T.C.
MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN
PURE AND APPLIED SCIENCES

COMPARISON OF GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION ALGORITHM FOR BICRITERIA PERMUTATION FLOWSHOP SCHEDULING PROBLEM

Özgür UYSAL

(Industrial Engineer, MSc.)

DISSERTATION

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

IN

INDUSTRIAL ENGINEERING PROGRAMME

SUPERVISOR
Asst.Prof.Dr. Serol BULKAN

CO-ADVISOR
Asst.Prof.Dr. M.Fatih TAŞGETİREN

İSTANBUL 2006

T.C.
MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN
PURE AND APPLIED SCIENCES

COMPARISON OF GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION ALGORITHM FOR BICRITERIA PERMUTATION FLOWSHOP SCHEDULING PROBLEM

Özgür UYSAL

(Industrial Engineer, MSc.)

(141200920000212)

DISSERTATION

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

IN

INDUSTRIAL ENGINEERING PROGRAMME

SUPERVISOR
Asst.Prof.Dr. Serol BULKAN

CO-ADVISOR
Asst.Prof.Dr. M.Fatih TAŞGETİREN

İSTANBUL 2006

MARMARA UNIVERSITY

THE INSTITUTE FOR

GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

ACCEPTANCE AND APPROVAL DOCUMENT

THESIS TITLE

Established committee listed below, on ..... and ..... by the *INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES*' Executive Committee, have accepted Mr./Mrs./Miss. ..... 's Master of Science / Doctor of Philosophy thesis, titled as " .....in .....

COMMITTEE

Advisor :

Member :

Member :

Member :

Member :

Date of thesis' / dissertation's defense before the committee : .....

APPROVAL

Mr. / Mrs. / Miss. ..... has satisfactorily completed the requirements for the degree of Doctor of Philosophy / Master of Science in ..... at Marmara University.
Mr. / Mrs. / Miss. ..... is eligible to have the degree awarded at our convocation on ..... Diploma and transcripts so noted will be available after that date.

Istanbul **DIRECTOR**

ACKNOWLEDGEMENT

This thesis is an outcome of a long journey.

First of all, I am very grateful to my supervisor Dr. Serol Bulkan and co-advisor Dr. M.Fatih Taşgetiren for their support and guidance.

My family of course deserves more thanks than I can possibly give them.

Also, I was lucky to have numerous friends who have helped me. Yet, I owe special thanks to Dr. Mehmet Şevkli, Ali Türkyılmaz, Hatice Uysal and Hatice Uçar who have enriched this dissertation with their support.

TABLE OF CONTENTS

ÖZET

İKİ KRİTERLİ PERMÜTASYONLU AKIŞ TİPİ ÜRETİM ÇİZELGELEMESİ PROBLEMİ İÇİN GENETİK ALGORİTMA VE PARÇACIK SÜRÜ OPTİMİZASYONU YÖNTEMLERİNİN KARŞILAŞTIRILMASI

Akış tipi üretim çizelgeleme problemi, üzerinde çok çalışılmış olan alanlardan biridir. Problemin çapı büyüdükçe, analitik çözüm bulmak imkansızlaşır ve burada sezgisel yaklaşımlar devreye girer.

Literatüre bakıldığında, bu problem için genelde tek kriterli yaklaşımlar geliştirildiği görülür; toplam yapım zamanı en çok kullanılmış olan kriterdir. Az sayıda makine için çok kriterli sezgisel yöntemler bulunsa da, ikiden fazla makine için genelde sadece tek kriter dikkate alınmıştır.

Bu tez çalışmasında, 50 iş – 20 makine gibi büyük çaplı problemler için, toplam yapım zamanı ve en büyük pozitif gecikme zamanı kriterleri birlikte dikkate alınmıştır. Bu amaçla, bir Parçacık Sürü Optimizasyonu (PSO), bir de Genetik Algoritma (GA) sezgisel yöntemi geliştirilmiş ve standart test problemlerine uygulanmıştır. PSO ve GA'nın sadece yalın şekilleri değil, aynı zamanda Değişken Komşuluk Arama isimli bir yerel arama yöntemiyle melezlenmiş olan şekilleri de denenmiş, ve iki algoritmanın performansları birbirleriyle karşılaştırılmıştır.

Elde edilen sonuçlara göre, en büyük pozitif gecikme zamanı kriterinin ağırlıklı olduğu durumlarda PSO daha iyi sonuç vermiş; toplam yapım zamanı kriterinde ise GA daha başarılı olmuştur. İşlem sürelerinde ise, her durumda PSO daha çabuk sonuca ulaşmıştır. Yerel arama katılmış melez algoritmalar, yalın hallerine göre daha iyi sonuçlara ulaşmış; ancak, işlem süresi ciddi oranda artmıştır.

Anahtar Kelimeler: Akış tipi üretim, sezgisel yaklaşımlar, PSO, GA, çift kriter

Temmuz 2006 **Özgür UYSAL**

ABSTRACT

COMPARISON OF GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION ALGORITHM FOR BICRITERIA PERMUTATION FLOWSHOP SCHEDULING PROBLEM

Flowshop scheduling problem is a well known research field for fifty years. As the problem size gets bigger, an analytical solution becomes impossible. Here, heuristic solutions come to the stage.

In the literature, generally solutions regarding a single criterion are developed; and makespan is the most common objective used. There are some multi objective solutions for one or two machines; but, only one criterion is generally used for more than two machines.

In this thesis, makespan and maximum tardiness criteria are used concurrently, for big problem sizes like 50 jobs-20 machines. For this purpose, a Particle Swarm Optimization (PSO), and a Genetic Algorithm (GA) is developed and applied to standard test problems.

Not only the pure versions of PSO and GA, but also their hybrid versions – i.e. with a local search called Variable Neighborhood Search (VNS) embedded - are tested; and the relative performances of the two algorithms are compared.

As a result, PSO performed better for the situations where the weight of maximum tardiness criterion was greater, while GA surpassed PSO when the makespan objective was dominant. Regarding the CPU times, PSO found a solution more quickly, for all occasions. The with-VNS versions of the algorithms found better solutions compared to the pure versions; but, it took them much longer.

Keywords: Flowshop scheduling, heuristic optimization, PSO, GA, bicriteria

July 2006 **Özgür UYSAL**

CLAIM FOR ORIGINALITY

COMPARISON OF GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION ALGORITHM FOR BICRITERIA PERMUTATION FLOWSHOP SCHEDULING PROBLEM

Flowshop is a well-known research field for a long time. Genetic Algorithms (GA) are used extensively for many years. Particle Swarm Optimization (PSO) is a relatively new concept, but the application areas are expanding rapidly.

This thesis is one of the few applications of PSO to scheduling. Also, research on multiobjective scheduling is very scarce, and this thesis is on bicriteria scheduling.

To the best of our knowledge, this thesis is the first application of PSO to bicriteria flowshop scheduling. Additionally, not only the pure algorithms, but also the hybrid versions (i.e. with a local search embedded) of PSO and GA are tested and compared.

So, we hereby declare that this thesis is original, and makes a humble contribution to science and technology.

July 2006 **Asst.Prof.Dr. Serol BULKAN** **Özgür UYSAL**

ABBREVIATIONS

GA : Genetic Algorithm
PSO : Particle Swarm Optimization
PFS : Permutation Flowshop Scheduling
PFSP : Permutation Flowshop Scheduling Problem
VNS : Variable Neighborhood Search
NEH : Nawaz-Enscore-Ham heuristic
SA : Simulated Annealing

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

Flowshop scheduling is a well-known research field for many years. A large quantity of research has been carried out in this field considering many different objectives, “makespan“ being the most popular one. And, almost always a single objective has been taken into account; research regarding more than one criterion is very scarce.

Also, as the size of the problem gets bigger, analytical solution becomes impossible, and heuristic solutions come into the field. Then, the objective is to find a “good” or “near optimal” solution; if the global optima has not been reached (Pinedo, 1995).

In this thesis, the Permutation Flowshop Scheduling Problem (PFSP) is considered, where the objective is to minimize a weighted sum of makespan and maximum tardiness. The performance of two famous evolutionary algorithms, Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), are compared for many different variations, on a standard benchmark suite for PFSP. Finally, the results are reported, and compared.

This first chapter is a general introduction, while the second one gives a complete literature review. Then two chapters on GA and PSO follows, where detailed information about the philosophy, structure and operation of the two algorithms is given.

The fifth chapter is on the local search algorithm that’s extensively used within this study embedded to both GA and PSO, namely Variable Neigborhood Search (VNS). The sixth chapter is the experimentation chapter, and focuses on the details of my study. The model, parameters, computing environment are listed, and the overall results are given, in terms of both the solution quality and the computing time.

The last chapter includes the concluding remarks for my study, and the interpretation of the comparison results of GA and PSO. References of the whole

material used in the study are listed; and, detailed tables about the computational results are placed in the Appendix.

I.1. SCHEDULING

Scheduling is a decision-making process that plays an important role in most manufacturing and service industries. It is used in procurement and production, in transportation and distribution, and in information processing and communication. The scheduling function in a company uses mathematical techniques or heuristic methods to allocate limited resources to the processing of tasks. A proper allocation of resources enables the company to optimize its objectives and achieve its goals.

Scheduling is an important aspect of operational level shop floor decisions. Its importance and relevance to industry has prompted researchers to study it from different perspectives over the past three decades. Scheduling literature ranges from deterministic case to the stochastic case, from single machine problem to the multiple machine problem and from static to dynamic problem. Research on multiple and bicriteria scheduling has been scarce, especially when compared to research in single criterion scheduling (Nagar et al., 1995).

I.2. FLOWSHOP

A flowshop consists of n jobs that must be processed on m machines in the same machine order. The scheduling problem in flowshops is then finding a sequence of jobs for each machine according to certain performance measure(s).

There are several assumptions that are commonly made regarding the flowshop scheduling problem (Baker, 1974):

- Each job *i* can be processed at most on one machine *j* at the same time.
- Each machine *m* can process only one job *i* at a time.
- No preemption is allowed, i.e. the processing of a job *i* on a machine *j* cannot be interrupted.
- All jobs are independent and are available for processing at time 0.
- The set-up times of the jobs on machines are negligible and therefore can be ignored.
- The machines are continuously available.

- In-process inventory is allowed. If the next machine on the sequence needed by a job is not available, the job can wait and joins the queue at that machine.

There are $(n!)^m$ different alternatives for ordering jobs on machines. In most research, however, only a subset of these alternatives is considered under the assumption that the operating sequences of the jobs are the same on every machine. In this case, the number of alternatives reduces to $n!$, and the schedules that satisfy this assumption are called *permutation schedules*.

Although permutation schedules do not always include an optimal schedule, the importance of permutation schedules cannot be underestimated. This is because only permutation schedules are feasible in many real situations and it is easier to devise a method to find a good permutation schedule, than a method to find a good schedule which is not a permutation schedule.

Since it is unlikely that an efficient method can be found for solving the flowshop problem optimally, various heuristic methods have been developed for the problem (Kim, 1993).

I.3. GENETIC ALGORITHMS

Genetic algorithm (GA) is a well-known and mostly used evolutionary computation technique, which was developed by John Holland and his PhD students (Holland, 1975). The idea was inspired from Darwin's natural selection theorem which is based on the idea of the survival of the fittest.

Genetic algorithms have an initial population composed of randomly generated solutions. There are three stochastic operators such as selection, crossover and mutation which are applied to the set of solutions iteratively to produce hopefully better solutions. In selection, most fit members survive and the least fit are eliminated. Differentiation is attained through crossover and mutation. There is a probability for crossover and mutation. Beasley give detailed information about the fundamental and advanced aspects of GAs (Beasley et al., 1993).

I.4. PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) is one of the latest evolutionary optimization methods. It is a population-based technique, which was originally

developed by Kennedy & Eberhart in 1995. PSO is based on the metaphor of social interaction and communication, such as bird flocking and fish schooling.

Since it is population-based and evolutionary in nature, the members in a PSO algorithm tend to follow the leader of the group, i.e., the one with the best performance.

PSO shares many common points with GA. Both algorithms start with a group of a randomly generated population, both have fitness values to evaluate the population. Both update the population and search for the optimium with random techniques. Both systems do not guarantee success.

But, PSO is distinctly different from other evolutionary-type methods in a way that it does not use the filtering operation (such as crossover and/or mutation) and the members of the entire population are maintained through the search procedure so that information is socially shared among individuals to direct the search towards the best position in the search space.

PSO can be easily implemented and it is computationally inexpensive, since its memory and CPU speed requirements are low (Eberhart et al., 1996). Moreover, it does not require gradient information of the objective function under consideration, but only its values, and it uses only primitive mathematical operators. PSO has been proved to be an efficient method for many global optimization problems and in some cases it does not suffer the difficulties encountered by other evolutionary computation techniques (Eberhart and Kennedy, 1995).

In evolutionary computation techniques, three main operators are involved: the recombination, the mutation and the selection operators.

PSO does not have a direct recombination operator. However, the stochastic acceleration of a particle towards its previous best position, as well as towards the best particle of the swarm (or towards the best in its neighborhood in the local version), resembles the recombination procedure in evolutionary computation (Eberhart and Shi, 1998; Rechenberg, 1994; Schwefel, 1995).

In a PSO algorithm, each member is called “particle”, and each particle flies around in the multi-dimensional search space with a velocity, which is constantly updated by the particle’s own experience and the experience of the particle’s neighbors or the experience of the whole swarm, instead of being carried from fitness dependent selected “parents” to descendants as in GAs.

Moreover, PSO's directional position updating operation resembles mutation of GA, with a kind of memory built in. This mutation-like procedure is multidirectional both in PSO and GA, and it includes control of the mutation's severity, utilizing factors such as V_{max} and κ.

PSO is actually the only evolutionary algorithm that does not use the "survival of the fittest" concept. It does not utilize a direct selection function. Thus, particles with lower fitness can survive during the optimization and potentially visit any point of the search space (Eberhart and Shi, 1998).

The original PSO algorithm is described as below:

$$v_{ij}^{k} = v_{ij}^{k-1} + c_1 r_1 (pb_{ij}^{k-1} - x_{ij}^{k-1}) + c_2 r_2 (gb_j^{k-1} - x_{ij}^{k-1}) \quad (1a)$$

$$x_{ij}^{k} = x_{ij}^{k-1} + v_{ij}^{k} \quad (1b)$$

where c_1 and c_2 are positive constants, and r_1 and r_2 are two random functions in the range [0,1];

Equation (1) is the equation describing the flying trajectory of a population of particles. Equation (1a) describes how the velocity is dynamically updated and Equation (1b) the position update of the "flying" particles.

Equation (1a) consists of three parts. The first part is the "momentum" part. The velocity can't be changed abruptly. It is changed from the current velocity. The second part is the "cognitive" part which represents private thinking of itself - learning from its own flying experience. The third part is the "social" part which represents the collaboration among particles - learning from group flying experience (Shi and Eberhart, 1998b).

Two variants of the PSO algorithm are developed, namely PSO with a local neighborhood, and PSO with a global neighborhood. According to the global neighborhood, each particle moves towards its best previous position and towards the best particle in the whole swarm, called gbest model. On the other hand, according to the local variant so called lbest, each particle moves towards its best previous position and towards the best particle in its restricted neighborhood (Kennedy et al., 2001). Kennedy and Eberhart also developed the discrete binary version of the PSO (Kennedy and Eberhart, 1997).

Although the applications of PSO on combinatorial optimization problems are still limited, PSO has its merit in its simple concept and low computational cost.

Finally, we can say that PSO is a powerful method for optimizing continuous functions. However, it is not sufficient for solving discrete cases. (Deroussi et al., 2004) showed the Discrete Particle Swarm Optimizer (DPSO) to solve the combinatorial optimization problems. He combines local search and path relinking to DPSO and applies it to the well-known Traveling Salesman Problem. The proposed algorithm competes with the best iterated local search methods.

II. LITERATURE SURVEY

Since Johnson's pioneering work (Johnson, 1954) on the two machine regular permutation flowshop, a wealth of research has been conducted in both exact and heuristic methods for the PFSP. Due to the *NP*-completeness of the PFSP (Garey et al., 1976), researchers have mainly focused on the development of effective heuristics and metaheuristics (Ruiz et al, 2004).

The majority of research on scheduling problems addresses only a single criterion while the majority of real-life problems require the decision maker to consider more than a single criterion before arriving at a decision. In practice, however, quality is a multidimensional notion.

A firm, for instance, judges a production scheme on the basis of a number of criteria, for example, work-in-process inventories and observance of due dates. If only one criterion is taken into account, then the outcome is likely to be unbalanced, no matter what criterion is considered.

If everything is set on keeping work-in-process inventories low, then some products are likely to be completed far beyond their due dates, whereas, if the main goal is to keep the customers satisfied by observing due dates, then the work-in process inventories are likely to be large.

In order to reach an acceptable compromise, one has to measure the quality of a solution on all important criteria. This notice has led to the development of the area of multicriteria scheduling (Hoogeveen, 2005).

Makespan (maximum completion time) and maximum tardiness are among the most commonly used criteria in the flowshop scheduling research. Makespan is a measure of system utilization while maximum tardiness is a measure of performance in meeting customer due dates.

Therefore, it is not surprising that both objectives, particularly makespan, have been subject of research during the last decades. Since both problems (minimizing makespan as well as minimising tardiness in permutation flowshop scheduling) are NP-hard (see e.g. (T'kindt and Billaut, 2002) for a formal proof),

most of the research effort concentrates on finding heuristics that allow obtaining good solutions in a relatively short period of time.

With respect to makespan minimisation, the NEH heuristic (Nawaz et al., 1983) is perhaps the most significant contribution, as it is considered to be the most efficient constructive heuristic for the problem (see e.g. the experimental results of (Taillard, 1990) or (Framinan et al., 2003)). NEH heuristic is based on the idea that jobs with high processing times on all the machines should be scheduled as early in the sequence as possible (Ruiz and Maroto, 2005).

The makespan criterion for m-machine flowshop has been addressed in the literature by many other researchers including (Chu et al., 1995), (Zegordi et al., 1995), and (Riane et al., 1998). Ruiz and Maroto present a review and evaluation of heuristics for the PFS problem with the makespan criterion. They propose a comparison of 25 methods, ranging from the classical Johnson's algorithm or dispatching rules to the most recent metaheuristics, including tabu search, simulated annealing, genetic algorithms, iterated local search and hybrid techniques (Ruiz and Maroto, 2005).

The tardiness criterion has also been addressed in the literature by many researchers, e.g. (Townsend, 1977), (Kim, 1993), (Kim, 1995), (Srinivasaraghavan and Rajendran, 1998) and Armentano and Ronconi (1999).

With respect to minimisation of maximum tardiness in flowshops, some research focused on finding exact solutions for the problem (e.g. (Townsend, 1977)).

The research on multiple criteria is mainly focused on the single machine scheduling problem, see (Nagar et al., 1995). The reason for this is that the scheduling problem with multiple-machine is difficult even with a single criterion. Therefore, considering more than a single criterion makes the multiple-machine problem even more difficult to solve.

To name a few papers on bicriteria scheduling, Serifoglu and Ulusoy handle a two machine PFS Problem and compare the relative performances of three Branch & Bound approaches and two flowshop heuristics (Serifoglu and Ulusoy, 1998); while Köksalan and Keha use GA for single machine scheduling and consider two bicriteria scheduling problems: minimizing flowtime and number of tardy jobs, and minimizing flowtime and maximum earliness (Köksalan and Keha, 2003). T'kindt et al. consider the 2-machine flowshop scheduling problem with the objective of

minimizing both the total completion time and makespan criteria. They propose an Ant Colony Optimization approach to solve the problem (T'kindt et al., 2002).

A survey of the literature has revealed that the m-machine problem with the objectives of C_{max} and T_{max} is addressed only by (Daniels and Chambers, 1990), (Chakravarthy and Rajendran, 1999), (Allahverdi, 2004) and (Arroyo and Armentano, 2005).

First, I'll review the Nawaz, Enscore and Ham (NEH) heuristic for the makespan minimization, without considering due dates. NEH heuristic is included in the review, since it is employed in most of the main heuristics for the problem under consideration.

The heuristic by (Nawaz et al., 1983) consists of two phases: First the jobs are ranked according to the descending sum of their processing times. In a second phase, a solution is constructed in the following manner: Starting from a partial sequence constructed by taking the first job of the rank, then, for k = 2, ..., n, k partial sequences are constructed by inserting the kth job of the rank in all k slots of the partial sequence. These k partial sequences are evaluated with respect to makespan and the one obtaining the lowest value is retained as partial sequence for step k+1.

(Daniels and Chambers, 1990) suggest a constructive heuristic in order to find a set of heuristically efficient solutions to the bi-criteria problem of minimising C_{max} and T_{max}. The heuristic, in loop k, is based on considering the set of so far non-scheduled jobs that may occupy a certain position in the final schedule without violating a given maximum allowed tardiness. This maximum tardiness allowed is initially set to the tardiness of the sequence resulting from applying the NEH heuristic to the problem, and then it is gradually decreased until a feasible solution is found.

(Chakravarthy and Rajendran, 1999) address the same problem to minimize the objective function value subject to the constraint that maximum tardiness is not greater than a given value. Obviously, the problem under consideration is a special case; and for that problem, the authors propose a fast local search algorithm based on the SA algorithm (Kirkpatrick et al., 1983).

Allahverdi is another researcher who takes into account the NEH heuristic, which was originally developed for the objective of makespan. It is modified for the two objectives considered, which he calles modified NEH or MNEH in his paper. Allahverdi considers the same problem addressed by Daniels and Chambers, and

Chakravarthy and Rajendran. But, he also considers the problem without the constraint on the maximum tardiness and addresses the m-machine flowshop problem with the objective of minimizing a weighted sum of the two criteria (Allahverdi, 2004).

Arroyo and Armentano propose a genetic local search algorithm with features such as preservation of dispersion in the population, elitism, and use of a parallel multi-objective local search so as to intensify the search in distinct regions. The algorithm is applied to the flowshop scheduling problem for the following two pairs of objectives: (i) makespan and maximum tardiness; (ii) makespan and total tardiness (Arroyo and Armentano, 2005).

Metaheuristics, such as simulated annealing, genetic algorithms, tabu search, ant colony optimization, scatter search, iterated local search, and particle swarm optimization have received considerable interest in the fields of applied artificial intelligence and combinatorial optimization. Plenty of hard problems in a huge variety of areas, including bioinformatics, logistics, engineering, business, etc., have been tackled successfully with metaheuristic approaches. For many problems, the resulting algorithms are considered to be the state-of-the-art methods.

The heuristic optimization algorithms presented in the literature can be classifies as constructive methods or neighborhood search methods based on local search. Some applications of constructive algorithms are presented in (Campbell et al, 1970), (Dannenbring, 1977), (Nawaz et al., 1983), (Koulamas, 1998), (Framinan et al., 2002) and (Framinan and Leisten, 2003). To achieve a better solution quality, modern heuristics based on the neighborhood search has been presented for the PFSP such as Simulated Annealing in (Osman and Potts, 1989) and (Ogbu and Smith, 1990), Tabu Search in (Nowicki and Smutnicki, 1996), (Reeves, 1993) and (Taillard, 1990), Genetic Algorithms in (Reeves, 1995) and (Reeves and Yamada, 1998), and Ant Colony Optimization in (Stutzle, 1998).

Starkweather et al. compared six crossover operators over Traveling Salesman Problem and a warehouse scheduling problem found that the effectiveness of the sequencing operators changes depending on the problem domain (Starkweather, 1991).

Whitley presents the strengths and weaknesses of evolutionary algorithms covering genetic algorithms, evolution strategies, genetic programming and evolutionary programming. He gives more experimental forms of GAs including the

parallel island models and parallel cellular genetic algorithms in his articles. He reviews the theoretical foundations of GA (Whitley, 2002).

Genetic algorithms have a wide range of applications ranging from optimization, design and machine learning problems to scheduling problems and etc.

The success in solving scheduling applications mostly depends on the choice of the search algorithm. Choosing an appropriate technique can be possible in two ways: the generality of the algorithm can be examined or a comparison can be done after applying many algorithms to the scheduling problem.

In this century, industrial robots perform many tasks. The aim of using them is to reduce the cycle time and to obtain high productivity. An industrial robot has constant finishing time for each operation, but it is possible to reduce the makespan with an optimal sequence of tasks. Zacharia and Aspragathos applied GA to cope with this problem (Zacharia and Aspragathos, 2005).

It is significant to arrange an optimal curriculum schedule for every school. This is not only required for educational goals, but also for effectively utilizing the faculty resources. This is a rather difficult task and GA has a great reputation in solving this problem. There are many articles written on this topic (Wang, 2005).

Chan and Chung emphasizes the trade-off between the earliness on time and the tardiness situations for a distribution network. They develop a multi-criterion genetic optimization methodology. The proposed algorithm combines analytical hierarchy process, a multi-criterion decision making tool, with GAs (Chan and Chung, 2004).

Whitley used GAs for setting weights in neural networks (NN). The training data were used to estimate the output behavior of NN. It is understood that combining GA with NN gives promising results (Whitley, 1995).

GA has a wide usage in scheduling job shops and flowshops and they give rather good results for both kinds of problems. There are many articles available in literature. Generally objective function is set to minimize the make span or minimize the earliness/tardiness in these articles. In Leu and Hwang, a resource constrained flowshop scheduling model is used to solve a mixed precast production problem. Since precast production has many tasks which are done in the same order through all tasks, it is as flowshop process (Leu and Hwang, 2002).

Good properties of search methods are generally integrated to the algorithms, so as to find better solutions in a reasonable amount of time. This kind of algorithms

are called as hybrid algorithms. In Kim, a hybrid GA was combined with fuzzy logic for solving resource-constrained project scheduling (Kim et al, 2003). Nearchou added some features of GA and local search to his Simulated Annealing (SA) algorithm for finding an optimal scheduling for a flowshop problem with the makespan criterion (Nearchou, 2004). The hybrid GA was successfully applied to permutation flowshops for solving sequence-dependent set-up times (Ruiz et al., 2004). The parameters were fine tuned by using design of experiments approach. Hino et al. combined the best characteristics of genetic algorithms and tabu search to solve the earliness/tardiness problem in a single machine environment (Hino et al., 2005).

Since PSO was first introduced by Kennedy and Eberhart in 1995, it has been successfully applied to optimize various continuous nonlinear functions. Some of the wide application areas of PSO are, power and voltage control (Yoshida et al., 2000), neural network training (Van den Bergh and Engelbecht, 2000), mass-spring system (Brandstatter and Baumgartner, 2002), and supplier selection and ordering problem (Yeh, 2003). More literature can be found in (Kennedy et al., 2001).

PSO has some tuning parameters which influence the performance of the algorithm; the exploration and exploitation tradeoff. In the work of Eberhart, Simpson & Dobbins, it was realized that some of the parameters were redundant, and they were removed from the original algorithm (Eberhart et al., 1996).

Trelea gives some insights about parameter selection in PSO. According to the article, some parameters can be discarded; since they add no value to the algorithm. Trelea analyzes the deterministic PSO algorithm for its dynamic behavior and convergence property (Trelea, 2003).

The velocities of particles' on each dimension are restricted to V_{max}. A larger V_{max} facilitates global exploration, while smaller V_{max} facilitates local exploitation. Shi and Eberhart added the inertia weight as a constant to the velocity in order to control the exploration and exploitation (Shi and Eberhart, 1998a). The use of inertia weight improved the performance of the algorithm in many applications.

Chatterjee & Siarry proposed a new variation of PSO which introduced a nonlinear inertia weight for the particle's old velocity in PSO equations and it improved the convergence as well. It also presented a way for parameter selection and compared the results with other parameter selection methods (Chatterjee and Siarry, 2006).

Clerc introduced the constriction factor (K) to PSO (Clerc, 1999). It controls, constrains velocities and thus insures convergence. The constriction factor negated the need for V_{max}.

Eberhart and Shi demonstrated that although previous evolutionary paradigms can generally solve static problems, PSO can successfully optimize dynamic systems (Eberhart and Shi, 2001). It can not be known when a larger or a smaller inertia weight is needed. Therefore, that value is set to a dynamic value which starts from 0.9 and descends linearly till 0.4.

We can see many applications of PSO algorithm in the literature. The first application was about the network architecture and is available in the article of (Kennedy et al., 2001). Eberhart & Hu evolved the neural network with PSO in order to diagnose the human tremor (Eberhart & Hu, 1999).

When numerically controlled machines emerged, the productivity was far from being optimal. As an example to optimize these systems, the metal removal operation was investigated by Tandon (Tandon et al, 2002).

(Pavlidis et al., 2005) compared PSO with other computational intelligence methods in finding the Nash equilibrium in game theory.

(Chang et al., 2001) research constitutes an alternative solution solving scheme for resource-constrained project scheduling problem.

(Ghoshal, 2004) compared some metaheuristic techniques such as PSO, a hybrid PSO and a hybrid GA-SA to optimize the proportional-integral derivative gains, which are used in multi-area thermal generating plants. He found PSO to be more optimal and it is achieved in least time.

Other applications include, power and voltage control, ingredient mix optimization, system design, multi-objective optimization, pattern recognition, biological system modeling, signal processing, robotic applications, decision-making, simulation,…etc.

The evolutionary computation paradigms and PSO algorithm were compared in many articles (Eberhart and Shi, 2001), (Angeline, 1998) and (Kennedy and Spears, 1998). In the study of Eberhart and Shi, the operators of each paradigm are reviewed. The objective of comparison is not to determine which algorithm is better; but to demonstrate how each of them works, and how can they be combined to improve the performance. Some of the features of GA were incorporated to PSO.

(Carlisle & Dozier, 2000) wanted to define a general purpose PSO swarm, to be used as a base swarm description. In another study, they showed a method for adapting the particle swarm optimizer to dynamic environments. The particle resets its previous best record whenever the environment changes and forgets about its experience to that point. The type of resetting changes based on the iteration count or the magnitude of change in the environment. It is concluded that a more gradual reset throughout the population might provide better convergence.

(Løvbjerg et al., 2001) combined PSO with genetic algorithm concepts and evaluated if it was competitive on function optimization. They employed the concepts of breeding and subpopulation for velocity and position updates. The method was heavy computationally due to the additional burden of breeding and subpopulation.

Like many other evolutionary and classical minimization methods, PSO suffers from being trapped at local optima. Another new technique to alleviate the local optima problem was introduced by (Parsopoulos and Vrahatis, 2002). The method is called Stretching Technique. It consists of two-stage objective function. The method is applied after a local minimum has been found. The local minima were eliminated while the global minimum is preserved in the method.

Very recently, (Ruiz and Stutzle, 2006) developed an iterative greedy algorithm with and without a local search. To compare the performance of the IG algorithms to well-known techniques from the literature, Ruiz and Stutzle re-implemented 12 classical or recent, well performing algorithms: NEH heuristic of (Nawaz et. al., 1983) with the improvements of (Taillard, 1990), the simulated annealing algorithm of (Osman and Potts, 1989), the tabu search algorithm of (Widmer and Hertz, 1989), the pure genetic algorithm of (Chen et. al. , 1995) and (Reeves et al., 1998), the hybrid genetic algorithm with local search of (Murata et al., 1996), two recent genetic algorithms of (Ruiz et. al., 2006), the genetic algorithm of (Aldowasian and Allahverdi, 2003), the iterated local search of (Stutzle, 1998); and two recent ant colony algorithms of (Rajendran and Ziegler, 2004).

(Liao et al. 2006) present a discrete version of PSO and apply it to the same benchmark suite that I used in my thesis by (Demirkol et al, 1998) for the total flow time criterion.

III. GENETIC ALGORITHMS

III.1. AN OVERVIEW

Genetic algorithms (GA) belong to the class of metaheuristics. It was firstly introduced by John Holland in 1960. The idea was inspired from a biological issue which is known to be Darwin's evolution theorem. The evolutionary ideas of the natural selection and genetics constitute the basics of GA. The concepts used in the algorithm are same as the ones which are used in biology, e.g., the genes on each chromosome correspond to variables of each solution. In the algorithm, the survival of the fittest among individuals over consecutive generations is simulated when a problem is being solved.

GAs are good at solving continuous and discrete combinatorial problems. The probability of getting 'stuck' at local optima is less than the gradient search methods. But GAs are computationally expensive. It is simple to deal with them; since very good results can be obtained for different kind of problems, even when a little change is done on the existing algorithm.

Whereas most stochastic search methods start with a single solution, genetic algorithms start with a population of solutions. An initial population is formed randomly or by means of a heuristic algorithm. Solutions are encoded in a form, which are called chromosomes. Each chromosome shows a complete solution to a problem. They are each assigned to a fitness score which represents the ability of chromosomes to compete for mating and staying alive.

Parents are picked up to mate according to their fitness values. The fitter chromosomes produce more offspring than the less fit chromosomes. The solution set is then imposed to crossover, mutation and inversion. These stochastic operators are required for diversifying the solution pool and especially getting better solutions. Since the size of the population should be maintained statically, some weak individuals in the population die, and better solutions thrive to stay alive. The cycle

continues until some certain number of iterations is executed or once the population converges. Convergence is defined as the progression of solutions towards uniformity. Similarity among fitness values increases as the population converges to the best fitness value obtained so far (Uçar, 2005).

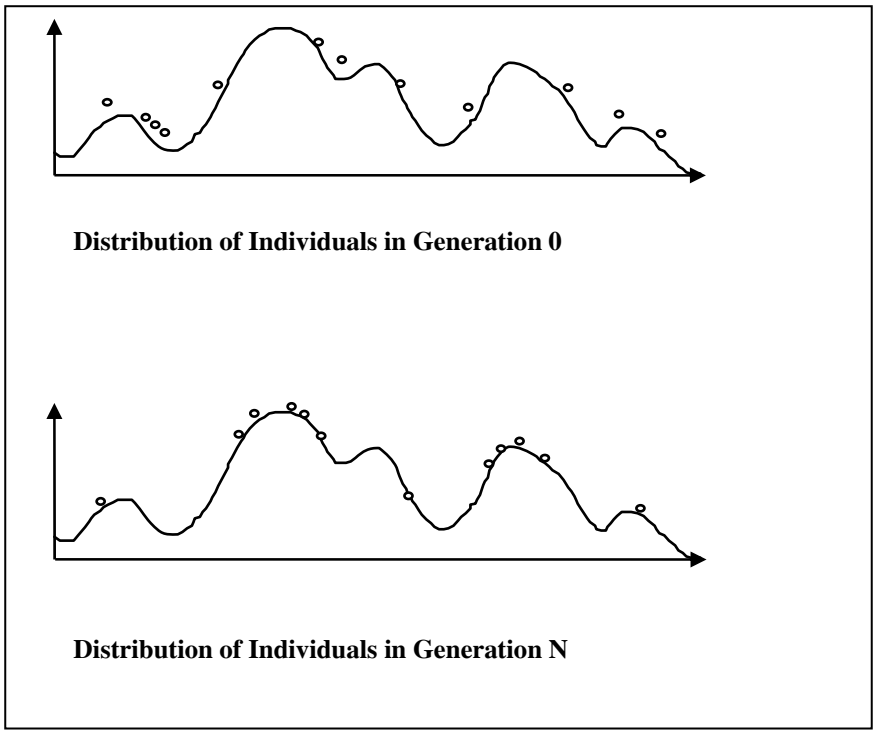

Figure III-1 Change in the distribution of individuals with no. of generations (Uçar, 2005)

III.2. THE PSEUDOCODE OF THE ALGORITHM

Genetic Algorithm ( )

{

Initialize population P of size λ /* a randomly generated population*/

Evaluate λ *individuals in P* /*check the fitness of each chromosome*/

While termination criteria not satisfied do

*{ Select 2** μ *individuals from P*

Crossover individuals to produce μ *offspring*

Mutate some individuals in μ

Add μ *offspring to* λ *individuals in P*

Evaluate (λ + μ) *individuals in P*

Select λ *individuals from* (λ + μ) *individuals in P }*

End Genetic Algorithm ( )}

III.3. REPRESENTATION

Any representation can be used for chromosomes such as; strings of bits, arrays, trees, lists, or any other object. Mutation and crossover operators are defined according to the representation used. For example, for permutation flow shop sequencing problem, each gene on a chromosome is represented as a list of the job numbers; e.g., in chromosome {9,2,4,1,5,7,3,6,10,8}, the numbers indicate the operating sequence of jobs on each n machines from 1 to n. In many application, string representation is used; e.g., {0,1,0,0,1,1,1}

A schema helps determining the similarities among chromosomes. The similar section of the chromosome is written neatly and the rest part is denoted with *. e.g., the sequences of genes on those two chromosomes are similar, which are to be {1, 4, 6, 3, 2, 5} and {5, 1, 6, 3, 2, 4}

In these two chromosomes, it is obviously seen that there is a similarity of genes at certain locations. The schema of can be represented as {*,*, 6, 3, 2, *}

As the number of schemas increases, the solution pool moves to uniformity; namely it converges. This means that the fitness of the chromosomes begins to stabilize, which helps the algorithm stop running.

Schema theorem has some formulations. With the calculation of the formulas, it helps to provide information about how GA works and to calculate the effect of selection, crossover and mutation.

III.4. SELECTION

Selection method is used for two objectives; for determining the mates to reproduce and for determining the fitter chromosomes which will be maintained in the next generation.

This method has a magnificent effect on the results. If the selector picks only the best individual, then the population will quickly converge to that best value. The selector should also pick individuals that are not so good, but have hopefully good genetic material to avoid from early convergence.

Selection is done according to the fitness scores. By using fitness scores, fitter chromosomes are chosen to reproduce and weaker ones are eliminated and hence the population is differentiated and diversified. There are many selection methods available.

III.4.1. Roulette Wheel Selection

Individuals have $\frac{f(i)}{\sum_i f(i)}$ probability to be chosen whereas f(i) denotes the fitness of that certain chromosome and ∑f(i) denotes the sum of the fitness of each chromosome in the population. The proportion is compared with a randomly generated number and the chromosomes are selected, whose fitness proportion is close to the generated value.

III.4.2. Ranking Selection

The fitness of the chromosomes is calculated and the values are sorted in descending ordered. Then, the selection is done downward.

III.4.3. Elitist Selection

Few of the best individuals are directly inserted to the mating pool. Another selection method is used for the rest of the pool.

III.4.4. Tournament Selection

In this selection method, the best being solution is picked up among k number of selected individuals and it is inserted to the mating pool. The best results are attained when k equals 2.

III.4.5. Steady-State Selection

Different selection strategies can be followed for both mating and replacement. e.g., Fitness of parents can be taken into account during mating whereas replacement can be done randomly or the selection can be done according to fitness for both stages, etc.

III.4.6. Stochastic Universal Selection

All individuals have the same probability to be selected.

III.5. CROSSOVER

In crossover, two individuals, called parents combine to produce two more individuals which are called the children. One chromosome exchanges its subpart with the latter, which is a mimicking of a biological recombination. But there are also asexual and single-child type crossovers. Crossover enables to move to promising regions of the solution space.

The main objective of crossover is to transfer the good characteristics of previous generation to the subsequent generation. Therefore, it matches generally good parents to produce better solutions.

III.5.1. Single Point Crossover

Parent chromosomes are broken from the same point and the alleles after that point are swapped between parents, e.g., let's say parent {a1,a2,a3,a4,a5,a6,a7} and {b1,b2,b3,b4,b5,b6,b7} chromosomes are broken after the third point. Then the produced chromosomes will be {a1,a2,a3,b4,b5,b6,b7} and {b1,b2,b3,a4,a5,a6,a7}

Goldberg [103] describes another single point crossover which performs well in flow shop sequencing problems. Both parents are broken randomly at a point.

P1 2 1 3 4 5 6 7 O1 2 1 4 3 6 7 5

*

P2 4 3 6 2 7 1 5 O2 4 3 2 1 5 6 7

As seen in the first offspring, the alleles 2 and 1 are erased from the second parent and the rest are directly replaced beyond the breaking point of the first parent without changing the sequence the second parent. The second offspring is reproduced in the same way.

2-point, 3- point and multi-point crossover have been developed from 1-point crossover.

III.5.2. Cycle Crossover (CX)

A single crossover point is selected. From starting at this point, elements from one parent is inherited to the offspring, as soon as the cycle is completed, the values are inherited from the other parent. Let's explain it on the example,

Parent 1	7 3 4 2 1 5 6
Crossover points	*
Parent 2	3 1 5 2 4 7 6
Offspring	7 3 4 2 1 5 6

Third point is selected as the crossover point. 4 is inherited from parent 1. We move on the second parent until we see 4, and inherit the across value to the offspring. This continues until a cycle is completed. The cycle is completed at location 3. After than the remaining loci is filled with the elements from the second parent.

III.5.3. Order Crossover (OX)

This is a 2-point crossover operator. The points are randomly selected. The offspring inherits the alleles between the selected points from one of the parents. The remaining locations are filled with the alleles from the alternate parent. The alleles are inherited from the beginning allele to the end if they don't appear in the offspring so far. Filling of offspring loci begins beyond the second crossover point.

Parent 1	7 3 4 2 1 5 6
Crossover points	* *
Parent 2	3 1 5 2 4 7 6
Offspring	1 6 5 2 4 7 3

III.5.4. Partially Mapped Crossover (PMX)

Two points are selected randomly. The elements among the points are inherited from one of the parents. Other unfilled loci are inherited from the alternate parent.

Parent 1	7 3 4 2 1 5 6
Crossover points	* *
Parent 2	3 1 5 2 4 7 6
Offspring	3 4 4 2 1 7 6

At this moment, if we met in the alternate parent the previously inherited elements, they are replaced with the other elements from the previous parent across them.

In the example, 4 duplicates; so the offspring is mutated and the second locus of the child chromosome is changed with 5; since 5 hasn't appeared in the sequence. So the new sequence is: Mutated offspring 3 5 4 2 1 7 6

III.5.5. Position-Based Crossover (PBX)

Some points are selected randomly and the alleles at these points are inherited from one of the parents to the offspring. The remaining gene loci are inherited from the latter parent. To avoid from the duplication of alleles, the gene values aligned with the crossover points should be replaced with the alleles across the points.

Parent 1	7 3 4 2 1 5 6
Crossover points	* * *
Parent 2	3 1 5 2 4 7 6
Offspring	7 1 4 2 5 5 6

As seen in the representation above, the allele 5 duplicates and 3, is not available in the offspring sequence. Let's do a mutation on the reproduced chromosome: Mutated offspring 7 1 4 2 3 5 6

III.6. MUTATION

Mutation changes the values of genes at some locations in the chromosome. It helps randomizing the search with a very low probability and finds solutions that cannot be encountered by crossover. It enables movement in the search space and restores lost information to the population.

Mutation has less impact near the beginning of a run, and more near the end while the crossover is more effective at the beginning and less at the end.

Min types of mutation operators are listed below:

III.6.1. Inversion

The genes in a randomly selected part of a chromosome is written in inverse sequence:

Chromosome	1	2	3	4	5	6	7	8	9
After Mutation	1	2	6	5	4	3	7	8	9

Figure III-2 Inversion mutation scheme (Şevkli, 2005)

III.6.2. Insertion

The gene in a randomly chosen position of a chromosome is inserted into another randomly selected position within the chromosome:

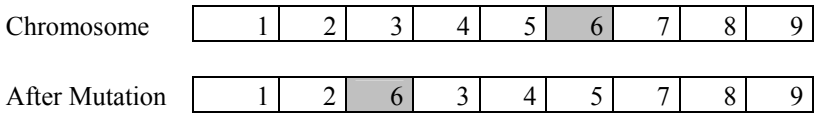

Figure III-3 Insertion mutation scheme (Şevkli, 2005)

III.6.3. Exchange

Two randomly selected genes in a chromosome are exchanged:

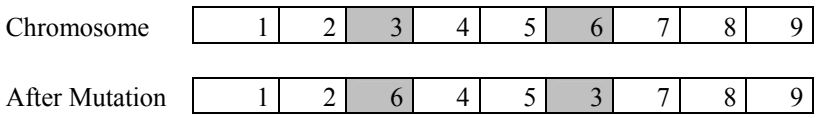

Figure III-4 Exchange mutation scheme (Şevkli, 2005)

III.7. TERMINATION CRITERIA

The algorithm is terminated whenever a certain number of iterations are reached or the population converges. Each iteration is called a generation. Typically a GA can be iterated from 50 to 500 or more generations.

IV. PARTICLE SWARM OPTIMIZATION

IV.1. AN OVERVIEW

Particle Swarm Optimization (PSO) is a population based metaheuristic which was developed by Eberhart and Kennedy in 1995 and introduced as an alternative to Genetic Algorithms (GA). It was inspired by the social behavior of flocking organisms such as bird swarms and fish shoals, which benefit from their previous experience or from the experience of the previous generation while they are searching for food and mate.

PSO is a rather successful method for the continuous optimization problems; however it is very difficult to adapt it for the discrete case. Researches were already done for the adaptation of the algorithm for the discrete case. These approaches can solve the combinatorial problems to some extent.

The PSO paradigm resembles to GA at some points. The initialization of the algorithm is done with a population of random solutions. It searches the optimal value by updating generations. Solutions are not generated by the crossover and mutation operators as in GAs. Instead, in PSO new generations are formed by means of velocity updates. The potential solutions, called particles, fly through the multi-dimensional search space, and follow the current optimum particles. Execution of the algorithm is terminated as soon as the maximum number of iteration is or maximum CPU time exceeded.

There is no replacement in PSO, all particles are kept in the population during the whole run. PSO does not incorporate the survival of the fittest, whereas all other evolutionary algorithms do.

Each particle has a velocity. Particles are carried to new positions with this velocity. The fitness values of particles are evaluated according to their positions at each iteration. The velocity, position and fitness of a particle are stored in a short term memory. The best position and fitness values of the particle are stored in the long term memory; which is named by Kennedy & Eberhart as autobiographical memory. The best experience stored in this memory is named as personal best; *pbest*.

The particle with the best fitness in the neighborhood is named as the local best; *lbest* and the best particle in the whole swarm is called as the global best; *gbest*.

PSO has two versions; the local version and the global version. According to the local neighborhood, each particle moves towards its best previous position, *pbest* and towards the best particle in its restricted neighborhood, *lbest*; rather than moving towards the best of the entire group, *gbest*. In the global neighborhood, each particle moves towards its best previous position, *pbest* and towards the best particle in the whole swarm, *gbest*.

There is a communication between particles, each particle shares its information with others. A particle exchanges its information with the particles in the neighborhood or a predetermined set of particles in the search space. Therefore; after some number of iterations the swarm loses its diversity and solutions progress to uniformity. If the convergence occurs too early, the probability of being stuck in local minima increases.

In recent years, PSO has been successfully applied in many areas. It solves a variety of optimization problems in a faster and cheaper way than the evolutionary algorithms in the early iterations, but its computational efficiency may reduce as the number of generations increases. In addition to this, PSO has few parameters to adjust. It works well for different kind of problems when the algorithm is slightly modified.

IV.2. THE PSEUDOCODE OF THE ALGORITHM

```
Initialize parameters
Initialize population
Find permutation
Evaluate
Do
{
        Find the personal best
        Find the global best
        Update velocity
        Update position
        Find permutation
        Evaluate
                Apply local search (optional)
         }
        While (Termination)
```

NOTATION

X_i^t : i[th] particle in the swarm at iteration t; $X_i^t = \left[x_{i1}^t, x_{i2}^t, .., x_{in}^t\right]$

x_{ij}^t : Position value of the i[th] particle with respect to the j[th] dimension ($j = 1,2,...,n$).

pop^t : Set of ρ particles in the swarm at iteration t, i.e., $pop^t = \left[X_1^t, X_2^t, ..., X_\rho^t\right]$

π_i^t : Permutation of jobs implied by the particle X_i^t; $\pi_i^t = \left[\pi_{i1}^t, \pi_{i2}^t, .., \pi_{in}^t\right]$

π_{ij}^t : Assignment of job *j* of the particle *i* in the permutation at iteration *t*.

V_i^t : Velocity of particle i at iteration t; $V_i^t = \left[v_{i1}^t, v_{i2}^t, ..., v_{in}^t\right]$

v_{ij}^t : Velocity of particle i at iteration t with respect to the j[th] dimension.

w^t : Inertia weight; a parameter to control the impact of the previous velocities on the current velocity.

P_i^t : The best position of the particle i with the best fitness until iteration t, personal best; $P_i^t = \left[p_{i1}^t, p_{i2}^t, ..., p_{in}^t\right]$

p_{ij}^t : Position value of the i[th] personal best with respect to the j[th] dimension ($j = 1,2,...,n$).

G^t : The best position of the globally best particle achieved so far, global best; $G^t = \left[g_1^t, g_2^t, ..., g_n^t\right]$

g_j^t : Position value of the global best with respect to the j[th] dimension ($j = 1,2,...,n$)

IV.3. ORIGINAL PSO ALGORITHM

Each particle updates its velocity and position according to its previous velocity and the distances of its current position from its own best experience and the group's best experience according to the equation 4.1.a given below:

$$v_{ij}^k = v_{ij}^{k-1} + c_1 r_1 (pb_{ij}^{k-1} - x_{ij}^{k-1}) + c_2 r_2 (gb_j^{k-1} - x_{ij}^{k-1}) \quad (4.1.a)$$

$$x_{ij}^k = x_{ij}^{k-1} + v_{ij}^k \quad (4.1.b)$$

Here, c_1 and c_2 are cognitive and social components respectively. These terms pull each particle to *pbest* and *gbest* locations. They are both set to 2 for almost all

applications; which is obtained from earlier experience. High or low values of these terms may hinder particles to reach the target.

r_1, r_2 are random numbers uniformly distributed in the interval [0,1]. Particles fly to new position with this velocity and their new position is calculated by the equation 4.1.b.

IV.4. SOLUTION REPRESENTATION

Solution representation is a very important issue in PSO algorithm. The representation changes depending on the type of the problem. For the PFSP, we present *n* number of dimensions for *n* number of jobs ($j = 1,..,n$). Each dimension in the sequence corresponds to a certain job. In addition, the particle $X_i^k = \left[x_{i1}^k,..,\ x_{in}^k\right]$ corresponds to the position values for *n* number of jobs in the PFSP problem. The position values of particles are in fact continuous. To discretize the positions, I use the SPV rule and by this way, determine processing sequence of jobs in the flow shop.

Table IV.1 illustrates the solution representation of particle X_i^t for the PSO algorithm for the PFSP together with its corresponding velocity and permutation. According to the proposed SPV rule, the smallest position value is $x_{i5}^t = -1.20$, so the dimension j=5 is assigned to be the first job $\pi_{i1}^t = 5$ in the permutation π_i^t; the second smallest position value is $x_{i2}^t = -0.99$, so the dimension j=2 is assigned to be the second job $\pi_{i2}^t = 2$ in the permutation π_i^t, and so on. In other words, dimensions are sorted according to the SPV rule.

This representation is unique in terms of finding new solutions since positions of each particle are updated at each iteration *k* in the PSO algorithm, thus resulting in different sequences at each iteration *k*.

Table IV.1 Solution Representation of a Particle

j	1	**2**	3	4	***5***	6
x_{ij}^k	1.80	**-0.99**	3.01	-0.72	***-1.20***	2.15
v_{ij}^k	3.89	2.94	3.08	-0.87	-0.20	3.16
s_{ij}^k	***5***	**2**	4	1	6	3

IV.5. INITIAL POPULATION

In PSO, the population is initialized randomly and the initial continuous position values are generated randomly using the following formula: $x_{ij}^0 = x_{\min} + (x_{\max} - x_{\min}) * U(0,1)$ where $x_{\min} = -1.0, x_{\max} = 1.0$. Initial continuous velocities are generated by similar formula as follows:
$v_{ij}^0 = v_{\min} + (v_{\max} - v_{\min}) * U(0,1)$ where $v_{\min} = -1.0, v_{\max} = 1.0$. $U(0,1)$ is a uniform random number between 0 and 1.

The complete computational procedure of the PSO algorithm for the PFSP can be summarized as follows:

Step 1: Initialization

- Set *k*=0, *m*=twice the number of dimensions.
- Generate *m* particles randomly as explained before, $\{X_i^0, i = 1,..,m\}$ where $X_i^0 = [x_{i1}^0,..,x_{in}^0]$.
- Generate initial velocities of particles randomly $\{V_i^0, i = 1,..,m\}$ where $V_i^0 = [v_{i1}^0,..,v_{in}^0]$
- Apply the SPV rule to find the sequence $S_i^0 = [s_{i1}^0,..,s_{in}^0]$ of particle X_i^0 for $i = 1,..,m$.
- Evaluate each particle *i* in the swarm using the objective function f_i^0 for $i = 1,..,m$.
- For each particle *i* in the swarm, set $PB_i^0 = X_i^0$, where $PB_i^0 = [pb_{i1}^0 = x_{i1}^0,.., pb_{in}^0 = x_{in}^0]$ along with its best fitness value, $f_i^{pb} = f_i^0$ for $i = 1,..,m$.
- Find the best fitness value $f_l^0 = \min\{f_i^0\}$ for $i = 1,..,m$ with its corresponding position X_l^0.
- Set global best to $GB^0 = X_l^0$ where $GB^0 = [gb_1 = x_{l,1},.., gb_n = x_{l,n}]$ with its fitness value $f^{gb} = f_l^0$

Step 2: Update iteration counter

- $k = k + 1$

Step3: Update inertia weight

- $w^k = w^{k-1} * \alpha$ where α is decrement factor.

Step 4: Update velocity

$$v_{ij}^k = w^{k-1} v_{ij}^{k-1} + c_1 r_1 \left(pb_{ij}^{k-1} - x_{ij}^{k-1} \right) + c_2 r_2 \left(gb_j^{k-1} - x_{ij}^{k-1} \right)$$

Step 5: Update position

- $x_{ij}^k = x_{ij}^{k-1} + v_{ij}^k$

Step 6: Find Sequence

- Apply the SPV rule to find the sequence $S_i^k = \left[s_{i1}^k, .., s_{in}^k \right]$ for $i = 1, .., m$.

Step 7: Update personal best

- Each particle is evaluated by using its sequence to see if personal best will improve. That is, if $f_i^k < f_i^{pb}$ for $i = 1, .., m$, then personal best is updated as $PB_i^k = X_i^k$ and $f_i^{pb} = f_i^k$ for $i = 1, .., m$.

Step 8: Update global best

- Find the minimum value of personal best.

$$f_l^k = \min \left\{ f_i^{pb} \right\} for \quad i = 1, .., m, l \in \{ i; i = 1, .., m \}$$

- If $f_l^k < f^{gb}$, then the global best is updated as $GB^k = X_l^k$ and $f^{gb} = f_l^k$

Step 9: **Stopping criterion**

- If the number of iteration exceeds the maximum number of iteration, or maximum CPU time, then stop, otherwise go to step 2.

IV.6. MAXIMUM VELOCITY

The velocities of particles are constrained to a maximum velocity, V_{max}. If a velocity on a dimension of a particle exceeds V_{max}, then it is limited to V_{max}. V_{max} controls the exploration and exploitation ability of a particle. It helps to search the regions between the current position and the target position.

Fine-tuning V_{max} is so important that a large value of V_{max} facilitates global exploration, while a smaller V_{max} encourages local exploitation. If V_{max} is set too high or too small, the particles can't explore the search space sufficiently and they could stuck at local optima.

IV.7. INERTIA WEIGHT

Eberhart & Shi introduced a new concept to PSO in 1998; the inertia weight, *w* which highly increased the performance of PSO in a number of applications. Before, PSO was not searching neighbors sufficiently. Dynamically adjusting the velocity by means of *w* provided the local search.

The inertia weight controls the effect of previous velocity of the particle to its current velocity as seen in the formulas below:

$$v_{ij}^{k} = w^{k-1}v_{ij}^{k-1} + c_1 r_1\left(pb_{ij}^{k-1} - x_{ij}^{k-1}\right) + c_2 r_2\left(gb_{j}^{k-1} - x_{ij}^{k-1}\right) \quad (4.3.a)$$

$$x_{ij}^{k} = x_{ij}^{k-1} + v_{ij}^{k} \quad (4.3.b)$$

where $r_1, r_2 \sim$ Uniform (0,1), w: inertia weight

and

$w^k = w^{k-1} * \alpha$, where α is the decrement factor

Setting high values to *w* at the beginning and small values at the end of the search is found to be better. It is generally reduced linearly from 1.2 to 0.4 during a run, but these values may change from application to application.

When suitably set, the inertia weight helps to balance the local and global exploration, thus the optimal value can be obtained in a few iterations. High values encourage global exploration, while low values facilitate local exploitation.

IV.8. CONSTRICTION FACTOR

Maurice Clerc has introduced in 1999 the *constriction factor, K*, which highly increases the performance of the algorithm by constraining and controlling the velocity of the particles. Shi and Eberhart found that when the constriction factor is used with V_{max} constraint, the performance PSO improves.

The velocity formula using *K* is stated in equation 4.4.a. Clerc used *K* as 0.729 in his calculations.

$$v_{id}^{k} = \kappa\left[v_{ij}^{k-1} + c_1 r_1\left(pb_{ij}^{k-1} - x_{ij}^{k-1}\right) + c_2 r_2\left(gb_{j}^{k-1} - x_{ij}^{k-1}\right)\right] \quad (4.4.a)$$

$$\kappa = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \text{ where } \varphi = c_1 + c_2, \quad \varphi > 4 \quad (4.4.b)$$

Both the constriction factor, *K*, and the inertia weight, *w*, are used to control the velocities of particles. Therefore, they both prevent the particles from explosion.

V. VARIABLE NEIGHBORHOOD SEARCH

V.1. AN OVERVIEW

In recent years, several general heuristics (or metaheuristics) have been proposed which extend local search in various ways and avoid being trapped in local optima with a poor value. A simple and effective metaheuristic may be obtained by proceeding to a systematic change of neighborhood within a local search algorithm. Mladenovic and Hansen call this approach the Variable Neighborhood Search (VNS) (Mladenovic and Hansen, 1997). The difference of VNS from most other local search methods is that VNS does not follow a trajectory, but explores increasingly distant neighborhoods of the current solution, and jumps from there to a new one if and only if a better solution is obtained.

The Variable Neighborhood Search (VNS) is rapidly becoming a well-established method in metaheuristics (see for instance (Framinan et al., 2002)). VNS is based on a simple and effective idea: a systematic change of neighborhood within a local search algorithm. To apply the method we first need to define different neighborhoods for our problem.

As is stated by Hansen and Mladenovic,VNS is based on three principles:

1. A local minimum with respect to one neighborhood is not necessarily so with another.

2. A global minimum is a local minimum with respect to all possible neighborhood structures.

3. For many problems, local minima with respect to one or several neighborhoods are relatively close to each other.

Principle 2 is true for all the optimization problems. However, principles 1 and 3 may or may not hold depending on the problem at hand (Garcia et al., 2006).

V.2. NEIGHBOURHOOD STRUCTURES

The performance of a metaheuristic algorithm significantly depends on the efficiency of the neighbourhood structure. The solutions are determined to move with the neighbourhood structure. In this study, the following two neighbourhood structures are employed:

Exchange is a function used to move around in which any two randomly selected operations are simply swapped. In Figure V-1, B and E are selected randomly and swapped.

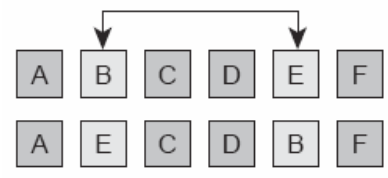

Figure V-1 Exchange function (Uysal, 2006)

Insert is another fine-tuning function that inserts a randomly chosen gene in front or back of another randomly chosen gene. In Figure V-2, B and E are selected randomly. B is inserted in front of E.

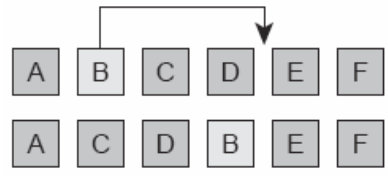

Figure V-2 Insert function (Uysal, 2006)

There are many neighbourhood structures reported in the literature, but I preferred these two, due to their simplicity, ease of use and reasonable efficiency.

V.3. THE VNS ALGORITHM

VNS is a simple and effective search procedure that proceeds to a systematic change of neighbourhood.

The local search for the PFS problem is applied to the job repetition π^t of the global best solution at each iteration *t*. The performance of the local search algorithm depends on the choice of the neighborhood structure.

Local search in this hesis is based on the *interchange+insert* variant of the Variable Neighborhood Search (VNS) method. For the PFS problem, following two neighborhood structures are employed:

- Interchange two jobs between η^{th} and κ^{th} dimensions, $\eta \neq \kappa$ *(Interchange)*
- Remove the job at the η^{th} dimension and insert it in the κ^{th} dimension $\eta \neq \kappa$ *(Insert)*

Pseudo code of the local search is given in Figure V-3 where η and κ are the random integer numbers between 1 and *n*. For convenience, $s = insert(s_0, \eta, \kappa)$ means removing the job from η^{th} dimension in the job repetition s_0 and inserting it in the κ^{th} dimension in the job repetition s_0, thus resulting in a job repetition s. In case of modifying the job repetition s_0, two swap and two interchange operations are used, to diversify the global best solution before applying the local search.

This modification is significantly important to direct the search towards the global optima by modifying the global best solution, since otherwise the global best solution remains the same after some iterations.

```
s0= π^t , job repetition  of global best G^t ;
do{
   u=rnd(1,nm); v=rnd(1,nm) η ≠ κ
  s=modify(s0, u, v)
   inloop=0;
   do{
      kcount=0; max_method=2;
     do{
        u=rnd(1,n); v=rnd(1,n) η ≠ κ
       if (kcount=0) then s1=interchange(s, u, v)
       if (kcount=1) then s1=insert(s, u, v)
     if   (f(s1)≤ f(s))   then {
                          kcount=0;
                          s= s1;}
     else { kcount++;}
     while (kcount<max_method)
   inloop++;
   while (inloop<n*(n-1)
   outloop++;
  if   (f(s1)≤ f(π^t))   then {
                          π^t =s;
                          repair(G^t );}
```

Figure V-3 Pseudo code of VNS local search employed

VI. EXPERIMENTATION

In this study, PSO and GA algorithms are applied to the Permutation Flowshop Scheduling Problem with the bicriteria of minimizing makespan(C_{max}) and maximum tardiness(T_{max}). That is, in my study, the objective function is:

Minimize $\mathbf{f = \lambda C_{max} + (1- \lambda) T_{max}}$, where λ is between 0 and 1.

For this model, I performed successive computer runs. Computer codes for different versions of the algorithms are run on PCs with Intel Pentium IV 2.66 GHz processors and 512 MB memory.

In GA, population size is taken as twice the number of jobs, crossover probability is taken as 1.0. For the mate selection, one individual is selected randomly and the other is selected by using the tournament selection method with size of 2. Again the tournament selection with size of 2 is employed for constructing the next generation.

In line with the PSO algorithm, GA that employed the simple exchange operator as a mutation scheme is denoted by GA_{pure} , while the GA that employed the VNS local search is denoted by GA_{vns} . The performance of the GA and PSO algorithms is evaluated by using the benchmark suite of (Demirkol et al., 1998).

Demirkol et al. have provided an extensive set of randomly generated test problems for minimizing makespan and maximum lateness in flowshops and job shops. The total number of problems they generated is 600 including three different types of routings, four different due date configurations and a variety of problem sizes ranging from 20 to 50 jobs with 15 and 20 machines. They have only provided 40 problem instances for makespan criterion and 160 problem instances for maximum lateness criterion for the flowshop problems.

They used five dispatching rules and three shifting bottleneck methods to solve the makespan problems and eleven dispatching rules and three shifting bottleneck methods to solve the maximum lateness problems.

They have reported only the best solution provided by any of these methods, which can be found in

http://palette.ecn.purdue.edu/~uzsoy2/benchmark/problems.html

The PSO algorithm for the PFSP is coded in C programming language. In addition, a traditional genetic algorithm (GA) with tournament selection is also coded in C to compare the performances of the two population based methods. The C codes of the algorithms can be found in the Appendix

I carried out parametric analysis for the *probability of mutation (Pmutation)* parametre of the proposed GA algorithm. I tested the Pmutation values of 0.1, 0.2 and 0.3, for 1000, 1500 and 2000 number of generations and for five different lambda(λ) values (namely, 0.00, 0.25, 0.50, 0.75 and 1.00). So, I made experimentation for all combinations of these values, namely 1000&0.1, 1000&0.2, 1000&0.3, 1500&0.1, 1500&0.2, 1500&0.3, 2000&0.1, 2000&0.2 and 2000&0.3.

As a result, I found out that the Pmutation value of 0.3 provided the best fitness values for all generation numbers. So I used this Pmutation parameter for GA throughout my study. Detailed information about the parametric analysis for GA can be found in the Appendix.

The "pure" versions of the PSO and GA algorithms are run for five different generation numbers (namely maxgen values of 1000, 1500, 2000, 2500 and 3000) and for the five different lambda(λ) values (namely, 0.00, 0.25, 0.50, 0.75 and 1.00).

As the second step, both GA and PSO algorithms are also run by embedding a simple but very efficient local search by Mladenovic and Hansen(42), so called Variable Neighborhood Search (VNS). Then, these *hybrid* versions of the PSO and GA are also applied to the test problems in the benchmark suite of (Demirkol et al., 1998). These runs lasted too long compared to the pure versions and the computers were kept busy for days, since I performed parallel runs on a sufficient number of computers. So, I selected the smallest and biggest generation numbers only (namely, maxgen values of 1000 and 3000), for the complete set of lambda values (0.00, 0.25, 0.50, 0.75 and 1.00). 10 replications were made for each of the 160 problem instances.

As the last step, the performances of the "pure" and "hybrid (with VNS)" versions of the PSO and GA algorithms were compared respectively. The two algorithms were compared in terms of not only the solution quality(e.g. the fitness values), but also the CPU times (that is, how long it took for them to find a solution).

Besides comparing the algorithms for their overall performance on the 160 problem instances taken from the benchmark suite, I also made detailed comparisons for problem sets with diferent sizes. The total of 160 problems consists of 8 different problem sets, the smallest being the 20x15 (that is, 20 jobs & 5 machines), ending up with the largest set of 50x20. Detailed Excel sheets for all of these comparisons can be found in the Appendix.

I performed statistical T-tests, to find out if one of the algorithms performed significantly better for a certain instance. Confidence intervals of 90%, 95% and 99.5% were used.

Summary tables for all these result are given below.

Note:

≈ : Two algorithms are not significantly different

NA: Not Available.

FITNESS COMPARISON RESULTS

For the "pure" versions of the algorithms:

Table VI-01 Pure Versions-Overall Fitness Results

lambda	0.00					0.25					0.50					0.75					1.00				
maxgen	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000
PSO better	109	103	96	92	87	116	110	107	101	97	114	111	101	91	91	76	68	63	52	50	49	36	33	24	20
GA better	51	57	64	68	73	44	50	53	59	63	46	49	59	69	69	84	92	97	108	110	110	124	127	136	140
PSO=GA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Best PSO (%)	21.86	21.54	21.03	19.43	18.64	15.13	14.10	14.20	14.19	14.19	9.32	9.08	8.93	8.58	8.58	3.71	3.69	3.33	3.32	3.14	1.30	1.20	1.08	1.06	0.97
Best GA (%)	4.71	6.07	6.34	6.35	6.62	2.02	2.06	2.36	2.38	2.48	2.59	2.74	2.80	2.87	2.89	2.11	2.16	2.19	2.28	2.36	2.15	2.11	2.11	2.20	2.32
Average (%)	-2.76	-2.15	-1.77	-1.48	-1.21	-2.86	-2.61	-2.43	-2.24	-2.13	-1.63	-1.48	-1.34	-1.22	-1.14	-0.06	0.04	0.14	0.21	0.27	0.33	0.44	0.53	0.59	0.65

For the pure versions of the algorithms, as seen in Table VI-01 and VI-02, PSO outperforms GA for lambda=0.00, 0.25 and 0.50, but for lambda=0.75 and 1.00, GA finds better results. But there is no big difference; the average difference between the two algorithms is always lower than 3 %. It should be noted here that, a minus(-) sign in a table means that PSO performs better (i.e. finds a smaller value) for that occasion.

Average percent deviation fitness results for different problem sizes:

Table VI-02 Pure Versions-Average Percent Deviation

lambda	0.00					0.25					0.50					0.75					1.00				
maxgen	**1000**	**1500**	**2000**	**2500**	**3000**	**1000**	**1500**	**2000**	**2500**	**3000**	**1000**	**1500**	**2000**	**2500**	**3000**	**1000**	**1500**	**2000**	**2500**	**3000**	**1000**	**1500**	**2000**	**2500**	**3000**
20x15	0.06	0.44	0.66	0.86	1.12	-0.67	-0.28	0.07	0.24	0.42	-0.80	-0.58	-0.49	-0.36	-0.28	-0.13	0.02	0.13	0.23	0.29	0.39	0.56	0.61	0.68	0.74
20x20	-0.71	-0.27	-0.05	0.21	0.47	-1.11	-0.79	-0.60	-0.50	-0.38	-0.38	-0.22	-0.07	-0.01	0.03	-0.11	0.10	0.21	0.30	0.35	0.02	0.17	0.27	0.40	0.46
30x15	-3.12	-2.63	-2.32	-1.97	-1.76	-2.67	-2.44	-2.26	-1.92	-1.78	-1.54	-1.28	-1.01	-0.93	-0.82	-0.18	-0.06	0.12	0.20	0.24	0.12	0.28	0.42	0.53	0.61
30x20	-1.60	-1.07	-0.63	-0.35	-0.19	-2.21	-1.93	-1.76	-1.54	-1.47	-1.05	-0.93	-0.79	-0.63	-0.51	0.05	0.19	0.30	0.35	0.40	0.13	0.23	0.35	0.41	0.49
40x15	-4.48	-3.74	-3.16	-2.81	-2.47	-3.55	-3.32	-3.21	-2.94	-2.79	-2.25	-2.10	-1.97	-1.83	-1.70	-0.26	-0.17	-0.11	-0.04	0.01	0.54	0.63	0.70	0.71	0.76
40x20	-2.82	-2.28	-2.01	-1.78	-1.46	-3.06	-2.86	-2.71	-2.58	-2.46	-2.24	-2.08	-1.93	-1.76	-1.68	0.07	0.13	0.23	0.26	0.30	0.39	0.48	0.56	0.64	0.71
50x15	-4.79	-4.07	-3.47	-3.18	-2.87	-4.78	-4.65	-4.51	-4.38	-4.33	-2.20	-2.19	-2.11	-1.99	-1.91	0.11	0.14	0.20	0.30	0.40	0.60	0.70	0.72	0.76	0.82
50x20	-4.62	-3.58	-3.16	-2.79	-2.54	-4.80	-4.57	-4.46	-4.33	-4.25	-2.59	-2.50	-2.40	-2.29	-2.22	-0.06	-0.06	0.03	0.09	0.13	0.46	0.51	0.56	0.60	0.63

In Table VI-03, you can see the t-values calculated for different problem sets and generation numbers. As known, t values are calculated as: [(Average of the differences of the two algorithms) / [(standart deviation of the differences of the two algorithms) / (square root of the problem size (n=20 in our case))].

Table VI-03 Pure Versions - t Values

Lambda	0.00					0.25					0.50					0.75					1.00				
Maxgen	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000
20x15	0.01	1.11	1.87	2.54	3.18	-1.42	-0.74	0.16	0.61	1.17	-2.38	-1.85	-1.54	-1.16	-0.93	-0.98	0.09	0.95	1.66	2.13	2.31	3.55	3.96	4.43	4.78
20x20	-1.47	-0.61	-0.08	0.74	1.68	-2.91	-2.18	-1.64	-1.38	-1.01	-2.04	-1.08	-0.36	-0.05	0.16	-0.62	0.64	1.38	2.02	2.38	0.15	1.35	2.29	3.55	4.29
30x15	-3.32	-3.15	-2.84	-2.48	-2.29	-3.59	-3.28	-3.09	-2.72	-2.54	-2.85	-2.43	-1.94	-1.78	-1.60	-0.69	-0.31	0.32	0.65	0.81	0.76	1.69	2.65	3.24	3.59
30x20	-2.05	-1.44	-1.02	-0.63	-0.41	-3.22	-2.81	-2.56	-2.28	-2.20	-2.61	-2.37	-2.08	-1.67	-1.39	0.25	0.92	1.45	1.73	2.00	1.05	1.80	2.85	3.19	3.76
40x15	-3.50	-3.24	-2.95	-2.73	-2.52	-3.34	-3.18	-3.03	-2.93	-2.84	-3.44	-3.24	-3.11	-2.92	-2.74	-0.89	-0.61	-0.41	-0.17	-0.02	4.36	5.38	6.12	6.07	6.63
40x20	-3.28	-2.77	-2.55	-2.29	-2.05	-3.39	-3.30	-3.15	-3.00	-2.85	-3.31	-3.08	-2.84	-2.68	-2.57	0.14	0.39	0.79	0.89	1.01	2.65	3.12	3.74	4.48	5.03
50x15	-2.81	-2.48	-2.20	-2.13	-1.98	-4.03	-3.93	-3.82	-3.72	-3.67	-2.89	-2.91	-2.79	-2.67	-2.57	0.44	0.54	0.78	1.22	1.65	5.00	5.50	5.73	6.03	6.45
50x20	-4.16	-3.74	-3.38	-3.18	-2.88	-4.23	-4.13	-4.07	-3.98	-3.92	-4.07	-4.10	-4.02	-3.83	-3.73	-0.31	-0.29	0.02	0.26	0.39	2.86	3.15	3.67	3.95	4.48

Hypothesis Testing:

H_0: Two algorithms are not significantly different

H_1: One algorithm performs significantly better

t-test critical values for n=20

0.1 (90%)	0.05 (95%)	0.005 (99.5%)
1.3277	**1.7291**	**2.8609**

In the following five tables (Table VI-04 to Table VI-08) , you can see the results of the hypothesis testing, that is if one algorithm performs significantly better or not. It can be easily noticed how the dominance of PSO shades away as lambda increases.

Table VI-04 Pure Versions, t-Test Results, lambda=0.00

lambda	0.00														
maxgen	1000			1500			2000			2500			3000		
CI	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%
20x15	≈	≈	≈	≈	≈	≈	GA	GA	≈	GA	GA	≈	GA	GA	GA
20x20	PSO	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	PSO	≈	≈
30x15	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	≈	PSO	PSO	≈	PSO	PSO	≈
30x20	PSO	PSO	≈	PSO	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈
40x15	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	≈	PSO	PSO	≈
40x20	PSO	PSO	PSO	PSO	PSO	≈	PSO	PSO	≈	PSO	PSO	≈	PSO	PSO	≈
50x15	PSO	PSO	≈	PSO	PSO	≈	PSO	PSO	≈	PSO	PSO	≈	PSO	PSO	≈
50x20	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO

Table VI-05 Pure Versions, t-Test Results, lambda=0.25

lambda	0.25														
maxgen	1000			1500			2000			2500			3000		
CI	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%
20x15	PSO	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈
20x20	PSO	PSO	PSO	PSO	PSO	≈	PSO	≈	≈	PSO	≈	≈	≈	≈	≈
30x15	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	≈	PSO	PSO	≈
30x20	PSO	PSO	PSO	PSO	PSO	≈	PSO	PSO	≈	PSO	PSO	≈	PSO	PSO	≈
40x15	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	≈
40x20	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	≈
50x15	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO
50x20	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO

Table VI-06 Pure Versions, t-Test Results, lambda=0.50

lambda	0.50														
maxgen	1000			1500			2000			2500			3000		
CI	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%
20x15	PSO	PSO	≈	GA	GA	≈	GA	≈	≈	≈	≈	≈	≈	≈	≈
20x20	PSO	PSO	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈
30x15	PSO	PSO	≈	PSO	PSO	≈	GA	GA	≈	GA	GA	≈	GA	≈	≈
30x20	GA	GA	≈	GA	GA	≈	GA	GA	≈	GA	≈	≈	GA	≈	≈
40x15	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	PSO	GA	GA	≈
40x20	GA	GA	GA	GA	GA	GA	GA	GA	≈	GA	GA	≈	GA	GA	≈
50x15	GA	GA	GA	GA	GA	GA	GA	GA	≈	GA	GA	≈	GA	GA	≈
50x20	PSO	PSO	PSO	PSO	PSO	PSO	GA	GA	GA	GA	GA	GA	GA	GA	GA

Table VI-07 Pure Versions, t-Test Results, lambda=0.75

Lambda	0.75														
maxgen	1000			1500			2000			2500			3000		
CI	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%
20x15	≈	≈	≈	≈	≈	≈	≈	≈	≈	GA	≈	≈	GA	GA	≈
20x20	≈	≈	≈	≈	≈	≈	GA	≈	≈	GA	GA	≈	GA	GA	≈
30x15	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈
30x20	≈	≈	≈	≈	≈	≈	GA	≈	≈	GA	≈	≈	GA	GA	≈
40x15	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈
40x20	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈
50x15	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	GA	≈	≈
50x20	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈	≈

Table VI-08 Pure Versions, t-Test Results, lambda=1.00

lambda	1.00														
maxgen	1000			1500			2000			2500			3000		
CI	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%	90%	95%	99.5%
20x15	GA	GA	≈	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA
20x20	≈	≈	≈	GA	≈	≈	GA	GA	≈	GA	GA	GA	GA	GA	GA
30x15	≈	≈	≈	GA	≈	≈	GA	GA	≈	GA	GA	GA	GA	GA	GA
30x20	≈	≈	≈	GA	GA	≈	GA	GA	≈	GA	GA	GA	GA	GA	GA
40x15	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA
40x20	GA	GA	≈	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA
50x15	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA
50x20	GA	GA	≈	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA	GA

For the hybrid (with VNS) versions of the algorithms:

Table VI-09 Hybrid Versions, Overall Fitness Results

lambda	0.00		0.25		0.50		0.75		1.00	
maxgen	1000	3000	1000	3000	1000	3000	1000	3000	1000	3000
PSO better	112	105	131	122	138	137	136	132	151	151
GA better	15	14	21	21	16	13	23	22	7	7
PSO=GA	33	41	8	17	6	10	1	6	2	2
Best PSO (%)	-1.97	-1.69	-2.05	-1.56	-1.14	-0.86	-1.00	-0.88	-1.46	-1.54
Best GA (%)	0.51	0.21	0.22	0.23	0.42	0.31	0.60	0.48	0.18	0.16
Average (%)	-0.32	-0.27	-0.27	-0.20	-0.23	-0.19	-0.23	-0.19	-0.31	-0.30

As seen in Table VI-09 and VI-10, for the hybrid versions PSO outperforms GA for all lambda values. But the differenre here is much smaller than the pure versions, the average difference between the two algorithms is always lower than 0.5%.

Average percent deviation fitness results for different problem sizes:

Table VI-10 Hybrid Versions-Average Percent Deviation

lambda	0.00		0.25		0.50		0.75		1.00	
maxgen	1000	3000	1000	3000	1000	3000	1000	3000	1000	3000
20x15	-0.009	-0.008	-0.077	-0.037	-0.122	-0.069	-0.050	-0.054	-0.057	-0.074
20X20	-0.009	-0.045	-0.066	-0.045	-0.160	-0.103	-0.092	-0.032	-0.127	-0.078
30x15	-0.389	-0.257	-0.242	-0.152	-0.204	-0.167	-0.265	-0.198	-0.325	-0.262
30x20	-0.161	-0.138	-0.161	-0.131	-0.119	-0.084	-0.163	-0.147	-0.206	-0.205
40x15	-0.454	-0.387	-0.363	-0.289	-0.386	-0.275	-0.325	-0.233	-0.451	-0.512
40x20	-0.350	-0.326	-0.317	-0.232	-0.216	-0.218	-0.230	-0.255	-0.341	-0.319
50x15	-0.553	-0.508	-0.519	-0.333	-0.320	-0.312	-0.364	-0.347	-0.537	-0.543
50x20	-0.608	-0.489	-0.391	-0.354	-0.313	-0.303	-0.356	-0.279	-0.422	-0.374

In Table VI-11, you can see the t-values calculated for different problem sets and generation numbers.

Table VI-11 Hybrid Versions - t Values

lambda	0.00		0.25		0.50		0.75		1.00	
maxgen	1000	3000	1000	3000	1000	3000	1000	3000	1000	3000
20x15	-0.503	-0.644	-2.717	-2.114	-3.349	-2.782	-1.087	-2.375	-2.159	-4.321
20X20	-1.126	-2.029	-2.639	-3.851	-2.579	-2.391	-3.021	-1.304	-3.758	-3.592
30x15	-3.805	-3.747	-4.387	-3.821	-5.183	-5.181	-4.586	-4.036	-6.767	-6.472
30x20	-3.752	-3.148	-3.912	-3.475	-3.559	-3.001	-4.727	-4.965	-6.648	-6.854
40x15	-3.601	-4.920	-6.072	-5.074	-6.673	-4.701	-5.126	-4.623	-8.499	-5.852
40x20	-4.914	-5.466	-4.583	-3.861	-6.341	-4.845	-7.077	-8.647	-9.873	-12.227
50x15	-4.812	-4.429	-5.024	-3.621	-5.364	-6.390	-6.603	-5.365	-7.380	-7.200
50x20	-6.112	-6.152	-4.946	-6.253	-4.709	-5.310	-6.974	-6.475	-8.783	-7.772

In the following five tables (Table VI-12 to Table VI-16) , you can see the results of the hypothesis testing, that is if one algorithm performs significantly better or not. It can be easily noticed how PSO remains significantly better for all lambda values.

Table VI-12 Hybrid Versions, t-Test Results, lambda=0.00

lambda	0.00					
maxgen	1000			3000		
CI	90%	95%	99.5%	90%	95%	99.5%
20x15	≈	≈	≈	≈	≈	≈
20x20	≈	≈	≈	PSO	PSO	≈
30x15	PSO	PSO	PSO	PSO	PSO	PSO
30x20	PSO	PSO	PSO	PSO	PSO	PSO
40x15	PSO	PSO	PSO	PSO	PSO	PSO
40x20	PSO	PSO	PSO	PSO	PSO	PSO
50x15	PSO	PSO	PSO	PSO	PSO	PSO
50x20	PSO	PSO	PSO	PSO	PSO	PSO

Table VI-13 Hybrid Versions, t-Test Results, lambda=0.25

lambda	0.25					
maxgen	1000			3000		
CI	90%	95%	99.5%	90%	95%	99.5%
20x15	PSO	PSO	≈	PSO	PSO	≈
20x20	PSO	PSO	≈	PSO	PSO	PSO
30x15	PSO	PSO	PSO	PSO	PSO	PSO
30x20	PSO	PSO	PSO	PSO	PSO	PSO
40x15	PSO	PSO	PSO	PSO	PSO	PSO
40x20	PSO	PSO	PSO	PSO	PSO	PSO
50x15	PSO	PSO	PSO	PSO	PSO	PSO
50x20	PSO	PSO	PSO	PSO	PSO	PSO

Table VI-14 Hybrid Versions, t-Test Results, lambda=0.50

lambda	0.50					
maxgen	1000			3000		
CI	90%	95%	99.5%	90%	95%	99.5%
20x15	PSO	PSO	PSO	PSO	PSO	≈
20x20	PSO	PSO	≈	PSO	PSO	≈
30x15	PSO	PSO	PSO	PSO	PSO	PSO
30x20	PSO	PSO	PSO	PSO	PSO	PSO
40x15	PSO	PSO	PSO	PSO	PSO	PSO
40x20	PSO	PSO	PSO	PSO	PSO	PSO
50x15	PSO	PSO	PSO	PSO	PSO	PSO
50x20	PSO	PSO	PSO	PSO	PSO	PSO

Table VI-15 Hybrid Versions, t-Test Results, lambda=0.75

lambda	0.75					
maxgen	1000			3000		
CI	90%	95%	99.5%	90%	95%	99.5%
20x15	≈	≈	≈	PSO	PSO	≈
20x20	PSO	PSO	PSO	≈	≈	≈
30x15	PSO	PSO	PSO	PSO	PSO	PSO
30x20	PSO	PSO	PSO	PSO	PSO	PSO
40x15	PSO	PSO	PSO	PSO	PSO	PSO
40x20	PSO	PSO	PSO	PSO	PSO	PSO
50x15	PSO	PSO	PSO	PSO	PSO	PSO
50x20	PSO	PSO	PSO	PSO	PSO	PSO

Table VI-16 Hybrid Versions, t-Test Results, lambda=1.00

lambda	1.00					
maxgen	1000			3000		
CI	90%	95%	99.5%	90%	95%	99.5%
20x15	PSO	PSO	≈	PSO	PSO	PSO
20x20	PSO	PSO	PSO	PSO	PSO	PSO
30x15	PSO	PSO	PSO	PSO	PSO	PSO
30x20	PSO	PSO	PSO	PSO	PSO	PSO
40x15	PSO	PSO	PSO	PSO	PSO	PSO
40x20	PSO	PSO	PSO	PSO	PSO	PSO
50x15	PSO	PSO	PSO	PSO	PSO	PSO
50x20	PSO	PSO	PSO	PSO	PSO	PSO

CPU-TIME COMPARISON RESULTS

For the "pure" versions of the algorithms:

Table VI-17 Pure Versions, Overall CPU-Time Results

lambda	0.00					0.25					0.50					0.75					1.00				
maxgen	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000	1000	1500	2000	2500	3000
PSO better	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160	160
GA better	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PSO=GA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Best PSO (%)	44.88	41.86	40.44	40.07	40.20	75.51	37.20	40.99	38.63	41.39	42.11	42.86	40.18	43.36	43.26	38.53	38.50	37.27	38.77	37.56	40.18	37.95	38.16	37.72	40.12
Best GA (%)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Average (%)	28.83	28.82	28.61	28.27	28.23	35.77	26.58	26.27	26.36	26.19	31.40	31.06	29.62	30.11	30.14	27.01	26.92	25.83	25.99	25.77	27.41	26.59	26.33	26.40	27.86

In the tables VI-17 to VI-20, CPU-time comparisons are given for the pure and hybrid versions of the algorithms. As seen, for nearly all of the instances, PSO finds a solution in a shorter time. The average difference is about 30% for the pure versions, and about 15% for the hybrid versions.

Average percent deviation CPU time results for different problem sizes:

Table VI-18 Pure Versions, Average Percent Deviation for CPU-times

lambda	0.00					0.25					0.50					0.75					1.00				
maxgen	**1000**	**1500**	**2000**	**2500**	**3000**	**1000**	**1500**	**2000**	**2500**	**3000**	**1000**	**1500**	**2000**	**2500**	**3000**	**1000**	**1500**	**2000**	**2500**	**3000**	**1000**	**1500**	**2000**	**2500**	**3000**
20x15	38.09	38.95	39.41	38.73	38.31	58.83	35.64	36.97	36.76	35.88	40.53	40.39	38.35	40.85	40.28	35.64	36.53	35.46	36.77	35.56	36.34	36.41	36.38	36.27	38.65
20x20	37.20	37.82	37.72	36.91	37.08	50.41	34.78	34.06	34.93	35.20	40.40	39.37	37.55	37.69	39.29	35.42	35.54	33.98	34.22	34.65	36.18	35.39	35.07	34.95	37.26
30x15	30.06	31.05	31.42	31.17	31.27	41.63	27.88	28.22	28.59	28.47	32.67	34.08	31.67	33.00	33.23	27.54	28.85	27.63	27.96	27.78	28.25	27.66	27.93	28.03	30.52
30x20	30.20	30.77	30.90	30.78	30.82	40.03	28.34	28.80	29.10	29.02	34.48	32.54	32.53	32.69	32.90	28.37	29.68	28.50	28.85	28.62	29.11	28.58	28.92	28.98	30.92
40x15	24.52	24.48	24.33	24.18	24.16	24.92	22.23	21.61	21.82	21.85	26.45	26.84	25.62	25.66	25.50	22.76	22.40	21.41	21.37	21.27	23.04	22.21	21.97	22.31	23.50
40x20	26.90	26.08	25.80	25.56	25.62	26.50	24.83	23.98	24.01	23.84	28.02	28.31	27.26	27.14	27.02	25.78	25.11	24.31	24.08	23.93	25.52	24.42	24.15	24.46	25.37
50x15	20.75	19.72	18.74	18.63	18.50	20.80	18.24	16.90	16.45	16.29	23.89	23.11	21.41	21.50	20.82	18.45	16.66	15.65	15.44	15.32	18.41	17.12	16.27	16.27	16.54
50x20	22.92	21.70	20.55	20.22	20.10	23.07	20.71	19.62	19.26	18.94	24.78	23.81	22.56	22.34	22.08	22.10	20.61	19.67	19.28	19.05	22.45	20.89	19.96	19.90	20.14

For the hybrid (with VNS) versions of the algorithms:

Table VI-19 Hybrid Versions, Overall CPU-Time Results

lambda	0.00		0.25		0.50		0.75		1.00	
maxgen	1000	3000	1000	3000	1000	3000	1000	3000	1000	3000
PSO better	159	160	157	160	160	160	158	160	158	160
GA better	1	0	3	0	0	0	2	0	2	0
PSO=GA	0	0	0	0	0	0	0	0	0	0
Best PSO (%)	22.35	22.90	33.24	31.25	23.21	24.22	21.16	21.24	20.39	22.50
Best GA (%)	27.18	NA	2.99	NA	NA	NA	39.21	NA	39.34	NA
Average (%)	-14.67	-17.54	-13.52	-18.12	-17.48	-16.55	-13.72	-15.99	-13.25	-16.29

Average percent deviation CPU time results for different problem sizes:

Table VI-20 Hybrid Versions, Average Percent Deviation for CPU-times

lambda	0.00		0.25		0.50		0.75		1.00	
maxgen	1000	3000	1000	3000	1000	3000	1000	3000	1000	3000
20x15	-13.12	-19.96	-11.75	-26.42	-19.99	-10.49	-4.59	-18.01	-4.12	-17.50
20X20	-13.12	-18.50	-7.21	-20.17	-18.72	-18.58	-8.16	-17.25	-8.08	-16.70
30x15	-15.28	-18.05	-11.63	-16.65	-18.15	-18.00	-16.13	-16.79	-15.76	-16.38
30x20	-16.12	-16.82	-14.92	-15.40	-16.27	-16.28	-15.25	-15.39	-15.11	-15.23
40x15	-17.27	-17.57	-16.11	-17.46	-17.64	-18.14	-16.41	-15.53	-15.67	-16.28
40x20	-17.04	-15.89	-16.83	-15.42	-16.71	-16.25	-16.31	-14.49	-16.12	-15.30
50x15	-16.09	-17.54	-15.59	-17.81	-16.91	-18.64	-15.32	-16.08	-13.90	-16.77
50x20	-15.46	-15.98	-14.27	-15.61	-15.71	-15.99	-14.50	-14.35	-14.02	-16.19

VII. RESULTS AND CONCLUSION

In this thesis, I compared the performances of GA and PSO algorithms, on the Permutation Flowshop Scheduling Problem. I tested the performance of the algorithms for minimizing multi objectives, namely makespan and maximum tardiness, concurrently.

The objective function I used was: Min $f = \lambda C_{max} + (1- \lambda)T_{max}$

Using five different lambda values, I found out the relative performances of the two algorithms for different combinations of makespan and tardiness, including the single objective cases; that is, makespan only (for $\lambda=1$), and maximum tardiness only (for $\lambda=0$). Also, I performed computer runs by using different generation numbers to see how the algorithms behave as the number of generations increases.

In addition to the standard versions of the algorithms, I tested the hybrid versions of them too. GA and PSO was hybridized by an efficient local search, namely VNS. The relative performance of the hybrid algorithms were tested and compared as well.

For the pure versions of the algorithms, it is observed that the solution quality of PSO is generally superior to GA. But, the difference decreases as lambda increases. For lambda = 0.00, 0.25 and 0.50, PSO performs better; for lambda=0.75 GA takes over, and finally, for lambda=1.00 (e.g. when only makespan is considered) GA certainly outperforms PSO. But the average difference between the algorithms is less than 3%, for all different occasions.

So, we can conclude that PSO algorithm is more suitable for the due-date based objectives (e.g. maximum tardiness).

For the, hybrid (e.g with VNS) versions of the algorithms, PSO outperforms GA for all lambda values. For all lambda values, either PSO performs strictly better, or there appears to be no significant difference between the performance of the two algorithms.

But, the difference between the performances of the algorithms is much lower, compared to the pure versions. The average difference between the two algorithms is less than 0.5% for all different combinations.

So, we can say that VNS local search works very efficiently and similar results are obtained for both of the algorithms.

When we consider the CPU times, for both pure and hybrid versions, PSO obviously outperforms GA. For the pure versions, PSO runs last about 30% shorter than the GA runs, and for the hybrid versions, the average difference is about 15%.

But of course, compared to pure versions, the with-VNS versions run much longer to reach a solution, because of the local search. But, the solution quality of the hybrid versions is better.

There are some newly developed “discrete PSO” algorithms that were nor present when I started my thesis. As I mentioned throughout the thesis, I made the continuous PSO algorithm discrete, by applying a special rule called SPV.

As a furtherwork, these new discrete PSO algotihms can be compared with the “old” version that I used. Also a GA, and maybe some other heuristics can be added, run on the same benchmark problems and compared.

Also, VNS or another local search can be run on the same problem sets seperately, and the results may be compared with the results of both versions of PSO and GA, pure and hybrid.

REFERENCES

Aldowaisan, T.; Allahverdi, A.: "New heuristics for no-wait flowshops to minimize makespan", *Computers and Operations Research* 30(8) **(2003)** 1219-1231.

Allahverdi, A.: "A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness", *Computers & Operations Research*, 31 **(2004)** 157–180.

Angeline P.J.: "Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences", In *Proceedings of the 7th International Conference on Evolutionary Programming* **(1998)** 601-610.

Armentano, V.A.; Ronconi, D.P.: "Tabu search for total tardiness minimization in flowshop scheduling problems", *Computers and Operations Research*, 26 **(1999)** 219–235.

Arroyo, J.E.C.; Armentano, V.A.: "Genetic local search for multi-objective flowshop scheduling problems", *European Journal of Operational Research*, 167 **(2005)** 717-738.

Baker, K.R.: "Introduction to Sequencing and Scheduling", John Wiley & Sons, New York, **(1974)**

Beasley, D.; Bull D.R.; Martin, R.R.: "An Overview of Genetic Algorithms, Fundamentals", University Computing, Vol. 15(2) **(1993)** 58 -69.

Brandstatter, B.; Baumgartner, U.: "Particle swarm optimization mass-spring system analogon", *IEEE Transactions on Magnetics*, vol.38 **(2002)** 997-1000.

Campbell H.; Dudek R.; Smith M.: "A heuristic algorithm for the n job, m machine sequencing problem", *Management Science*, 16(10) **(1970)** B630-B637.

Carlisle, A; Dozier, G.: "Adopting particle swarm optimization to dynamic environments", In *Proceedings of the International Conference on Artificial Intelligence* **(2000)** 429-434.

Chakravarthy, K.; Rajendran, C.: "A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization", *Production Planning and Control*, 10 **(1999)** 707–714.

Chan, F.T.S.; Chung, S.H., "Multicriterion genetic optimization for due date assigned distribution network problems", *Decision Support Systems*, **(2004)**

Chang, C.K.; Christensen, M.J.; Zhang T.: "Genetic Algorithms for Project Management", *Annals of Software Engineering*, 11, 1 **(2001)** 107-139.

Chatterjee, A.; Siarry, P.: "Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization", *Computers & Operations Research*, 33 **(2006)** 859-871.

Chen, C. L.; Vempati, V.S.; Aljaber, N.: "An application of Genetic algorithms for flow shop problems", *European Journal of Operational Research* 80(2) **(1995)** 389-396.

Chu, C.; Proth, J.M.; Sethi, S.: "Heuristic procedures for minimizing makespan and the number of required pallets", *European Journal of Operational Research*, 86 **(1995)** 491–502.

Clerc, M.: “The swarm and the queen: towards a deterministic and adaptive particle swarm optimization”, In *Proceedings of ICEC*, Washington, D.C. **(1999)** 1951-1957.

Coit, D.W.; Smith, A.E.:, “Penalty guided genetic search for reliability design optimization”, *Computers and Industrial Engineering*, Vol. 30, No. 4 **(1996)** 895-904.

Daniels, R.L.; Chambers, R.J.: “Multiobjective flow-shop scheduling”, *Naval Research Logistics*, 37 **(1990)** 981–995.

Dannenbring, D.: “An evaluation of flow shop sequencing heuristics”, *Management Science*, 23(11) **(1977)** 1174-1182.

Demirkol, E.; Mehta, S.; Uzsoy, R.: “Benchmarks for shop scheduling problems”, *European Journal of Operational Research*, 109 **(1998)** 137-141.

Eberhart, R.C.; Hu, X.: “Human Tremor Analysis Using Particle Swarm Optimization”, In *Proceedings of the Congress on Evolutionary Computation*, Washington, D.C. **(1999)** 1927-1930, Piscataway, NJ: IEEE Service Center,

Eberhart, R.C.; Kennedy, J., “A new optimizer using particle swarm theory”, In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Nagoya, Japan **(1995)** 39-43.

Eberhart, R.C.; Shi, Y.: “Comparison between genetic algorithms and particle swarm optimization”. In Evolutionary Programming VII, Porto, V.W., Saravanan, N., Waagen, D. and Eiben, A.E. (eds) **(1998)** 611–616 Springer

Eberhart, R.C.; Shi, Y.: “Tracking and Optimizing Dynamic Systems with Particle Swarms”, In *Proceedings of the Congress on Evolutionary Computation*, Seoul, Korea **(2001)** Piscataway, NJ: IEEE Service Center,

Eberhart, R.C.; Simpson, P.K.; Dobbins, R.W.: “Computational Intelligent PC Tools”, Boston, MA. **(1996)** Academic Press Professional.

Framinan, J.M.; Leisten R.: “A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness”, *International Journal of Production Economics*, 99 **(2006)** 28–40.

Framinan, J.M.; Leisten, R.: “An efficient constructive heuristic for flowtime minimisation in permutation flow shops”, *Omega*, 31 **(2003)** 311-317.

Framinan, J.M.; Leisten, R.; Ruiz-Usano, R.: “Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation”, *European Journal of Operational Research*, 141 **(2002)** 559-569.

Garcia, C.G.; Pérez-Brito, D.; Campos, V.; Martí, R.: “Variable neighborhood search for the linear ordering problem”, *Computers & Operations Research*, 33 **(2006)** 3549–3565.

Garey, M.; Johnson, D.; Sethi, R.: “The complexity of flowshop and jobshop scheduling”, *Mathematics of Operations Research*, 1 2, **(1976)**, 117–129.

Ghoshal, S.P.; Roy, N.K.: “A novel approach for optimization of proportional integral derivative gains in automatic generation control”, *Australasian Universities Power Engineering (Conference (AUPEC)*, **(2004)**.

Hino, C.M.; Ronconi, D.P.; Mendes, A.B.: “Minimizing earliness and tardiness penalties in a single-machine problem with a common due date”, *European Journal of Operational Research*, Vol. 160 **(2005)** 190–201.

Holland, J.: “Adaptation in natural and artificial systems”, Ann Arbor, MI. **(1975)** The University of Michigan Press.

Hoogeveen, H.: “Multicriteria scheduling”, *European Journal of Operational Research*, 167 (2005) 592–623.

Johnson, S.: “Optimal two- and three-stage production schedules with setup times included”, *Naval Research Logistics Quarterly*, 1 **(1954)** p61.

Kennedy, J.; Eberhart, R.C.: “A Discrete Binary Version of the Particle Swarm Optimization”, :In *Proceedings of the Conference on Systems, Man, and Cybernetics* SMC97 **(1997)** 4104-4109.

Kennedy, J.; Eberhart, R.C.: “Particle swarm optimization”, In *Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ, USA **(1995)** 1942-1948.

Kennedy, J.; Eberhart, R.C.; Shi, Y.: “Swarm intelligence”, Morgan Kaufmann, San Mateo, CA. **(2001)**

Kennedy, J.; Spears W.M.: “Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator”, In *Proceedings of the International Conference on Evolutionary Computation* **(1998)** 78-83.

Kim, K.W.; Gen, M.; Yamazaki, G.: “Hybrid genetic algorithm with fuzzy logic for resource-constrained project scheduling”, Applied Soft Computing, **(2003)**

Kim, Y.D.: “Heuristics for Flowshop Scheduling Problems Minimizing Mean Tardiness”, *Journal of the Operational Research Society*, Vol. 44, No.1 **(1993)** 19-28.

Kim, Y.D.: “Minimizing total tardiness in permutation flowshops”, *European Journal of Operational Research*, 85 **(1995)** 541–555.

Kirkpatrick, S.; Gellat C.D.; Vecchi M.P.: “Optimization by Simulated Annealing”, *Science,* 220 **(1983)** 671-680.

Koulamas, C.: “A new constructive heuristic for the flowshop scheduling problem”, *European Journal of Operational Research*, 105 **(1998)** 66-71.

Köksalan, M.; Keha, A.B.: “Using genetic algorithms for single-machine bicriteria scheduling problems”, *European Journal of Operational Research*, 145 **(2003)** 543-556.

Leu, S.; Hwang, S.: “GA-based resource-constrained flow-shop scheduling model for mixed precast production”, *Automation in Construction*, Vol. 11 **(2002)** 439– 452.

Liao, C-J.; Tseng C-T.; Luarn P.: “A discrete version of Particle Swarm Optimization for Flowshop Scheduling Problems”, *Computers & Operations Research*, In Press, **(2006)**.

Løvbjerg, M.; Rasmussen, T.; Krink, T.: “Hybrid particle swarm optimiser with breeding and subpopulations”, In *Proceedings of the Third Genetic and Evolutionary Computation Conference* (Gecco) vol 1 **(2001)** 469-476.

Mladenovic, N.; Hansen, P.: “Variable neighborhood search”, *Computers and Operations Research*, 24 **(1997)** 1097-1100.

Murata, T.; Ishibuchi, H.; Tanaka, H.: “Genetic algorithms for flowshop scheduling problems”, *Computers and Industrial Engineering*, 30(4) **(1996)** 1601-1071.

Nagar, A.; Haddock, J.; Heragu, S.: “Multiple and Bicriteria Scheduling: A Literature Survey”, *European Journal of Operational Research*, 81 **(1995)** 88-104.

Nawaz, M.; Enscore Jr. E.E.; Ham, I.: "A heuristic algorithm for the m-machine, n-job flow shop sequencing problem", *OMEGA* 11(1) **(1983)** 91-95.

Nearchou, A.C.: "A novel metaheuristic approach for the flowshop scheduling problem", *Engineering Applications of Artificial Intelligence*, Vol. 17 **(2004)** 289-300.

Nowicki, E.; Smutnicki, C.: "A fast tabu search algorithm for the permutation flowshop problem", *European Journal of Operational Research*, 91 **(1996)** 160-175.

Ogbu, F.; Smith, D.: "The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem", *Computers & Operations Research*, 17(3) **(1990)** 243-253.

Osman, I.; Potts, C.: "Simulated annealing for permutation flow shop scheduling", *OMEGA* 17(6) **(1989)** 551-557.

Parsopoulos, K.E.; Vrahatis, M.N.: "Recent approaches to global optimization problems through Particle Swarm Optimization", *Natural Computing* 1 **(2002)** 235–306, Kluwer Academic Publishers.

Pavlidis, N.G.; Parsopoulos, K.E., Vrahatis M.N.: "Computing Nash equilibria through computational intelligence methods", *Journal of Computational and Applied Mathematics*, 175 **(2005)** 113-136.

Pinedo, M.: "Scheduling: theory, algorithms, and systems", Prentice-Hall, Englewood Cliffs, NJ. **(1995)**

Rajendran, C.; Ziegler, H.: "Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs", *European Journal of Operational Research*, 155(2) **(2004)** 426-438.

Rechenberg, I.: "Evolution Strategy", In "Computational Intelligence: Imitating Life", Zurada, J.M., Marks, R.J.II and Robinson, C. (eds), **(1994)** IEEE Press, Piscataway, NJ

Reeves, C.: "A genetic algorithm for flowshop sequencing", *Computers & Operations Research*, 22(1) **(1995)** 5-13.

Reeves, C.: "Improving the efficiency of tabu search for machine sequencing problem", *Journal of the Operational Research Society*, 44(4) **(1993)** 375-382.

Reeves, C.; Yamada, T.: "Genetic algorithms, path relinking and the flowshop sequencing problem", *Evolutionary Computation,* 6 **(1998)** 45-60.

Riane, F.; Artiba, A.; Elmaghraby, S.E.: "A hybrid three-stage flowshop problem: efficient heuristics to minimize makespan", *European Journal of Operational Research*, 109 **(1998)** 321–329.

Ruiz, R.; Maroto, C.: "A comprehensive review and evaluation of permutation flowshop heuristics", *European Journal of Operational Research*, 165 (2005) 479-494.

Ruiz, R.; Maroto, C.; Alcaraz, J.: "Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics", European Journal of Operational Research, In Press, Corrected Proof, **(2004)**

Ruiz, R.; Maroto, C.; Alcaraz, J.: "Two new robust genetic algorithms for the flowshop scheduling problems", *OMEGA* 34 **(2006)** 461-476.

Ruiz, R.; Stützle, T.: "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem", *European Journal of Operational Research*, In Press, Corrected Proof, **(2006)**.

Schwefel, H-P.: "Evolution and Optimum Seeking" **(1995)** Wiley, New York.

Şerifoğlu, F.; Ulusoy, G.: "A bicriteria two-machine permutation flowshop problem", *European Journal of Operational Research*, 107 **(1998)** 414-430.

Şevkli, M.: "Comparison of the Particle Swarm Optimization Algorithm and Genetic Algorithm for the Jobshop Scheduling Problem", PhD Dissertation (in Turkish), (**2005**), Dep't of Industrial Engineering, Istanbul Technical University.

Shi, Y.; Eberhart, R.C.: "Parameter selection in particle swarm optimization", Evolutionary Programming VII, *Lecture Notes in Computer Science*, vol. 1447 Springer **(1998a)** 591–600.

Shi, Y.; Eberhart, R.C.: "A modified particle swarm optimizer", In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation,* **(1998b)** 69-73.

Srinivasaraghavan, P.; Rajendran, C.: "Scheduling to minimize mean tardiness and weighted mean tardiness in flowshop and flowline-based manufacturing cell", *Computers and Industrial Engineering*, 34 **(1998)** 531–546.

Starkweather, T.; McDaniel, S.; Mathias K.; Whitley, D.; Whitley, C.: "A Comparative Study of Genetic Sequencing Operators", In *Proceedings of the 4th International Conference on Genetic Algorithms*, R. Belew and L. Booker, eds., **(1991)** 69-76. Morgan Kaufmann.

Stutzle, T.: "An ant approach to the flowshop problem", in *Proceedings of EUFIT'98*, **(1998)**.

Stützle, T.: "Applying iterated local search to the permutation flowshop problem", Technical Report, AIDA-98-04, Darmstad University of Technology, Computer Science Department, Intellctics Group **(1998)** Darmstad, Germany.

T'kindt, V.; Billaut, J.C.: "Multicriteria Scheduling: Theory, Models and Algorithms", **(2002)** Springer, Berlin.

T'kindt, V.; Monmarche, N.; Tercinet F.; Laugt, D.: "An Ant Colony Optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem", *European Journal of Operational Research*, 142 **(2002)** 250-257.

Taillard, E.: "Benchmarks for basic scheduling problems", *European Journal of Operational Research*, 64 **(1993)** 278-285.

Taillard, E.: "Some efficient heuristic methods for the flowshop sequencing problems", *European Journal of Operational Research*, 47 **(1990)** 65-74.

Tandon, V.; El-Mounayri, H.; Kishawy, H.: "NC end milling optimization using evolutionary computation", *Machine Tools & Manufacture*, **(2002).**

Taşgetiren M.F.; Şevkli, M.; Liang, Y-C; Gencyilmaz, G.: "Particle swarm optimization algorithm for single-machine total weighted tardiness problem", In *Proceedings of Congress on Evolutionary Computation, CEC2004* **(2004)** Portland, Oregon, USA.

Taşgetiren, M.F.; Liang, Y.C.: "A Binary Particle Swarm Optimization Algorithm for Lot Sizing Problem", *Journal of Economic and Social Research*, Vol.5 No.2. **(2003)**

Townsend, W.: "Sequencing n jobs on m machines to minimize maximum tardiness: a branch-and-bound solution", *Management Science*, 23 **(1977)** 1016–1019.

Trelea, I. C.: "The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection", *Information Processing Letters*, Vol. 85 **(2003)** 317-325.

Uçar, H.: "Comparison of genetic algorithm and particle swarm optimization algorithm for permutation flowshop sequencing problem with criterion of number of tardy jobs", MS thesis, **(2005)**, Dep't of Industrial Engineering, Fatih University, Turkey.

Uysal, H.: "A Variable Neighborhood Search Algorithm For Identical Parallel Machine Problem", MS thesis, **(2006),** Dep't of Industrial Engineering, Fatih University, Turkey.

Van den Bergh, F.; Engelbecht, A.P.: "Cooperative learning in neural networks using particle swarm optimizers", *South African Computer Journal*, vol.26 **(2000)** 84-90.

Wang, Y.: "A GA-based methodology to determine an optimal curriculum for schools", *Expert Systems with Applications*, Vol. 28 **(2005)** 163-174.

Whitley, D.: "Genetic Algorithms and Evolutionary Computing", Van Nostrand's Scientific Encyclopedia, **(2002)**.

Whitley, D.: "Genetic Algorithms in Engineering and Computer Science", Winter, Periaux, Galan and Cuesta, eds. **(1995)** 203-216, John Wiley.

Widmer, M.; Hertz, A.: "A new heuristic method for the flow shop sequencing problem", *European Journal of Operational Research*, 41(2) **(1989)** 186-193.

Yeh, L.W.: "Optimal procurement policies for multi-product multi-supplier with capacity constraint and price discount", Master thesis, **(2003)**, Department of Industrial Engineering and Management, Yuan Ze University.

Yoshida, H.; Kawata, K.; Fukuyama, Y.; Nakanishi, Y.: "A particle swarm optimization for reactive power and voltage control considering voltage security assessment", *IEEE Transactions on Power Systems*, vol.15 **(2000)** 1232-1239.

Zacharia, P.T.; Aspragathos, N.A.: "Optimal robot task scheduling based on genetic algorithms", *Robotics and Computer-Integrated Manufacturing*, Vol. 21, **(2005)** 67–79.

Zegordi, S.H.; Itoh, K.; Enkawa, T.: "Minimizing makespan for flowshop scheduling by combining simulated annealing with sequencing knowledge", *European Journal of Operational Research*, 85 **(1995)** 515–531.

CV

Özgür UYSAL

E-mail: ouysal@fatih.edu.tr

Education

- **PhD**, Marmara University, Dep't of Industrial Engineering – 2006, İstanbul
- **MS**, Fatih University, Dep't of Industrial Engineering -2000, İstanbul
- **BS**, Boğaziçi University, Dep't of Industrial Engineering -1994, İstanbul

Experience

- Lecturer, Fatih University, since Feb. 2004
- Director of Computer Center, Fatih University, Jun. 2000 - Feb. 2004
- Research Assistant, Fatih University, Sep.1998 - Jun. 2000

Research Interests

- Production Planning, Scheduling, Heuristic Optimization, Computer Integrated Manufacturing

Teaching Interests

- Manufacturing Processes, Materials Science, Technical Drawing, Computer Integrated Manufacturing Systems, Database Management Systems, Management Information Systems

Proceedings

- Ali Türkyılmaz, Özgür Uysal, Metin Şatır, **"Üniversite Personeli İçin Bir Performans Değerlendirme Modeli: AHP Metodunun Kullanımı "**, Yöneylem Araştırması/Endüstri Mühendisliği XXV. Ulusal Kongresi (YAEM2005), Istanbul, Jul. 2005,
- Ö.Uysal, M.Şevkli, **"KOBİ'lere Yönelik KKP(ERP) Uygulamalarında Gözönünde Bulundurulması Gereken Noktalar"**, 1. KOBİLER ve VERİMLİLİK KONGRESİ, İstanbul, Dec. 2004,
- Ö.Uysal, M.Şevkli, A.Türkyılmaz, M.S.Aksoy, **"A New Method For Noise Reduction Based On Mode Filter Algorithm"**, TAINN 2003 (International XII. Turkish Symposium on Artificial Intelligence and Neural Networks), Çanakkale/Türkiye, Jul. 2003, Proceedings of Intl. XII. Turkish Symposium on Artificial Intelligence and Neural Networks; International Journal of Computational Intelligence, ISSN 1304-2386, 1, 1,pp. 327-329
- Ö.Uysal, A.Türkyılmaz, **"New Methods for Noise Removal in Digital Image Processing"**, EURO/INFORMS Istanbul 2003 Joint International Meeting, İSTANBUL/TÜRKİYE, Jul. 2003, Proceedings,

Courses Taught

- **"Materials Science"**, IE 225, Credit:3
- **"Manufacturing Processes"**, IE 226, Credit:3
- **"Data Processing for Social Sciences"**, CENG 152, Credit:3
- **"Basic Computer Skills"**, CENG 111, Credit:3