, ISBN-13: 978-1599044989), 2008.

[PV10] Parsopoulos K.E., Vrahatis M.N.: Particle Swarm Optimization and Intelligence: Advances and Applications, Information science reference, 2010.

[PPV07] Petalas Y. G., Parsopoulos K. E., Vrahatis M. N.: Memetic particle swarm optimization. Annals of Operations Research, 156(1), 99–127, 2007.

[PV88] Pramanik S., Vineyard D.: Optimizing join queries in distributed databases, IEEE Transactions on Software Engineering 14, 1988.

[PC04] Pulido T.G., Coello Coello C. A.: Using clustering techniques to improve the performance of a particle swarm optimizer. [Berlin: Springer.]. Lecture Notes in Computer Science, 3102, 225–237, 2004.

[PBP09] Puranik P., Bajaj P.R., Palsodkar P.M.: Fuzzy based Color Image Segmentation using Comprehensive Learning Particle Swarm Optimization (CLPSO) – A Design Approach. In the Proceeding of the international MultiConference of Engineers and Computer Scientists 2009 Vol I IMECS 2009, March 18-20, 2009.

[RN05] Raquel C. R., Naval P. C.: An effective use of crowding distance in multiobjective particle swarm optimization. In Proceedings of the 2005 Genetic and

Evolutionary Computation Conference (GECCO’05), Washington (DC), USA (pp. 257-264), 2005.

[RL02] Ray T., Liew K.M.: A swarm metaphor for multiobjective design optimization. Engineering Optimization, 34(2):141–153, March 2002.

[RC06 (a)] Reyes-Sierra M., Coello Coello C. A. : Multi-objective particle swarm optimizers: a survey of the state-of-the-art. International Journal of Computational Intelligence Research, 2(3), 287–308, 2006.

[RC05] Reyes-Sierra M., Coello Coello C. A.: Improving PSO-based multi-objective optimization using crowding, mutation and ε-dominance. [Berlin: Springer.]. Lecture Notes in Computer Science, 3410, 505–519, 2005.

[RC06 (b)] Reyes-Sierra M., Coello Coello C. A.: On-line adaptation in multi-objective particle swarm optimization. In Proceedings of the 2006 IEEE Swarm Intelligence Symposium (SIS’06), Indianapolis (IN), USA (pp. 61-68), 2006.

[RMN11] Rezazadeh I., Meybodi M.R., Naebi A.: Adaptive particle swarm optimization algorithm for dynamic environments. Proceeding ICSI'11 Proceedings of the Second international conference on Advances in swarm intelligence - Volume Part I Pages 120-129 Springer-Verlag Berlin, Heidelberg,2011.

[RM97] Rho S., March S.T.: Optimizing distributed join queries: A genetic algorithmic approach. Annals of Operations Research, 71, pp. 199-228, 1997.

[SAA02] Salman A., Ahmad I., Al-Madani S.: Particle swarm optimization for task assignment problem. Microprocessors Microsyst., vol. 26, no. 8, pp. 363–371, Aug. 2002.

[SMK00] Sbalzarini I.F., Muller S., Koumoutsakos P.: Multiobjective optimization using evolutionary algorithms. In Proceedings of the Summer Program 2000.

[S86] Segev A.: Optimization of join operations in horizontally partitioned database systems, ACM Transactions on Database Systems 11, 1986.

[SH06] Sha D.Y., Hsu C.: A hybrid particle swarm optimization for job shop scheduling problem. Comput. Ind. Eng., vol. 51, no. 4, pp. 791–808, Dec. 2006.

[SW91] Shasha D., Wang T.: Optimizing equijoin queries in distributed databases where relations are hash partitioned, ACM Transactions on Database Systems 16, 1991.

[SYY06] Shen B., Yao M., Yi W.: Heuristic information based improved fuzzy discrete PSO method for solving TSP. In Proc. 9th Pacific Rim Int. Conf. Artif. Intell. (PRICAI), 2006.

[SE98] Shi Y., Eberhart R.C.: Parameter selection in particle swarm optimization. In Proceedings of the Seventh Annual Conference on Evolutionary Programming, pages 591-600, 1998.

[SK86] Stephens D. W., Krebs, J. R.: Foraging Theory. Princeton: Princeton University Press, 1986.

[SM+08] Sulaiman S., Mariyam S., Shamsuddin H., Forkan F.B., Abraham A.: Intelligent Web Caching Using Neurocomputing and Particle Swarm Optimization

Algorithm, In proceeding of: Second Asia International Conference on Modelling and Simulation (AMS 2008), Kuala Lumpur, Malaysia, 13-15 May, 2008.

[VSV11] Vijay Kumar, T.V., Singh, V., Verma, A. K.: Distributed Query Processing Plans Generation using Genetic Algorithm, International Journal of Computer Theory and Engineering, Vol.3, No.1, pp. 38-45, February 2011.

[WH+03] Wang K.P., Huang L., Zhou C.G., Pang W.: Particle swarm optimization for traveling salesman problem. In Proc. 2nd Int. Conf. Mach. Learning Cybern., 2003.

[WW+07] Wang Y., Wanga Y., Fenga X.Y., Huanga Y.X., Pub D.B., Zhoua W.G., Lianga Y.C., Zhoua C.G.: A novel quantum swarm evolutionary algorithm and its applications. Neurocomputing, vol. 70,nos. 4–6, pp. 633–640, Jan. 2007.

[WG+09] Wu H., Geng J., Jin R., Qiu J., Liu W., Chen J., Liu S.: An Improved Comprehensive Learning Particle Swarm Optimization and Its Application to the Semiautomatic Design of Antennas. IEEE Transactions on Antennas and Propagation., vol. 57, no. 10, pp. 3018-3028, Oct. 2009.

[YL89] Yoo H., Lafortune S.: An intelligent search method for query optimization by semijoins, IEEE Transactions on Knowledge and Data Engineering 1, 1989.

[YC83] Yu C.T., Chang C.C.: On the design of a distributed query processing algorithm, Proceedings of the ACM-SIGMOD International Conference on the Management of Data, San Jose, 1983,pp. 30–39, 1983.

[YC84] Yu C.T., Chang C.C.: Distributed Query Processing. ACM Computing Surveys, volume 16, no. 4, pp. 399-433, 1984.

[ZZ10] Zhan Z., Zhang J.: An parallel particle swarm optimization approach for multiobjective optimization problems.In Proc. GECCO, 2010.

[ZZ+09] Zhan Z.H., Zhang J., Li Y., Chung H.S.: Adaptive particle swarm optimization," IEEE Trans. Syst., Man, and Cybern. B., vol. 39, no. 6. pp. 1362-1381, Dec. 2009.

[ZZ+10] Zhan Z.H., Zhang J., Li Y., Shi Y.H.: Orthogonal learning particle swarm optimization. IEEE Trans. Evol. Comput., 2010.

[ZSD10] Zhao S.Z., Suganthan P.N., Das S.: Dynamic Multi-Swarm Particle Swarm Optimizer with Sub-regional Harmony Search. WCCI 2010 IEEE World Congress on Computational Intelligence July, 18-23, 2010.

[ZDT00] Zitzler E., Deb K., Thiele L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results Evolutionary Computation, 8(2):173–195, Summer 2000.

URLs

[U1] http://cims.nyu.edu/~gn387/glp/lec1.pdf
[U2] http://en.wikipedia.org/wiki/Artificial_Ants
[U3] http://en.wikipedia.org/wiki/Multi-objective_optimization
[U4] http://en.wikipedia.org/wiki/Shoaling_and_schooling
[U5] http://en.wikipedia.org/wiki/Swarm_behavior

[U6] http://en.wikipedia.org/wiki/Swarm_techniques
[U7] http://worldwidescience.org/topicpages/c/collective+intelligence+temporal.html
[U8] http://www.calresco.org/lucas/pmo.htm
[U9] http://www.digplanet.com/wiki/Multi-objective_optimization
[U10] http://www.hindawi.com/journals/acisc/2012/897127/fig
[U11] http://www.powershow.com/view/14e8b7-ZDJjZ/Swarm_Intelligence
[U12] http://www.scholarpedia.org/article/Particle_swarm_optimization
[U13] http://www.sciencedirect.com/science/article/pii/S0010465512000136-gr001.gif
[U14] http://www.shef.ac.uk/acse/staff/peter_fleming/intromo
[U15] http://www.stigmergicsystems.com/stig_v1/stigrefs/article3.html
[U16] http://www.tutorgig.info/ed/Swarm
[U17] http://www.uni-leipzig.de/~vignesh/images/PSO-Scheme.jpg
[U18] http://www.wattpad.com/92392-swarm-intelligence
[U19] http://www23.homepage.villanova.edu/varadarajan.komanduri/PSO_meander-line.ppt