A PARALLEL IMPLEMENTATION OF A MULTI-OBJECTIVE EVOLUTIONARY ALGORITHM

Christos Kannas

A Thesis

Submitted in Partial Fulfilment of the

Requirements for the Degree of

Master of Science

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

February, 2010

ABSTRACT

The use of Evolutionary Algorithms (EAs) in difficult problems, where the search space is unknown, urges researches to find ways to exploit their parallel aspect. Multi-objective Evolutionary Algorithms (MOEAs) have features that can be exploited to harness the processing power offered by modern multi-core CPUs. Modern programming languages offer the ability to use threads and processes in order to achieve parallelism that is inherent in multi-core CPUs. This thesis presents a parallel implementation of a MOEA algorithm and its application to the de novo drug design problem. Drug discovery and De novo Drug design is a complex task that has to satisfy a number of conflicting objectives, where a MOEA finds a suitable problem to be used on. Further more such a task needs high amount of execution time. The aim is to minimize this time by the use of a parallel MOEA. The results indicate that using multiple processes that execute independent tasks of a MOEA can reduce significantly the execution time required and maintain comparable solution quality thereby achieving improved performance.

APPROVAL PAGE

Master of Science Thesis

A PARALLEL IMPLEMENTATION OF A MULTI-OBJECTIVE

EVOLUTIONARY ALGORITHM

Presented by

Christos Kannas

Research Supervisor

_____
Research Supervisor's Name

Committe Member

_____
Committe Member's Name

Committe Member

_____
Committe Member's Name

University of Cyprus

February, 2010

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Parallel and distributed computing are key technologies in the present days. During the last few years we have been experiencing a technological paradigm shift in the field of processor design, exemplified by multi- and many-core Central Processing Units (CPUs) and the introduction of General Programming (GP) on Graphical Processing Units (GPUs).

The majority of software programs are written for serial computation. This kind of computation does not take advantage of the multi-/many-core technology available in current CPUs. Parallel computing, on the other hand, is a form of computation in which many calculations are carried out in parallel, simultaneously. Following the paradigm of parallel computing, software programs can take advantage of current and future CPU architecture to speed-up the execution of an algorithm. The speedup of an algorithm as a result of parallelization is a topic that has attracted considerable research. An overview of Amdahl's Law, which governs algorithm speedup, is given in section 2.7.

Writing a parallel software program is more difficult than writing a sequential one, because concurrency introduces several new categories of software issues. Among them, communication and synchronization of the execution of different concurrent tasks are some of the biggest obstacles to getting good parallel program performance.

In general, parallel programs are very dependent on the specific hardware architecture for which they were developed. General parallel programming turns out to be extremely difficult to implement due to the existence of different hardware architectures and to the limitations of current programming paradigms and languages. Despite this, many important problems are sufficiently regular in their space and time dimensions as to be suitable for parallel or

distributed computing and evolutionary algorithms, with which the present research is dealing, are certainly among those [1].

Evolutionary Algorithms (EAs) are stochastic search and optimization techniques which were inspired by natural evolution and population genetics. Evolutionary Algorithms are an interdisciplinary research field with relationships to biology, artificial intelligence, numerical optimization and applications for decision support in almost any engineering discipline [2].

Evolutionary Algorithms are based on models of organic evolution, so they mimic what happens in nature. In its struggle to survive, a population of individuals functioning in a specific environment needs to appropriately adapt; therefore reproduction is promoted by the elimination of useless and harmful traits and by the rewarding of useful behaviour [1]. EAs maintain a population of individuals that evolves, by the use of mutations, crossover and selection, over time and ultimately converges to a commonly unique solution. Each individual represents not only a search point in the space of potential solutions to a given problem, but may also be a temporal container of current knowledge about the "laws" of the environment [2].

Parallel Evolutionary Algorithms (PEAs) emerged for two main reasons: one is the need to achieve time-savings by distributing the computational effort and the second is to benefit from a parallel environment from an algorithmic point of view.

There are many ways for an algorithm to take advantage of a parallel environment. The most trivial one is to use the parallel resources to run independent problems, as there is no communication between the different problems/processes. This does not add something new to the nature of the algorithm but it can provide large savings in time. The most challenging way is to change the algorithm in order to use those parallel resources. This can be achieved by parallelizing the algorithm at several possible levels.

Parallelization of EAs is achieved at three possible levels, the fitness level which does not require any major change to the standard EA since the fitness evaluation of an individual is independent of the other operations of the algorithm, the individual level where every individual evolves independent of the rest of the population, and the population level where

the population is distributed into several subpopulations that evolve independently for periods of time.

The goals of this thesis are to:

- Explore the possibility of using a Parallel Evolutionary Algorithm on a multi-/many-core CPU environment.

- Implement a Parallel Evolutionary Algorithm that exploits multi-/many-core architectures, so as to produce solutions of the same quality as an ordinary Evolutionary Algorithm in much less time.

- Learn more about parallel programming in multi-/many-core environment with the use of high level programming languages, such as Python.

- Gain experiences in Parallel Evolutionary Algorithms, their strengths, weakness and potential applications.

The remainder of this thesis consists of 6 sections. Section 2 reviews related literature and provides some background to the problem. Section 3 describes two algorithms, the one that served as the base algorithm and the one that was implemented for the purposes of the thesis. Section 4 describes in detail the implementation of the algorithm, while section 5 presents the experimental results obtained. Section 6 provides a discussion on the results obtained through the experiments and finally section 7 summarizes the conclusions and outlines possible future work.

Chapter 2

Literature Review

In mathematics and computer science, optimization refers to choosing the best element from some set of available alternatives. In the simplest case, this means solving problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables from within an allowed set. This formulation, using a scalar, real-valued objective function, is probably the simplest example; the generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics. More generally, it means finding "best available" values of some objective function given a defined domain, including a variety of different types of objective functions and different types of domains [3].

In order to find the optimal solution(s) for a problem, optimization techniques are used. Typically, optimization techniques are classified based on their characteristics, such as the methodology used to generate solutions, the method used for evaluating the quality of the solutions, the strategy used for exploring the search space, the number of objective functions used, etc. [3].

The optimal solution to an optimization problem is the best possible solution that can be found given the problem constraints, by finding the minimum (minimization problem) or maximum (maximization problem) solution of an objective function.

According to the number of objective functions considered, optimization techniques can also be classified into two large categories, Single Objective OPtimization (SOOP) methods and Multiple Objective OPtimization (MOOP).

2.1 Optimization

In mathematics and computer science, optimization refers to choosing the best element from some set of available alternatives. In the simplest case, this means solving problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables from within an allowed set. This formulation, using a scalar, real-valued objective function, is probably the simplest example; the generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics. More generally, it means finding "best available" values of some objective function given a defined domain, including a variety of different types of objective functions and different types of domains [3].

In order to find the optimal solution(s) for a problem, optimization techniques are used. Typically, optimization techniques are classified based on their characteristics, such as the methodology used to generate solutions, the method used for evaluating the quality of the solutions, the strategy used for exploring the search space, the number of objective functions used, etc. [3].

The optimal solution to an optimization problem is the best possible solution that can be found given the problem constraints, by finding the minimum (minimization problem) or maximum (maximization problem) solution of an objective function.

According to the number of objective functions considered, optimization techniques can also be classified into two large categories, Single Objective OPtimization (SOOP) methods and Multiple Objective OPtimization (MOOP).

2.2 Single Objective Optimization

Single Objective Optimization techniques are techniques aiming to solve problems where solutions need to satisfy only one objective function. Since these problems try to find the

optimum of a one objective function they typically have only one optimal solution. In these problems Single Objective Optimization techniques are most commonly used.

Single Objective Optimization Problem is defined as:

$$\begin{cases} minimize\ f(\vec{x}) \\ g_j\ (\vec{x}) \geq 0\ (j = 0, ..., m) \\ \vec{x} \in X \subset R^n \end{cases}$$

With an objective function $f(\vec{x})$, which must be minimized, a number of constrains $g_j\ (\vec{x})$, a vector $\vec{x}$ of n decision variables, $\vec{x} = (x_1, x_2, ..., x_n)^T$, and X that represents a feasible region.

SOOP techniques try to satisfy, by minimization or maximization, the single objective related to a problem. However, the majority of the problems in the real world do not have only one objective to satisfy, but a number of objectives. Often, these techniques are used in problems which need to satisfy many objective functions, by mapping those objective functions to one composite objective function.

Such a problem can be defined as:

$$\begin{cases} minimize\ f(\vec{x}) \\ f(\vec{x}) = \sum_i^b w_i k_i(\vec{x}) \\ g_j\ (\vec{x}) \geq 0\ (j = 0, ..., m) \\ \vec{x} \in X \subset R^n \end{cases}$$

Where the composite objective function $f(\vec{x})$ is the weighed sum of all objective functions $k_i(\vec{x})$ and w_i a weight constant for each objective function, with $i = (0, 1, ..., b)$.

2.3 Multi-Objective Optimization

Multi-Objective Optimization techniques are used to solve problems which need to satisfy a number of objective functions which, in most cases, are conflicting between them. As a result there exists a set of equally optimal solutions instead of a single optimal one. This set of optimal solutions is called the Pareto Front. The solutions which form the Pareto Front are also called nondominated solutions, because of the absence of any solution that is better than

them in all objectives. An example of such solutions is shown in Figure 1, where solutions noted with label '0' are considered nondominated solutions, and solutions with labels '1' to '4' are considered dominated solutions. The number of their labels indicates the number of solutions that dominates them.
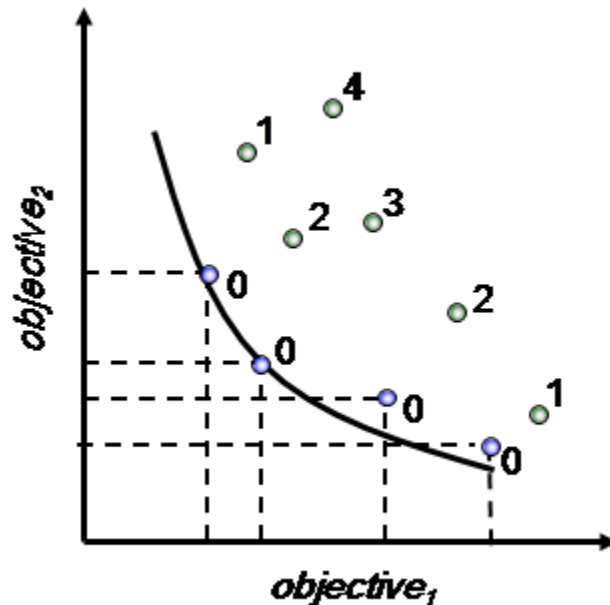
Figure 1: A Multi-Objective Problem with two minimization objectives and a set of solutions (circles). Non-dominated solutions are labeled '0'. The curved line represents the Pareto Front.

2.4 From Darwinian Theory to Evolutionary Algorithms

Evolutionary Algorithms are based on a model of natural, biological evolution, which was formulated for the first time by Charles Darwin [4]. Darwin's "Theory of Evolution" explains the adaptive change of species by the principle of natural selection, which favours those species for survival and further evolution that are best adapted to their environmental conditions. In addition to selection, Darwin recognized the occurrence of small, apparently random and undirected variations between the phenotypes (any observable characteristic or trait of an organism). These mutations prevail through selection, if they prove their worth in the light of the current environment, otherwise they perish. The basic driving force for selection is given by the natural phenomenon of production of offspring. Under advantageous environment conditions, population size grows exponentially, a process which is generally

limited by finite resources. When resources are no longer sufficient to support all the individuals of a population, those organisms are at a selective advantage which exploit resources most effectively [5].

Modern biochemistry and genetics has extended the Darwinian Theory by microscopic findings concerning the mechanisms of heredity, the resulting theory is called the "Synthetic Theory of Evolution" [6]. This theory is based on genes as transfer units of heredity. Genes are occasionally changed by mutations. Selection acts on the individual, which expresses in its phenotype the complex interactions within its genotype, its total genetic information, as well as the interaction of the genotype with the environment in determining the phenotype. The evolving unit is the population which consists of a common gene pool included in the genotypes of the individuals.

In the evolutionary framework, the fitness of the individual is measured only indirectly by its growth rate in comparison to others. Furthermore, natural selection is no active driving force, but differential survival and reproduction within a population makes up selection. Selection is simply a name for the ability of those individuals that have outlasted the struggle for existence to bring their genetic information to the next generation. This point of view however, reflects just our missing knowledge about the mapping from genotype to phenotype, a mapping which, if it were known, would allow us to evaluate fitness in terms of a variety of physical properties [2].

2.5 Drug Discovery and De Novo Drug Design

This section introduces the problems of Drug Discovery and De Novo Drug Design largely in line with [7]. Drug discovery and development is a complex, lengthy process, where failure of a candidate molecule can occur as a result of a combination of reasons, such as poor pharmacokinetics, lack of efficacy, or toxicity. Successful drug candidates necessarily represent a compromise between the numerous, sometimes competing objectives so that the benefits to patients outweigh potential drawbacks and risks. De novo drug design involves

searching an immense space of feasible, druglike molecules to select those with the highest chances of becoming drugs using computational technology [7].

Traditionally, de novo design has focused on designing molecules satisfying a single objective, such as similarity to a known ligand or an interaction score, and ignored the presence of the multiple objectives required for druglike behaviour. The process of drug discovery focuses on identifying molecules that selectively bind and interact with specific biological receptors and cause a certain desired behaviour. The ability to interact is controlled by the molecular structure of the drug and namely by its complementarity to the targeted receptor site. However, not all potent binding molecules are suitable as drugs. In order to be truly effective within a living organism a molecule must satisfy several additional properties. These properties depend on the way a drug behaves "in vivo" (i.e., in the living organism to be treated) and how well it reaches the region of the target without binding nonselectively to other receptors [8]. An additional equally important requirement drugs need to satisfy is synthetic feasibility. The presence of these requirements turn drug discovery into a multiobjective problem in which any candidate solution needs to fulfil multiple objectives concurrently. Computational de novo drug design involves searching an immense space of feasible, druglike molecules to select those with the highest chances of becoming drugs [9]. Modelled after traditional experimental procedures, which typically follow a sequential optimization paradigm, most de novo design research has been ignoring the multiobjective nature of the problem and focused on the optimization of one molecular property at a time [10]. Typically the property serving as the primary objective has been similar to a known ligand or an interaction score with a target receptor.

MOOP methods introduce a new approach for optimization that is founded on compromises and tradeoffs among the various, potentially conflicting objectives. In a multiobjective problem setting multiple equivalent solutions representing different compromises among the objectives are possible. Although in many applications the presence of multiple solutions may be considered a problem, and therefore methods for selecting a priori the single 'best' solution are often employed, in drug discovery the availability of

several diverse solutions that can serve as leads is generally preferred. Based on this requirement a truly multiobjective de novo design system must be able to produce multiple, diverse solutions and enable users to choose a posteriori from a variety of candidates. Considering the combinatorial nature of the problem, the system must also employ a powerful search strategy in order to detect the best possible solutions within a reasonable amount of time. The strategy must be able to handle complex, non uniform search spaces since the presence of multiple conflicting objectives point to the potential presence of multiple solutions at different regions of the space. The system must also be able to take advantage of existing pharmaceutically relevant knowledge to streamline the search process and achieve better performance. Such knowledge may be supplied in the form of implemented objective functions or rules to be used for rejecting low quality candidates. Not only to facilitate human expert understanding of the internal operations but also to avoid information loss and misleading results the system must represent candidate solutions using a molecular graph data structure. Special emphasis must also be placed on system flexibility, to enable easy choice of objectives, and on performance and scalability issues to ensure practical usefulness [7].

De novo drug design methods face the task of effectively exploring a chemical search space estimated to be on the order of 1060-10100 [11]. Such space cannot possibly be fully enumerated, and so powerful search methods need to be applied to detect the best possible solutions in a limited amount of time.

2.6 Evolutionary Algorithms Family

Evolutionary Algorithms apply principles of biological evolution with the aid of computer algorithms to solve problems in various sciences. Evolutionary Algorithms is a big family of algorithms that uses evolution, as described above, as their primary search mechanism. The members of this family are, Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary Strategies (ES) and Evolutionary Programming (EP).

Every algorithm uses a different representation for its individuals. GA represents each individual with a chromosome string, more often a bit string. GP represents each individual

with a tree data structure. ES represents each individual with natural problem dependent representation with the aid of real value vectors. EP represents each individual with a finite state machine.

However, all EAs have a common approach; this involves the main steps of the algorithm shown in Figure 2.
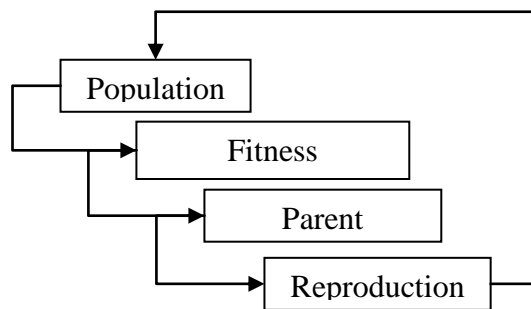
Figure 2: The basic steps of all EAs.

Where EAs differ among them is, as stated above, the representation of their individuals, the way reproduction is achieved, which is directly related to the problem and to the representation of their individuals, and also the evaluation method used, which is derived directly from the objectives of the problem.

The initial population can be selected randomly or use individuals selected by the use of another algorithm or provided by a human expert. Once the initial population is prepared the algorithm enters a loop. At the end of the iteration of the loop, a new population is created which follows the same algorithmic steps.

The first operation applied on the population is fitness evaluation. The next operation that is applied on the population is the selection of a number of parents. This set of parents is selected based on their fitness values. The reproduction operation is applied on the set of parents to create a new set of individuals. In a next step, this new set of individuals has their fitness evaluated. After that the new population is selected. There are two approaches that are used to select the new population. The first approach, often referred to as the elitist approach, selects the new population from the merger of the set of the parents and the set of the evolved

individuals. The second approach selects the new population only from the set of the evolved individuals.

2.7 Parallel Evolutionary Algorithms Family

Parallel Evolutionary Algorithms (PEAs) are a next step in the development of EAs. Due to the nature of EAs, i.e. each individual can be evolved and evaluated independently, researchers have been able to implement parallel EAs by exploiting this feature. In a PEA model the entire population available needs to be in a distributed or shared form. Two major models of PEAs exist, coarse-grained or distributed, and fine grained. In coarse-grained PEAs, there exist multiple independent or interacting subpopulations, while in fine-grained there is only one population and each population member is processed in parallel.
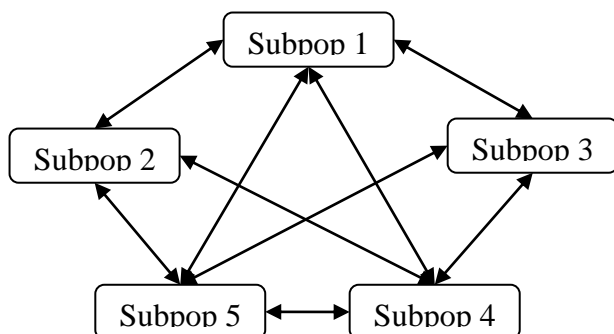
2.7.1 Coarse-grained PEAs

In a coarse-grained PEA, the populations are divided into several subpopulations. These subpopulations evolve independently of each other for a certain number of generations (isolation time). Upon completion of the isolation time a number of the resulting individuals is distributed between the subpopulations, a process often referred to as migration. The number of exchanged individuals (migration rate), the selection method of the individuals for migration and the scheme of migration determines how much genetic diversity can occur in the subpopulation as well as the exchange of information between subpopulations.

The selection of the individuals for migration typically takes place using one of the following two methods:

- Uniformly at random, e.g. pick individuals for migration in a random manner

- Fitness-based, e.g. select the best individuals for migration.

- Several possibilities exist for the migration scheme of individuals among subpopulations. Common migration schemes include:

- Complete, unrestricted net topology, which exchanges individuals among all subpopulations, (see Figure 3),

- Ring topology, where exchange of individuals is allowed only to a specific subpopulation, (see Figure 4),

- Neighborhood topology which exchanges individuals across a "neighborhood" (see Figure 5).

Figure 3: Subpopulations Model for a coarse-grained PEA with unrestricted migration topology [12].
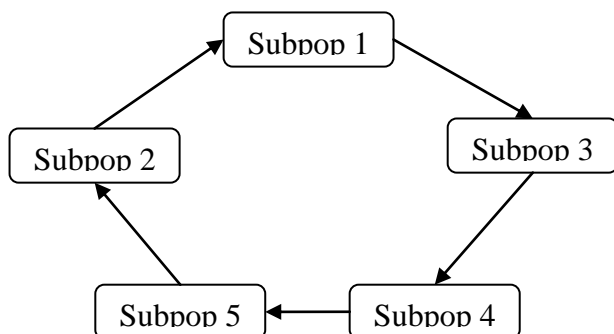

Figure 4: Subpopulations Model for a coarse-grained PEA with ring migration topology [12].
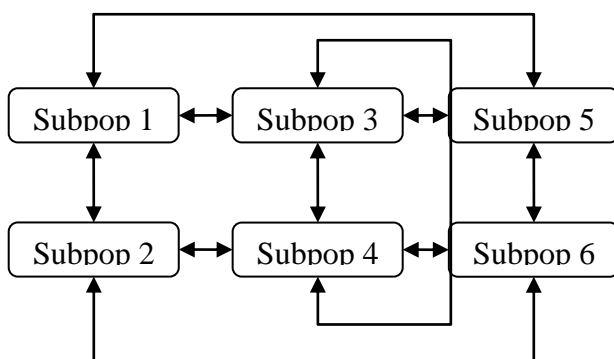
Figure 5: Subpopulations Model for a coarse-grained PEA with neighborhood migration topology [12].

2.7.2   Fine-grained PEAs

In a fine-grained PEA, also known as Global Model or Master/Slave, the population is not divided. Instead, the global model employs the inherent parallelism of evolutionary algorithms i.e. the presence of a population of individuals, and features of the classical evolutionary algorithm. During the calculations where the whole population is needed, Pareto-ranking and selection are performed by the master. All remaining calculations, which are performed for one or two individuals at a time, are distributed to a number of slaves. The slaves perform recombination, mutation and the evaluation of the objective function separately. This is known as synchronous master-slave-structure, (see Figure 6) [12], [1], and [13].
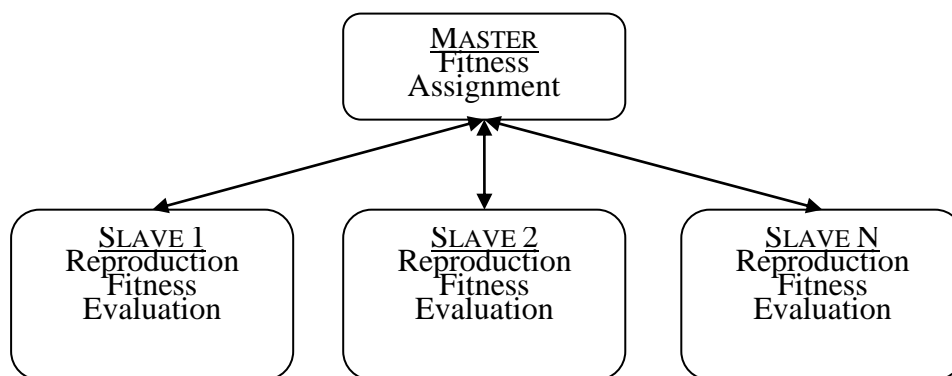
Figure 6: Global Model – Master/Slave for a fine-grained PEA [12].

2.8 What is Speedup and Efficiency?

In parallel computing, speedup [14] refers to how much faster a parallel algorithm is than a corresponding sequential algorithm.
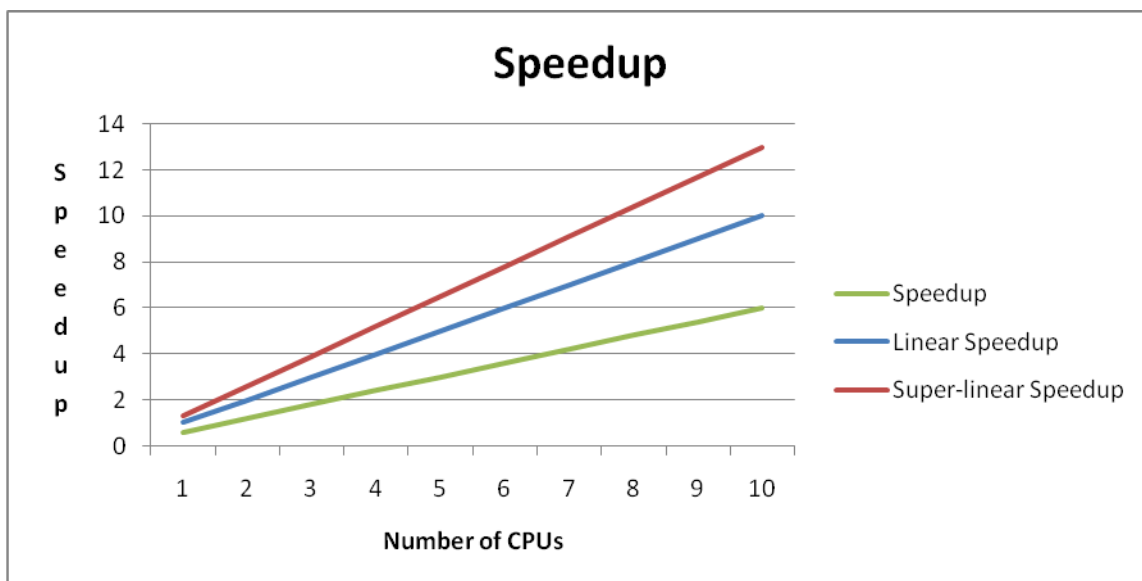
Speedup is defined by the formula [14]:

$$S_p = \frac{T_1}{T_p}$$

Where:

- p is the number of processors used,

- T_1 is the time a sequential algorithm takes to execute a given problem,

- T_p is the time a parallel algorithm takes to execute the same problem using p processors.

Figure 7: Chart that shows basic speedup categories. The reference speedup is the Linear Speedup, which is the ideal speedup. Super linear speedup is the category where the algorithm's speedup is higher than the ideal one. Speedup is the category where the majority of algorithm's speedup exists. The speedup is this category is lower than the ideal one.

Linear speedup or ideal speedup is obtained when $S_p = p$. When running an algorithm with linear speedup, doubling the number of processors doubles its speed of execution. As this is ideal, it is typically considered for speedup evaluation purposes. Linear speedup is impossible to achieve in most algorithms due to the fact that in almost every algorithm there is a part that cannot be run in parallel (see below Amdahl's Law). In some cases algorithms may achieve speedup greater than the linear speedup, known as super-linear speedup however, this occurs due to other factors, such as locating the whole problem data in the CPU's cache [14].

The speedup is called absolute speedup when T_1 is the execution time of the best sequential algorithm, and relative speedup when T_1 is the execution time of the same parallel algorithm on one processor. Relative speedup is usually implied if the type of speedup is not specified, because it doesn't require implementation of the sequential algorithm.
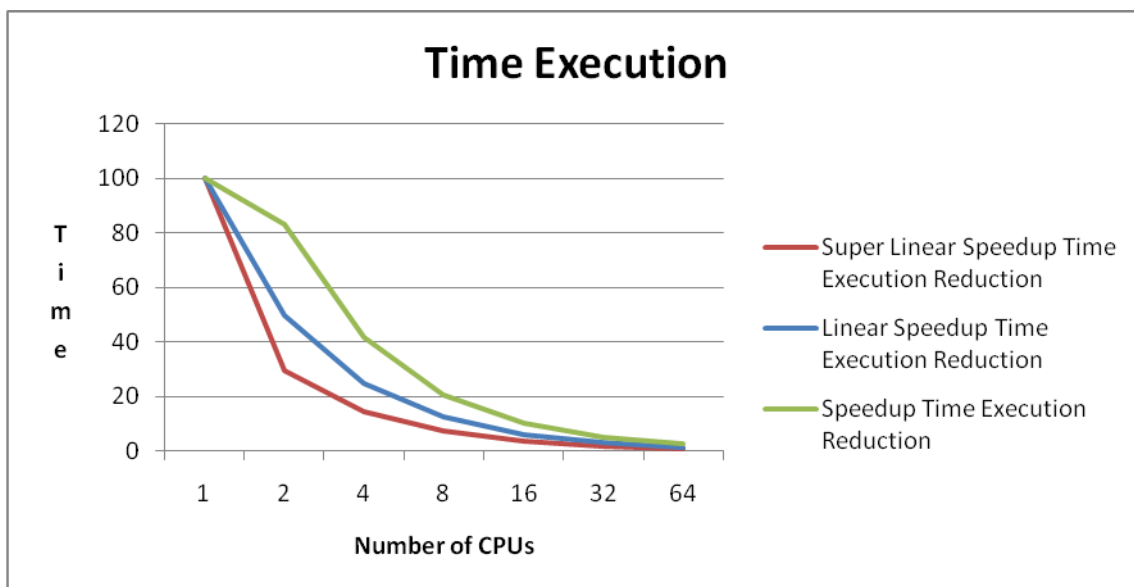
Figure 8: Example of how execution time is reduced, when having (a) Super Linear Speedup, (b) Linear Speedup and (c) a normal Speedup scheme.

Efficiency [14] is a performance metric defined as:

$$E_p = \frac{S_p}{p} = \frac{T_1}{pT_p}$$

Efficiency is a value, typically ranging between zero and one, estimating how well-utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization. Algorithms with linear speedup and algorithms running on a single processor have an efficiency of 1, while many difficult-to-parallelize algorithms have efficiency such as $\frac{1}{\log p}$ that approaches zero as the number of processors increases [14].

2.9 Parallelization and Amdahl's Law

In reality things are not so simple, since in most cases several factors contribute to reduce significantly the theoretical performance improvement expectations. Amdahl's [15] law is used to find the maximum expected improvement to an overall system when only part of the system is improved. It is often used in parallel computing to predict the theoretical maximum speedup using multiple processors.

Amdahl's law is a model for the relationship between the expected speedup of parallelized implementations of an algorithm relative to the serial algorithm, under the assumption that the

problem size remains the same when parallelized. More technically, the law is concerned with the speedup achievable from an improvement to a computation that affects a proportion P of that computation where the improvement has a speedup of S [16]. Amdahl's law states that the overall speedup of applying the improvement will be:

$$\frac{1}{(1-P) + \frac{P}{S}}$$

To see how this formula was derived, assume that the running time of the old, original computation was 1, for some unit of time. The running time of the new computation will be the length of time the unimproved fraction takes, (which is 1 − P), plus the length of time the improved fraction takes. The length of time for the improved part of the computation is the length of the improved part's former running time divided by the speedup, making the length of time of the improved part P/S. The final speedup is computed by dividing the old running time by the new running time, which is what the above formula does.

In the case of parallelization, Amdahl's law states that if P is the proportion of a program that can be made parallel (i.e. benefit from parallelization), and (1 − P) is the proportion that cannot be parallelized (remains serial), then the maximum speedup that can be achieved by using N processors is

$$\frac{1}{(1-P) + \frac{P}{N}}$$

In the limit, as N tends to infinity, the maximum speedup tends to 1 / (1 − P). In practice, performance to price ratio falls rapidly as N is increased once there is even a small component of (1 − P). As an example, if P is 90%, then (1 − P) is 10%, and the problem can be speed up by a maximum of a factor of 10, no matter how large the value of N used. For this reason, parallel computing is only useful for either small numbers of processors, or problems with very high values of P: so-called embarrassingly parallel problems. A great part of the craft of parallel programming consists of attempting to reduce the component (1 − P) to the smallest possible value.

P can be estimated by using the measured speedup S on a specific number of processors N using:

$$P_{estimated} = \frac{\frac{1}{S} - 1}{\frac{1}{N} - 1}$$

$P_{estimated}$ in this way can then be used in Amdahl's law to predict speedup for a different number of processors [16].
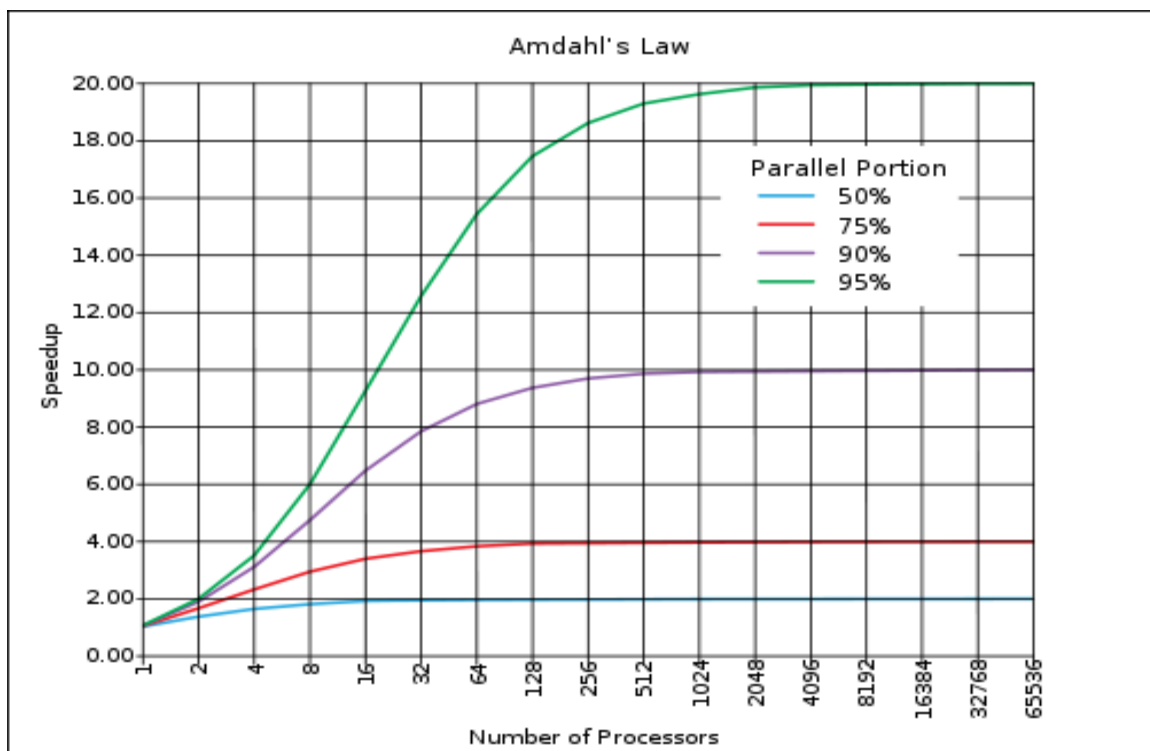
Figure 9: The speedup of a program using multiple processors in parallel computing is limited by the sequential fraction of the program. For example, if 95% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be 20× as shown in the diagram, no matter how many processors are used [16].

Chapter 3

Algorithms

Recently, the Multi-objective Evolutionary Graph Algorithm (MEGA), was proposed, that combines evolutionary techniques with graph data structures to directly manipulate graphs and perform a global search for promising solutions [7]. MEGA has been designed to enable the use of problem-specific knowledge and local search techniques, to improve performance and scalability. Parallel Multi-objective Evolutionary Graph Algorithm (PMEGA) is an extension of MEGA that exploits parallelism in order to reduce execution time. MEGA and PMEGA have been developed primarily for the design of optimal graphs satisfying multiple objectives. Special emphasis has been placed to the problem of designing small molecular graphs with therapeutic properties commonly known as de novo drug design [7]. This section briefly describes the MEGA algorithm and elaborates on the parallel features implemented for PMEGA.

3.1 The MEGA algorithm

The standard version of the MEGA algorithm operates on one population set, referred to as the working population. The population of a single generation consists of the individuals subjected to objective performance calculation and obtained through evolution in a single iteration. Note that solution chromosomes are represented as graphs.

The first phase of the algorithm applies the objectives on the working population to obtain a list of scores for each individual. The list of scores may be used for the elimination of solutions with values outside the range allowed by the corresponding active hard filters. In the next step, the individuals' list of scores is subjected to a Pareto-ranking procedure to set the

rank of each individual. Non-dominated individuals are assigned rank order 1, similar to [17]. The algorithm then proceeds to calculate the efficiency score for each individual, which is used to select a subset of parents via a roulette-like method [18] that favours individuals with high efficiency score, i.e. low domination rank and high chromosome graph diversity [7].

The parents are then subjected to graph-specific mutation and crossover according to the probabilities indicated by the user. The new working population is formed by merging the original working population and the newly produced offspring. The process iterates as shown in Figure 10. The execution of the algorithm completes when the user defined termination conditions, typically a maximum number of iterations, are fulfilled. A more detailed description of MEGA algorithm can be found in [7].
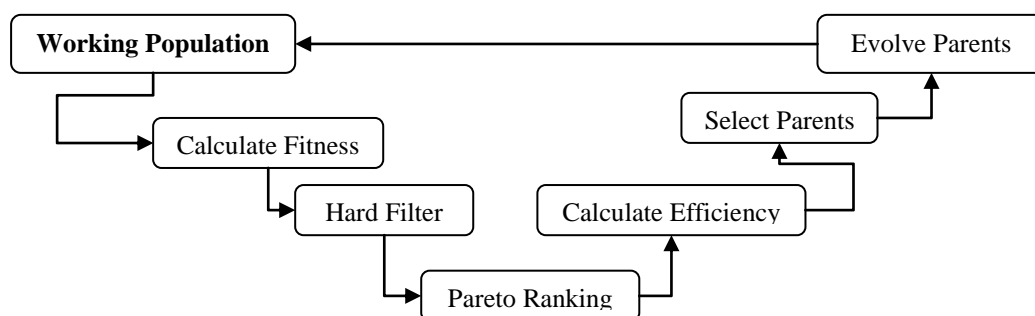
Figure 10: A diagram of the MEGA Algorithm.

3.2 The PMEGA Algorithm

This section describes the PMEGA algorithm. As noted above PMEGA is the parallel version of MEGA. At section 2.4 a brief description of the available parallelization methods was explained. From those methods one was selected to parallelize MEGA. The decision was taken after a thorough investigation of pros and cons of each method and the limitations that the implementation platform has, which are explained in section 4.

The aims for the parallel version of MEGA are:

    a)   Produce solutions of at least equal quality solutions to MEGA.

    b)   Exploit multi-core CPUs.

c) Reduce execution time to obtain results in less time.

d) Explore algorithmic enhancements to produce better quality solutions than MEGA in the future.

Points (a) to (c) can be achieved using any parallelization method, though point (d) needs to be a topic of further research; a promising approach may be through algorithmic enhancements such as the manipulation of the population using some available knowledge. The only methods that provide the ability to manipulate population are the ones based on population level parallelism. This is achieved by distributing the population into subpopulations and thus enables the researcher to use some knowledge factors when, for example, distributing the population. Based on our desire to explore variations of the algorithm with the aim of producing better solution sets the decision to use population level parallelism was taken. Consequently, PMEGA uses population level parallelism, and distributes the population into several smaller subpopulations to be evolved concurrently.

PMEGA operates on one population set referred to as the working population. The algorithm randomly splits the working population to several subpopulations and uses a predefined pool of processes to which it assigns tasks for execution. An example of a task is the independent evolution of a subpopulation set. The number of subpopulations must be greater or equal to the number of the processes, in order to take full advantage of the processes that will handle the tasks.

Subpopulations are evolved independently for a specific number of iterations defined by a user-supplied epoch_counter, which is set to a percentage of the total iterations the algorithm has to run. The default setting of PMEGA is set to 10% of total iterations. The independent evolution of each subpopulation is a scaled-down execution of MEGA algorithm as shown in Figure 10. Specifically, during execution time a pre-constructed process from the pool of processes is assigned a task i.e. to execute a scaled-down MEGA. The working population of the process/task is set to a subpopulation set and the number of iterations is set to the epoch_counter. During the evolution of subpopulations, migrations are not permitted between

the subpopulations. Upon completion of the task, the process returns the results produced and gets assigned a new task, if one is pending.

When all subpopulations complete their evolution, their results are gathered and merged. The new working population is created from the merger of the resulting populations, provided by the set of task executions. The new working population passes a stage where the dominated individuals are removed from it, so that the working population to be a Pareto approximation of the solutions.

Following PMEGA checks for the termination conditions; if satisfied the process terminates. However, if this is not the case the process moves to repeat the previous steps. A diagram of PMEGA is shown in Figure 11.
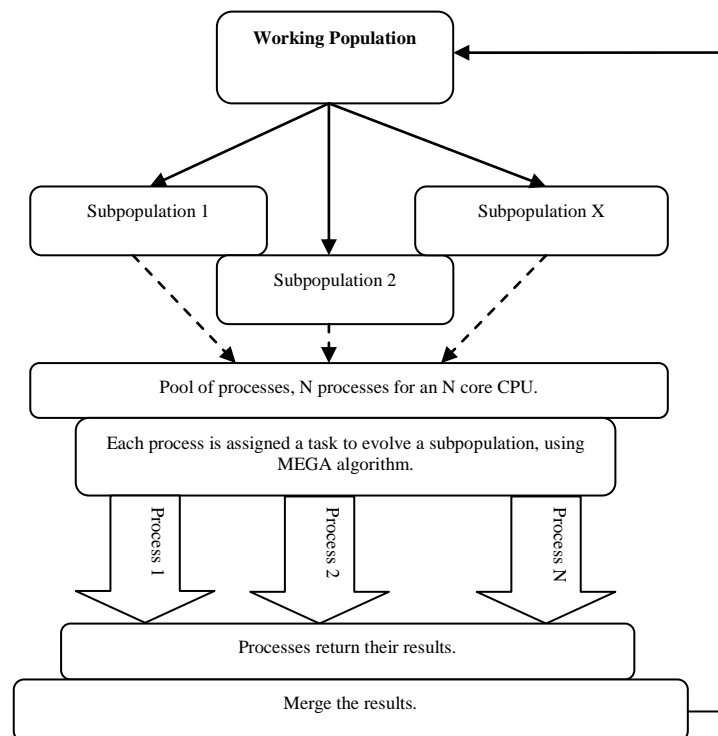
Figure 11: A diagram of the PMEGA algorithm.

The pseudo-code of PMEGA is given below:

```
# Pseudo-Code for PMEGA.
create pool_of_processes;
initialize working_population:
 do while iteration_counter < max_iterations:
         split working_population into several subpopulations;
         create tasks_list; # Each task is to evolve a given subpopulation.
         assign tasks to pool_of_processes;
         wait for results;
         gather results;
         post processing on results;
          create new working_population;
 end do while;

# Process
receive task;
evolve subpopulation for epoch_counter iterations;
prepare results;
send results;
```

Figure 12: The PMEGA algorithm pseudo-code.

Chapter 4

Implementation

Our parallel MEGA implementation uses the Python programming language [19]. Python, a high level scripting language, has been chosen for its ease of use, extensive set of 3rd party add-ons that speed up development, and, inherent object oriented structure. Python's extendability through 3rd party add-ons enables it to be used along with C/C++ for parallel programming and thus makes it promising for high performance computing applications. Currently, there exist several add-ons in the area of parallel programming, including Parallel Python [20] which uses the threads and processes mechanism of Python itself, and MPI4PY [21] which implements a Message Passing Interface (MPI). There are also add-ons building on CUDA [22] and OpenCL [23] for General Purpose GPU (GPGPU) programming such as PyCUDA [24], and PyOpenCL [25].

The best way to achieve parallelism for an algorithm that is to be used in multi-core environment with Python is with the use of processes. This occurs due to the limitations resulting from the manner with which Python manages threads. More specifically, Python has a mechanism called Global Interpreter Lock (GIL) which is a mutual exclusion lock held by Python interpreter to avoid sharing of non thread safe code [26]. This mechanism allows only one thread at a time to access an object. Python can spawn processes in a similar way as is done in Linux with fork. Various 3rd party modules exist, which can aid in multi-processing coding. The new versions of Python, 2.6 and 3.x, are shipped with a multiprocessing module [27]. Users of older versions of Python like 2.5 used here, need to download and install it separately.

4.1 Using Processes

Our experiments exploited multi-processing as offered by Python. Python, through the multiprocessing module [27] offers a very user-friendly API for using processes to achieve parallelism. The API is similar to the one used by the threading module [28]. We decided to use a subpopulations model similar to Figure 10.

The benefit of using processes is due to their ability to execute a large portion of code or even another program separately from the parent process. The subpopulations model provides the flexibility to decide how and when the subpopulations exchange information among them.

To fully utilize an N-core CPU a program needs to spawn at least N processes. Python allows having a pool of processes to which execution tasks can be assigned. According to the subpopulations model, the initial population has to be split into several subpopulation sets. Since we have N processes we must have at least N subpopulations [29].

Python through its multiprocessing module provides us two ways to create a pool of processes. The first one is by using the Pool class to create a new pool object as is defined by the multiprocessing module. The second one is to use the various components, classes and submodules provided by the module to create our own pool implementation. Experimentation was performed with both ways.

4.1.1   Python managed Pool of Processes

Using the implemented Pool class we get an object that creates and manages a defined number of processes, by default equal to the number of cores of the CPU, also called Pool-Workers. This object has a hidden layer of synchronization and communication for the processes created.

The execution model is based on task assignment to the processes of the pool, i.e. tasks for execution are given to the pool and the underlining synchronization mechanism assigns a task to an available process. A task is a function, given for execution, along with the appropriate arguments. The tasks assigned can be different among them. In our case the tasks were executing MEGA but with different arguments each time.

Upon the completion of a task from a Pool-Worker the results of it are sent back to the main process and a new task, if available, gets assigned to the Pool-Worker. This is repeated until all tasks are handled.

After a task is completed the results are collected by the main process, where they are stored for further processing. When all tasks are completed the collected results pass through a processing step from which the following is derived:

- The new working population and

- An updated chromosome cache which keeps track of the molecules created in past generations.

This scheme is repeated for a defined number of iterations.

When using this approach the programmer does not have any responsibility about the processes communication and synchronization. The responsibility focuses on the function(s) to be executed in parallel, their arguments and possible shared resources, which are best to be avoided due to their non-trivial way of handling. This approach has many limitations, but ensures good behaviour, when following the right steps.

4.1.2 Programmer managed Pool of Processes

The other way to create a pool of processes is by using classes, components and submodules of the multiprocessing module to create a "system" of multiple Processes that communicate through queues and are synchronized through the use of locks and events.

In this approach we create a number of processes, by default equal to the number of the cores of the CPU, which execute a function that waits for a task to execute. The task is a set of data which the function waits as input for the evolution algorithm. The function executes the MEGA in a repeated fashion.

The communication between the main process and the child processes is achieved with the use of queues. One queue, called taskQueue, is used to submit the tasks to the child processes from the main process and one queue, called doneQueue, is used to submit the results from child processes to main process. Queues are protected by locks.

An Event is used to signal all child processes that initialization has completed so that they can start their repeated evolutionary part. At this stage the child processes wait for a set of input data, the task, from the taskQueue. With the completion of the task the results are inserted in the doneQueue, from where the main process retrieves them. The processes then proceed to the next available task.

The main process retrieves results from doneQueue and stores them to its local variables for further processing after the completion of all tasks. This processing gives:

- The new working population and
- An updated chromosome cache

This scheme is repeated for a defined number of iterations.

When using this approach the programmer has responsibility over the whole "system", starting from the spawning and starting of the processes, the handling of the communication between main and child processes, the data collections sent throughout the execution, the synchronization of tasks and, ending with the normal completion of the spawned processes to avoid zombie processes in the system. This approach gives the programmer a lot of control ability, has fewer limitations on things to use, but encapsulates high risk in creating faults.

Chapter 5

Results

Some of the experiments held for the purpose of this thesis are shown below. The experiments were designed and executed in order to investigate the behaviour of the proposed PMEGA algorithm versus the established MEGA.

Two features of PMEGA that change its behaviour are:

- Number of subpopulations: for a fixed population size the more subpopulations we have the smaller they get.

- Isolation time (epoch_counter): defines the period of time/iterations for which the subpopulations evolve independently.

In the current implementation of PMEGA both of these variables can be controlled by the user. For the purposes of this thesis, we decided to study the effect of the number and the size of subpopulations on the behaviour of the algorithm. The isolation time remained the same for all experiments and was set to a default value of 10% of maximum iterations, which were set to 200.

The quality measures used to compare the proposed solutions across algorithms are based on clustering. More specific, diversity analysis on objective and parameter space makes use of agglomerative clustering [30]. In objective space the clustering is based on the objective values of the proposed solutions while in parameter space the clustering is based on the molecular characteristics of the proposed solutions.

5.1 MEGA versus PMEGA on 2-core CPU

This section summarizes the experimental results from the execution of MEGA versus the process-enabled version of PMEGA using a two core CPU machine.

The specifications of our testing machine are:

- Machine 1:

  - CPU: Intel Core 2 Duo E8400 @ 3.0 GHz (2 Physical Cores)

  - Memory: 4 Gbytes

5.1.1   Trial Experiment – MEGA vs. PMEGA (2 subpopulations)

The results shown below have also been reported during the ITAB 2009 Conference [31]. The purpose of this experiment was to compare the quality of the solutions of MEGA with PMEGA. This experiment was among the first we be executed, and its role was to give as an indication for the speedup and efficiency of the parallel algorithm.

The experiment had two objectives, a population of 100 and executed for 200 iterations. The objectives were based on similarity on three ligands that are known to be selective to the Estrogen Receptor-beta and dissimilarity on two ligands known to be selective to the Estrogen Receptor-alpha. Dissimilarity is calculated using the Tanimoto coefficient [32] in order to represent the problem as bi-objective minimization problem while similarity is calculated using the Soergel measure, which is the complement of Tanimoto coefficient. All the datasets used by the experiment were taken from PubChem's [33] compound library. The initial population was taken from the compounds in Bioassay 713. A set of 3662 building blocks was used, which were obtained via fragmentation of the compounds in Bioassay 1211 using the NSisToolkit0.9 package [34]. This setup ran for five times for each of the two algorithm versions using different initial population each time. The specific problem examined is that of de novo design. In this setting, PMEGA chromosomes represent molecular graphs where vertices correspond to atoms and edges correspond to bonds. An explanation of the DND problem can be found in section 2.3.

Table 1 presents the execution times collected from the experiment. The timing results show a speedup of almost 1.6 on a dual core CPU. This is satisfactory considering that parts of the PMEGA algorithm are executed serially.

Figure 13 is a graphical demonstration of the solutions of a run of the MEGA algorithm. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red spots represent the individuals of the working populations over a period of iterations.

Figure 14 is a graphical demonstration of the solutions of a run of the PMEGA algorithm on the same problem. Same as in figure 7 the X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red spots represent the individuals of the working populations over a period of iterations.

Figure 15 compares all Pareto fronts of the MEGA and PMEGA algorithms from all runs performed. The first graph shows the spacing between the solutions, where spacing describes the uniformity of the spread of the solutions of the Pareto-approximation produced [18]. Bigger values indicate a larger spread of solutions. The second graph represents the diversity of the solutions in parameter/genotype space, which is based on the similarity distance of the graph chromosomes. The bigger the value is, the more diverse the individuals are. The third graph describes the diversity of the solutions in objective/phenotype space, which reflects the similarity of individuals with respect to objective values. The bigger the value is, the more distant the solutions are.

High values of diversity, both in parameter and objective space are desirable since they indicate a wealth of equivalent solutions structurally different for the user to evaluate. The comparisons demonstrate the similar quality of the results produced by MEGA and PMEGA.
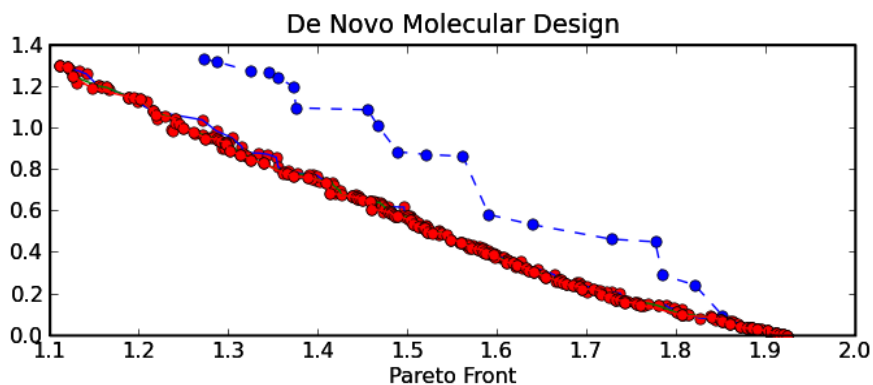
Figure 13: A graph of Pareto fronts taken from one of the runs for MEGA. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working populations over a period of iterations. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
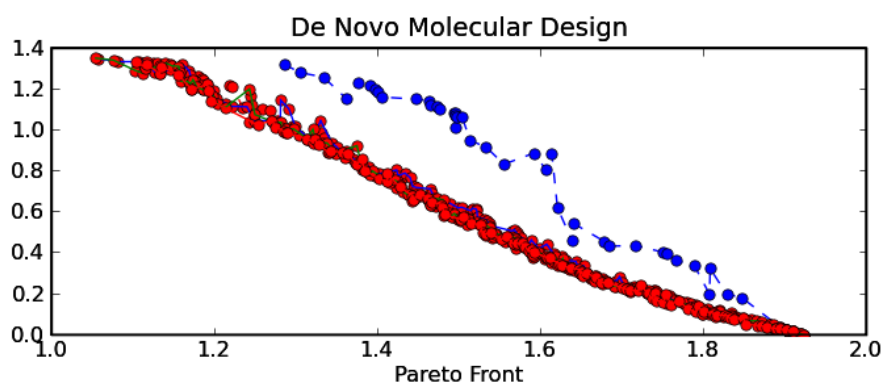
Figure 14: A graph of Pareto fronts taken from one of the runs for PMEGA. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
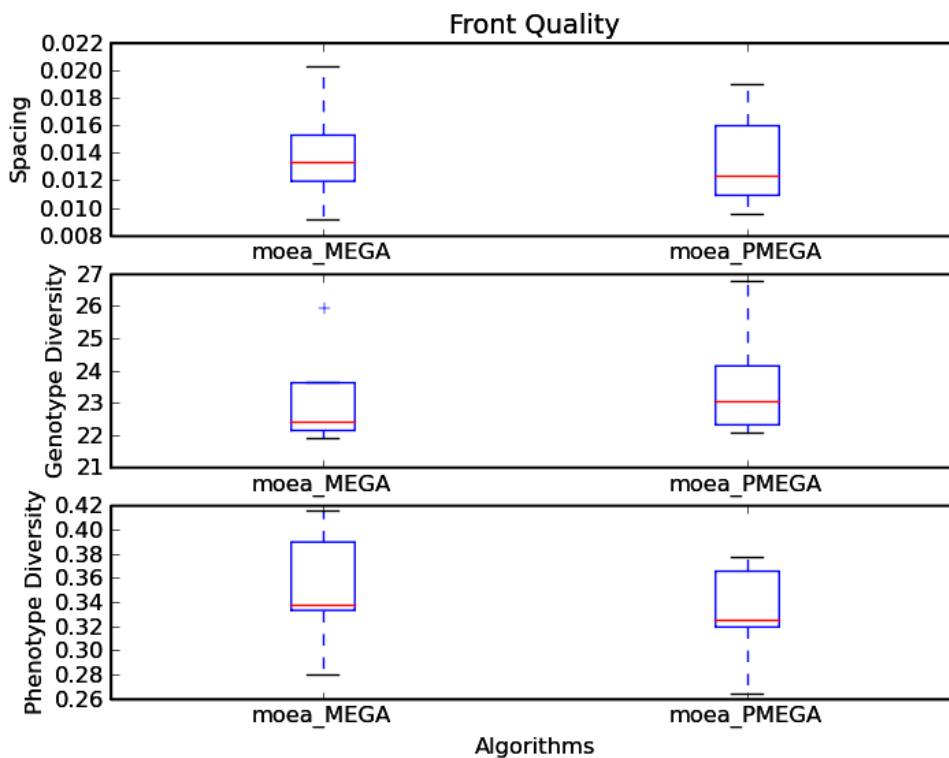
Figure 15: A comparison of MEGA and PMEGA Pareto fronts. The first graph shows how dense the solutions are. The second graph shows how spread are the solutions in parameter space. The third graph shows how spread are the solutions in objective space.

Table 1

Time results from the experiment, calculations for Average execution time, Speedup and Efficiency.

	MEGA	PMEGA
Run 0	00:16:26	00:11:32
Run 1	00:17:02	00:10:20
Run 2	00:18:07	00:11:02
Run 3	00:18:13	00:11:27
Run 4	00:17:32	00:10:53
Total	1:27:20	0:55:14
Max	00:18:13	00:11:32
Min	00:16:26	00:10:20
Average	00:17:34	00:11:07
Speedup		1.5789211
Efficiency		0.7894605

Time measured is Wall Clock Time (Total Execution Time).

Time format is HH:MM:SS.

Average is calculated as (Total – Max – Min) / (5 - 2).

Speedup is calculated as Time of serial / Time of parallel.

Efficiency is calculated as Speedup / Number of processes.

5.1.2   1st Experiment – MEGA vs. PMEGA (2 Subpopulations) vs. PMEGA (4 Subpopulations)

This second set of experiments took place with an improved version of the algorithm. The improvements of the algorithm are focused on the selection of nondominated solutions after the merger, so that the working population is guaranteed to be a Pareto approximation of the solutions. In these set of experiment we would like to see the behaviour of the algorithm with larger data sets and with more subpopulations. The objectives were slightly different this time.

The experiment had two objectives, a population of 100 which executed for 200 iterations. For PMEGA the subpopulations variable number was set to 2 and 4 subpopulations. The purpose of the experiment was to study the effect of the number and the size of the subpopulations on the quality of the solutions and on the execution time.

The objectives were based on similarity on two ligands known to be selective to the Estrogen Receptor-alpha and dissimilarity on three ligands that are known to be selective to the Estrogen Receptor-beta. Dissimilarity is calculated using the Tanimoto coefficient [32] in order to represent the problem as bi-objective minimization problem while similarity is calculated using the Soergel measure, which is the complement of Tanimoto coefficient. All the datasets used by the experiment were taken from PubChem's [33]. The initial population was taken from the compounds in Bioassay 1211. A set of 3662 building blocks was used, which were obtained via fragmentation of the compounds in Bioassay 1211 using the NSisToolkit0.9 package [34]. This setup ran for five times for each of the two algorithm versions using different initial population each time. The specific problem examined is that of chemical structure design, also known as de novo design (DND). In this setting, PMEGA chromosomes represent molecular graphs where vertices correspond to atoms and edges correspond to bonds. An explanation of the DND problem can be found in section 2.3.

Table 2 shows the execution time of the experiment. We should note here that the algorithm used had some improvements at the direction of the quality of solutions obtained, so some post processing after the parallel section was added in order to select a Pareto approximation of the solutions. This post processing lowers the parallel percentage of the algorithm by a small factor, which as shown by the execution timings has a minor effect on speedup and efficiency. Speedup measured is 1.7 and efficiency of 0.86. The other important result seen in Table 2 is the time results of PMEGA using 4 subpopulations handled by 2 processes which is slightly slower than the one using 2 subpopulations. These subpopulations have a size of 25 individuals compared to the 50 individuals when using 2 subpopulations.

Figure 16 is a graphical demonstration of the solutions of a run of the MEGA algorithm. The X and Y axis represent the two objectives. Note that both objectives need to be

minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red spots represent the individuals of the working populations over a period of iterations.

Figure 17 is a graphical demonstration of the solutions of a run of the PMEGA with 2 Subpopulations algorithm on the same problem.

Figure 18 is a graphical demonstration of the solutions of a run of the PMEGA with 4 Subpopulations algorithm on the same problem.

Figure 19 compares all Pareto fronts of the MEGA and PMEGA algorithms from all runs performed.

The comparisons demonstrate the similar quality of the results produced by MEGA and PMEGA. The number of subpopulations used has a minor affect on the quality of the solutions. The second graph shows that PMEGA with 2 Subpopulations (PMEGA_2) has the same diversity with PMEGA with 4 Subpopulations (PMEGA_4) in parameter space. The third graph shows that PMEGA_2 has more diverse solutions than PMEGA_4 in objective space. The reason for this is probably that for each subpopulation the algorithm scans a specific part of the search space. When having a global population it scans the entire search space, when having two subpopulations, each subpopulation focuses on a different part of the search space, which is depended on the nature of the individuals of the subpopulation, this is why the solutions produced will be more diverse between them. When having more than two subpopulations it seems that somehow that the subpopulations spread in such manner that their solutions create a denser Pareto front.
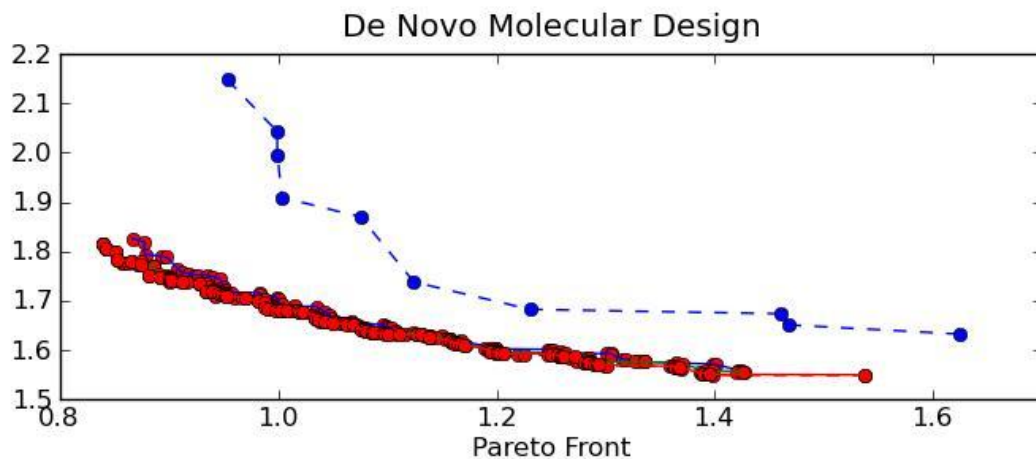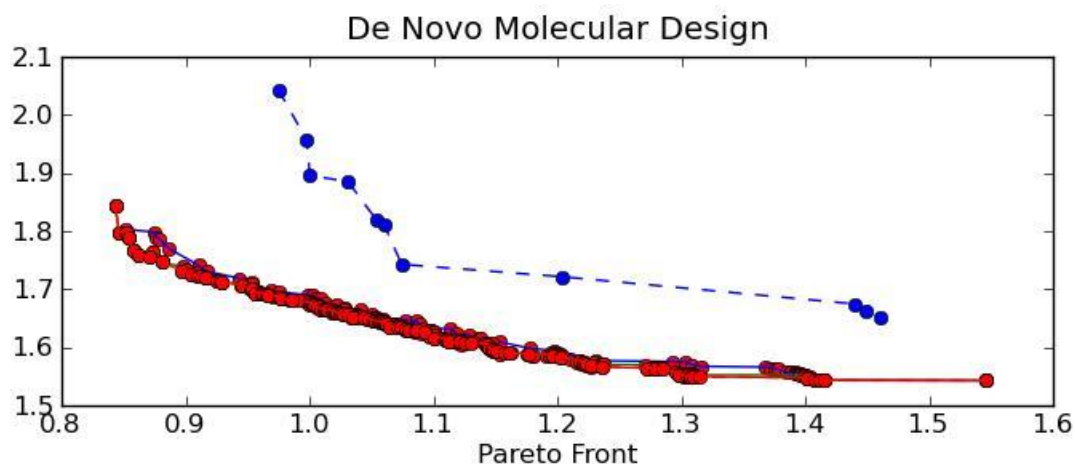
Figure 16: A graph of Pareto fronts taken from one of the runs for MEGA. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working populations over a period of iterations. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
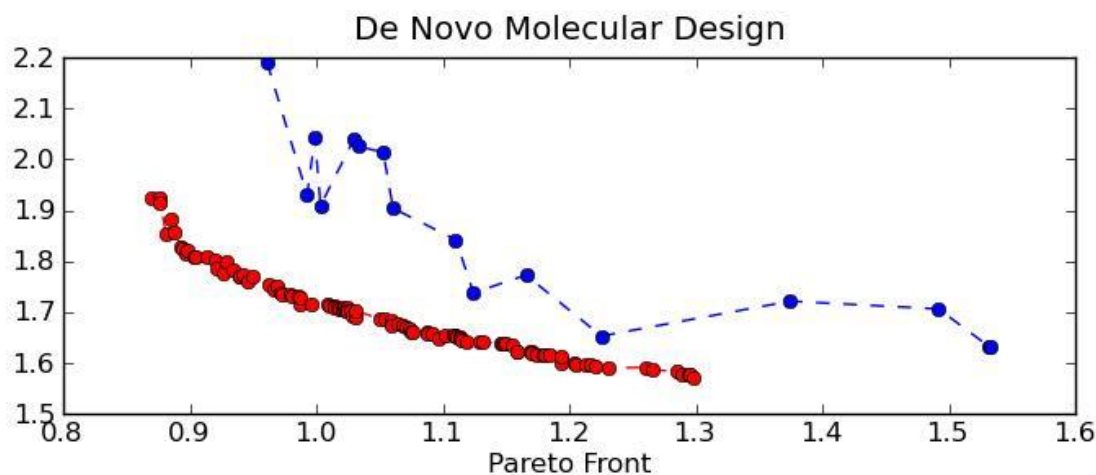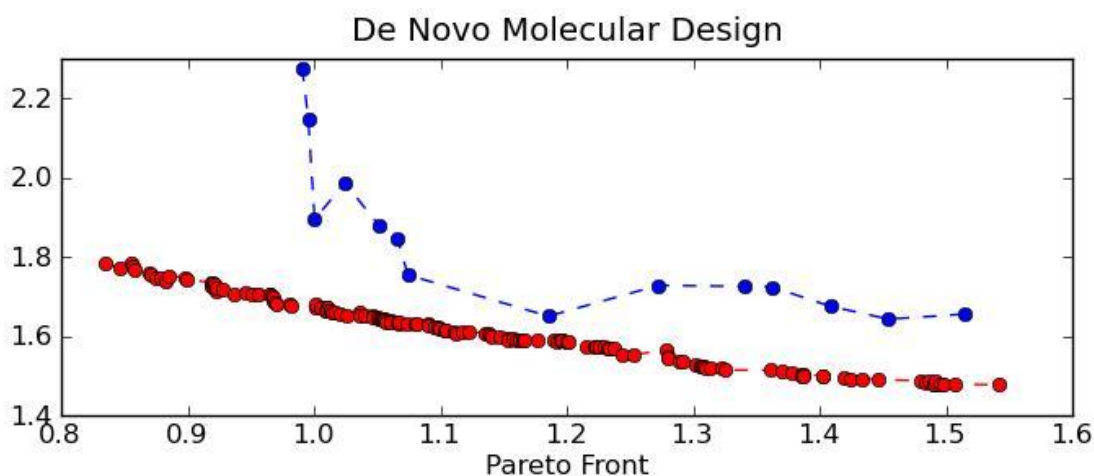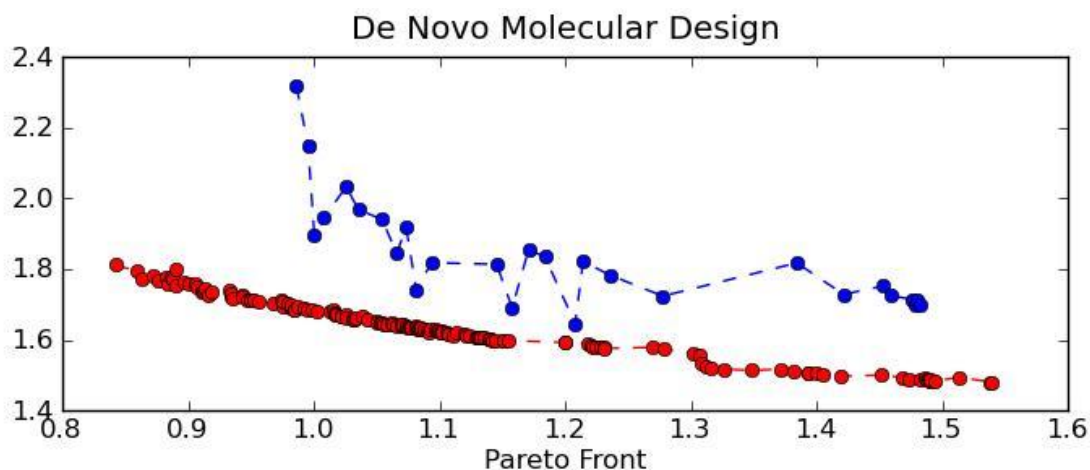

Figure 17: A graph of Pareto fronts taken from one of the runs for PMEGA with 2 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
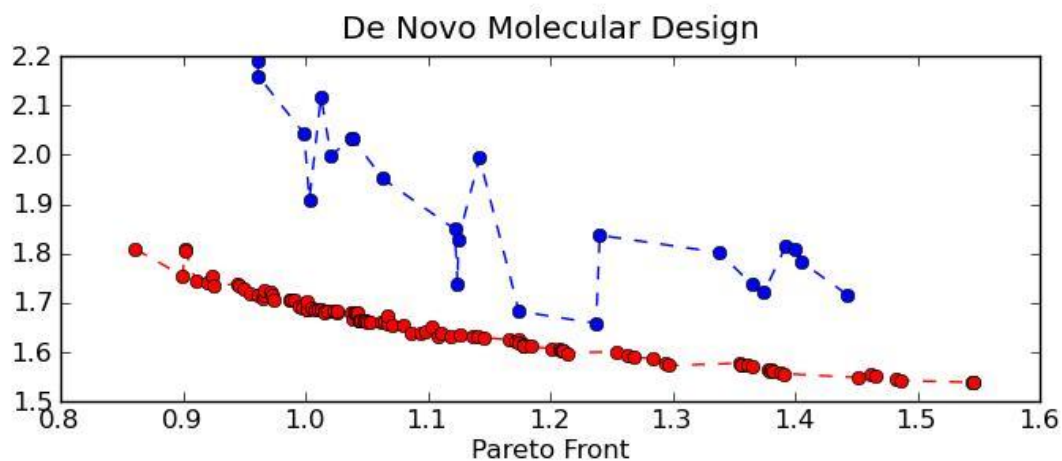
Figure 18: A graph of Pareto fronts taken from one of the runs for PMEGA with 4 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
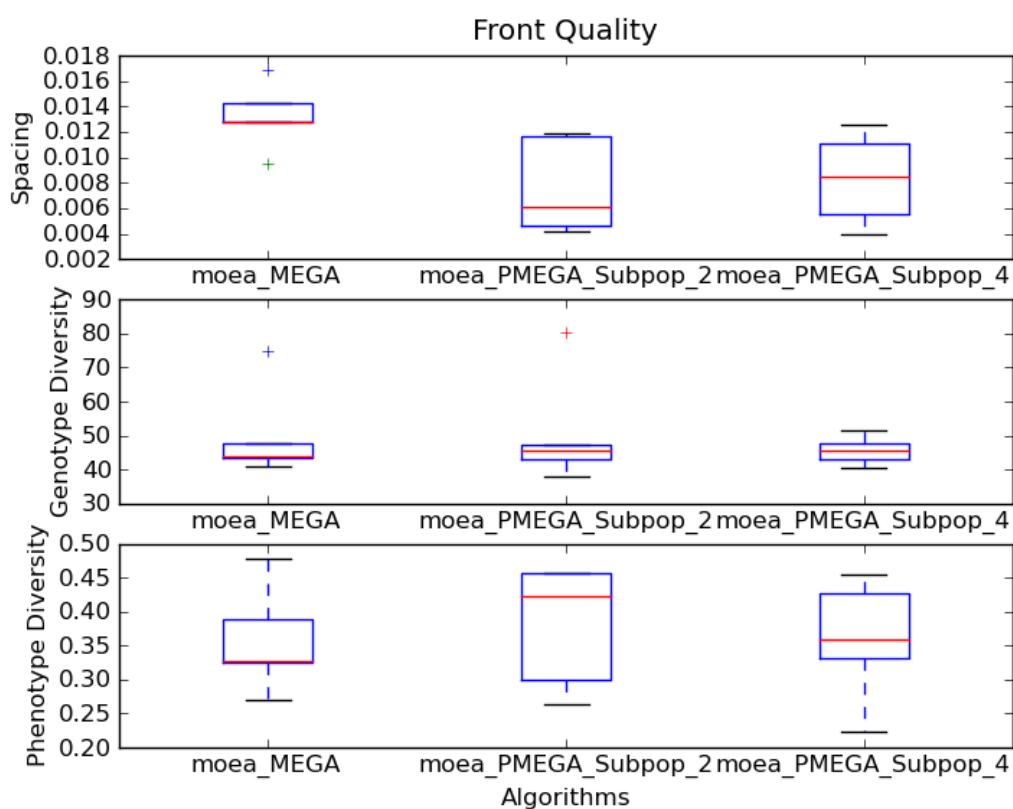
Figure 19: A comparison of MEGA, PMEGA with 2 Subpopulations and PMEGA with 4 Subpopulations Pareto fronts.

Table 2

Time results from the experiment, calculations for Average execution time, Speedup and Efficiency.

	MEGA	PMEGA 2 Subpop	PMEGA 4 Subpop
Run 0	0:40:23	0:22:53	0:25:32
Run 1	0:36:05	0:24:40	0:23:40
Run 2	0:38:58	0:22:41	0:28:05
Run 3	0:45:38	0:22:37	0:27:27
Run 4	0:38:25	0:22:28	0:24:32
Total	3:19:29	1:55:19	2:09:16
Max	0:45:38	0:24:40	0:28:05
Min	0:36:05	0:22:28	0:23:40
Average	0:39:15	0:22:44	0:25:50
Speedup		1.727206062	1.519243174
Efficiency		0.863603031	0.759621587

Time measured is Wall Clock Time (Total Execution Time).

Time format is HH:MM:SS.

Average is calculated as (Total – Max – Min) / (5 - 2).

Speedup is calculated as Time of serial / Time of parallel.

Efficiency is calculated as Speedup / Number of processes.

5.1.3   2nd Experiment – MEGA vs. PMEGA (2 Subpopulations) vs. PMEGA (4 Subpopulations)

Continuing with the experiments about the effect of subpopulation size on the quality of the results, this experiment increased the size of the population to 150. By doing so the size of the subpopulations was 75 when the algorithm used 2 subpopulations and 37-38 when it used 4 subpopulations.

The experiment had two objectives, a population of 150 which executed for 200 iterations. Also for PMEGA another variable is the subpopulations number, which was set to 2 and 4 subpopulations. The purpose of the experiment was to study the effect of the number and the size of the subpopulations on the quality of the solutions and on the execution time.

This experiment had the same objectives and datasets as the experiment described in section 5.1.2.

Table 3 shows the execution time of the experiment and the speedup gained. As with the previous experiment, Table 3 is showing, the time results of PMEGA using 4 subpopulations with 37-38 individuals and with 2 subpopulations having 75 individuals.
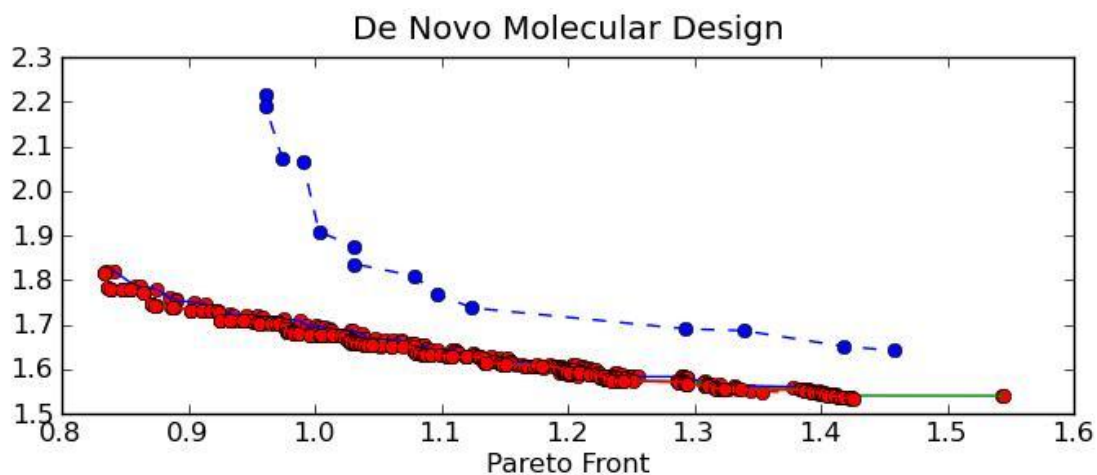
Figure 20 is a graphical demonstration of the solutions of a run of the MEGA algorithm. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red spots represent the individuals of the working populations over a period of iterations.

Figure 21 is a graphical demonstration of the solutions of a run of the PMEGA with 2 Subpopulations algorithm on the same problem.

Figure 22 is a graphical demonstration of the solutions of a run of the PMEGA with 4 Subpopulations algorithm on the same problem.

Figure 23 compares all Pareto fronts of the MEGA and PMEGA algorithms from all runs performed.

The comparisons demonstrate the similar quality of the results produced by MEGA and PMEGA. Here the observations are the same as at the 1st experiment, in parameter space the solutions are similar, and in objective space using two subpopulations gives more diverse solutions.

Figure 20: A graph of Pareto fronts taken from one of the runs for MEGA. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working populations over a period of iterations. The red dots represent the individuals of the Pareto Fronts over a period of iterations.

Figure 21: A graph of Pareto fronts taken from one of the runs for PMEGA with 2 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.

Figure 22: A graph of Pareto fronts taken from one of the runs for PMEGA with 4 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
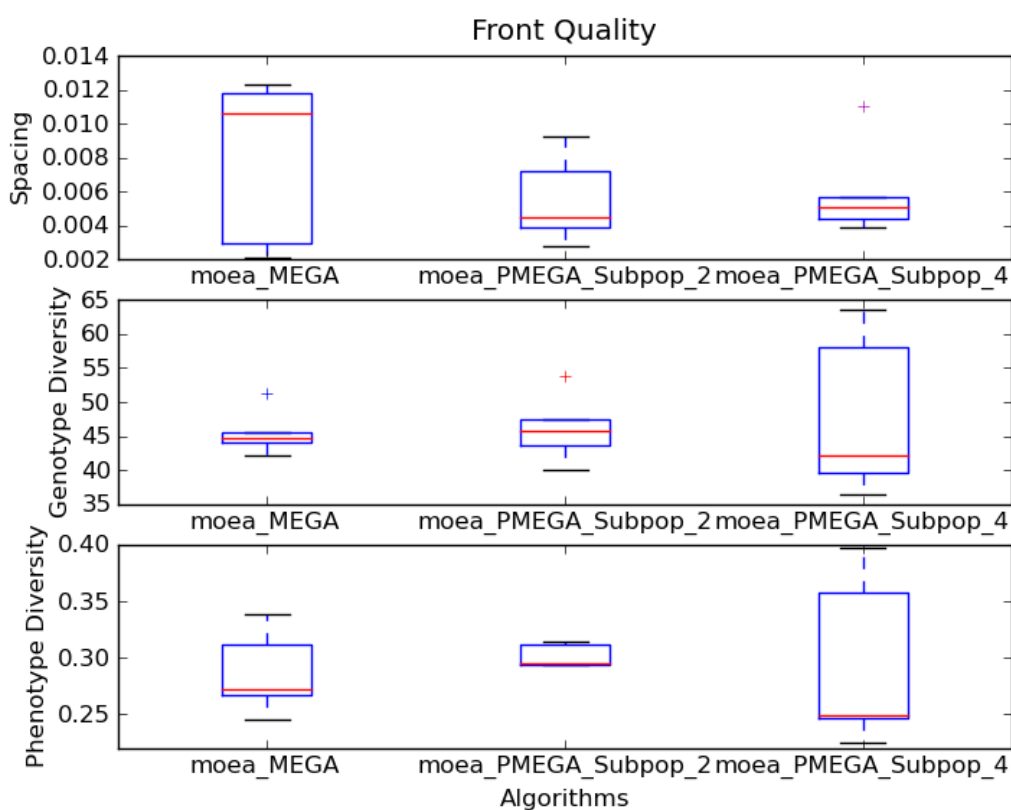
Figure 23: A comparison of MEGA, PMEGA with 2 Subpopulations and PMEGA with 4 Subpopulations Pareto fronts.

Table 3

Time results from the experiment, calculations for Average execution time, Speedup and Efficiency.

	MEGA	PMEGA 2 Subpop	PMEGA 4 Subpop
Run 0	1:02:38	0:36:32	0:37:19
Run 1	1:09:36	0:33:58	0:42:04
Run 2	1:13:26	0:35:37	0:45:51
Run 3	1:01:57	0:34:48	0:39:47
Run 4	1:01:42	0:34:00	0:39:47
Total	5:29:19	2:54:55	3:24:48
Max	1:13:26	0:36:32	0:45:51
Min	1:01:42	0:33:58	0:37:19
Average	1:04:44	0:34:48	0:40:33
Speedup		1.859696728	1.596464785
Efficiency		0.929848364	0.798232392

Time measured is Wall Clock Time (Total Execution Time).

Time format is HH:MM:SS.

Average is calculated as (Total – Max – Min) / (5 - 2).

Speedup is calculated as Time of serial / Time of parallel.

Efficiency is calculated as Speedup / Number of processes.

5.1.4   3rd Experiment – MEGA vs. PMEGA (2 Subpopulations) vs. PMEGA (4 Subpopulations)

Continuing with the experiments about the effect of subpopulation size on the quality of the results, this experiment increased the size of the population to 200. By doing so the size of the subpopulations was 100 when the algorithm used 2 subpopulations and 50 when it used 4 subpopulations.

The experiment had two objectives, a population of 200 which executed for 200 iterations. Also for PMEGA another variable is the subpopulations number, which was set to 2 and 4 subpopulations. The purpose of the experiment was to study the effect of the number and the size of the subpopulations on the quality of the solutions and on the execution time.

This experiment had the same objectives and datasets as the experiment described in section 5.1.2.

Table 4 shows the execution time of the experiment and the speedup gained. As on the previous experiment, Table 4 is showing, the time results of PMEGA using 4 subpopulations with 50 individuals and with 2 subpopulations having 100 individuals.

Figure 24 is a graphical demonstration of the solutions of a run of the MEGA algorithm. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red spots represent the individuals of the working populations over a period of iterations.

Figure 25 is a graphical demonstration of the solutions of a run of the PMEGA with 2 Subpopulations algorithm on the same problem.

Figure 26 is a graphical demonstration of the solutions of a run of the PMEGA with 4 Subpopulations algorithm on the same problem.

Figure 27 compares all Pareto fronts of the MEGA and PMEGA algorithms from all runs performed.

The comparisons demonstrate the similar quality of the results produced by MEGA and PMEGA. Here the observations are the same as at the 1st and 2nd experiment, in parameter space the solutions are similar, and in objective space using two subpopulations gives more diverse solutions.

Figure 24: A graph of Pareto fronts taken from one of the runs for MEGA. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working populations over a period of iterations. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
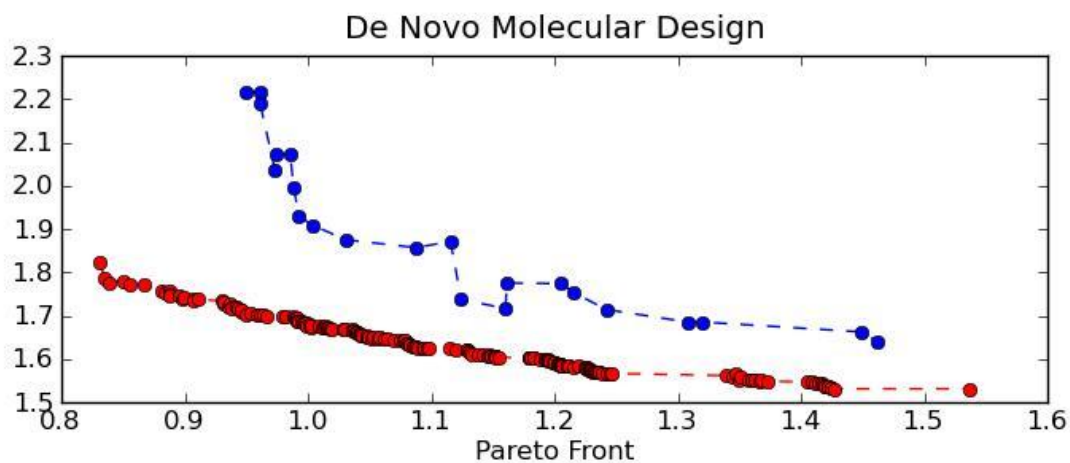
Figure 25: A graph of Pareto fronts taken from one of the runs for PMEGA with 2 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
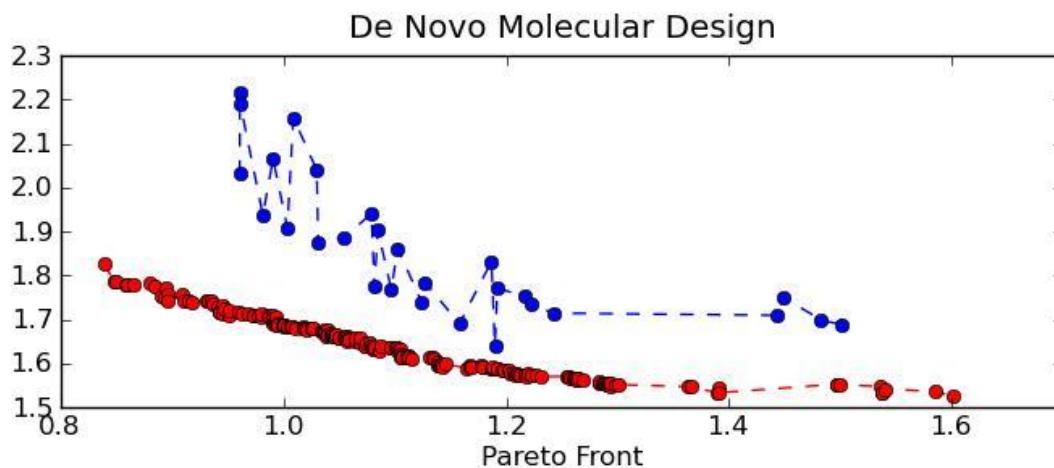
Figure 26: A graph of Pareto fronts taken from one of the runs for PMEGA with 4 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
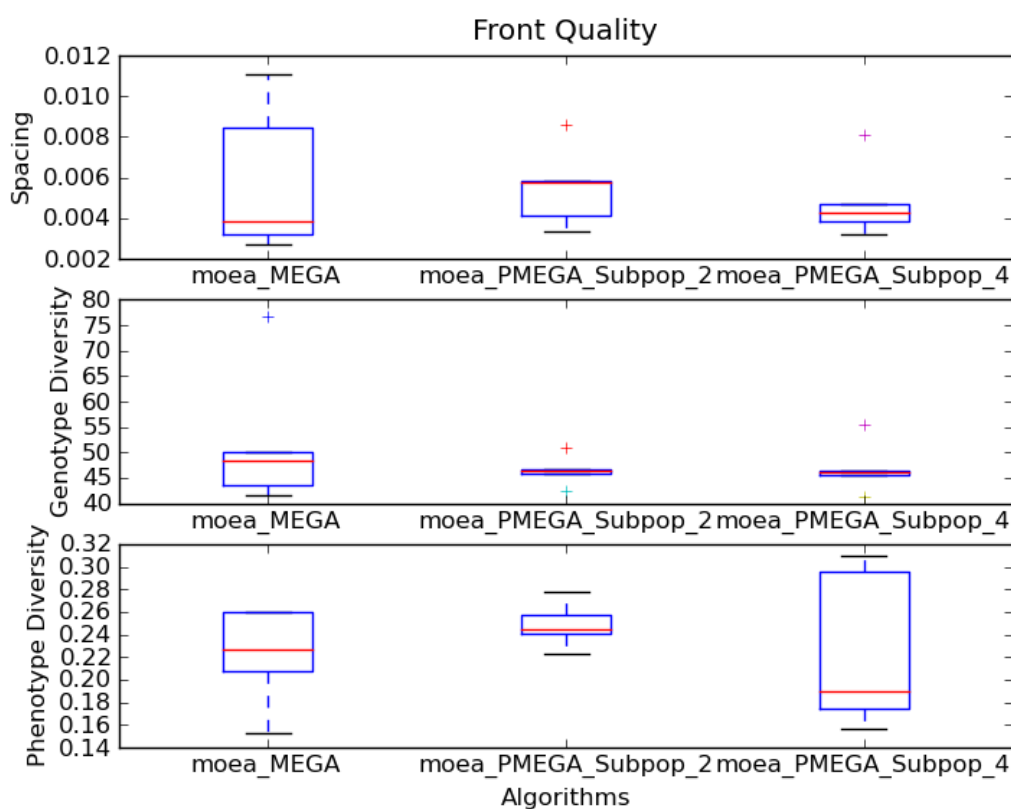
Figure 27: A comparison of MEGA, PMEGA with 2 Subpopulations and PMEGA with 4 Subpopulations Pareto fronts.

Table 4
Time results from the experiment, calculations for Average execution time, Speedup and Efficiency.

	MEGA	PMEGA 2 Subpop	PMEGA 4 Subpop
Run 0	1:23:17	0:50:44	0:57:39
Run 1	1:21:08	0:46:22	0:53:57
Run 2	1:25:46	0:48:13	0:52:53
Run 3	1:20:19	0:52:44	0:54:21
Run 4	1:22:35	0:49:10	0:54:49
Total	6:53:05	4:07:13	4:33:39
Max	1:25:46	0:52:44	0:57:39
Min	1:20:19	0:46:22	0:52:53
Average	1:22:20	0:49:22	0:54:22
Speedup		1.667604366	1.514253602
Efficiency		0.833802183	0.757126801

Time measured is Wall Clock Time (Total Execution Time).

Time format is HH:MM:SS.

Average is calculated as (Total – Max – Min) / (5 - 2).

Speedup is calculated as Time of serial / Time of parallel.

Efficiency is calculated as Speedup / Number of processes.

5.1.5 Summary

This section summarizes the results from the experiments that were executed on the two cores CPU. A summary of the quality of the solutions is shown, comparing a random sample of the proposed solutions with the requested objectives. Also a summary of the gained speedup is shown.

The graphs displayed in the experiments sections show a general trend of the solutions, but choosing a random sample of the proposed solutions and comparing them with the

requested objectives gives a more detailed view of how well the algorithms work. The 2D representations of the molecules were obtained using the online tools of MolInspiration [35].

Table 5 shows the ligands that are known to be active with ER-a, which are used for the similarity objective. Table 6 shows the ligands that are known to be active with ER-b, which are used for the dissimilarity objective.

Table 7 shows a random sample of the algorithm's proposed compounds. The solutions displayed are grouped by the algorithm used to produce them. Using the solutions provided by MEGA as a reference to compare with the solutions provided by PMEGA, the following observations are reported:

- Some solutions of PMEGA are similar to solutions of MEGA.

- Some solutions provided by the same algorithm are similar between them.

Table 8 shows a summary of the experiments executed on the two cores CPU. From measured speedup on all the experiments the average speedup is calculated to be used with Amdahl's Law for the calculation of the maximum expected speedup of the parallel algorithm. From this set of experiments the average measured speedup is 1.8 and the efficiency obtained with this speedup is 0.9.

Table 5
Compounds that algorithm's resulting compounds must be similar to. These compounds are known to be active with ER-a.

| Oc3ccc(c2oc(=O)c1cc(O)ccc1c2P)cc3 | CCCc3c(c1ccc(O)cc1)nn(c2ccc(O)cc2)c3c4ccc(O)cc4NO |
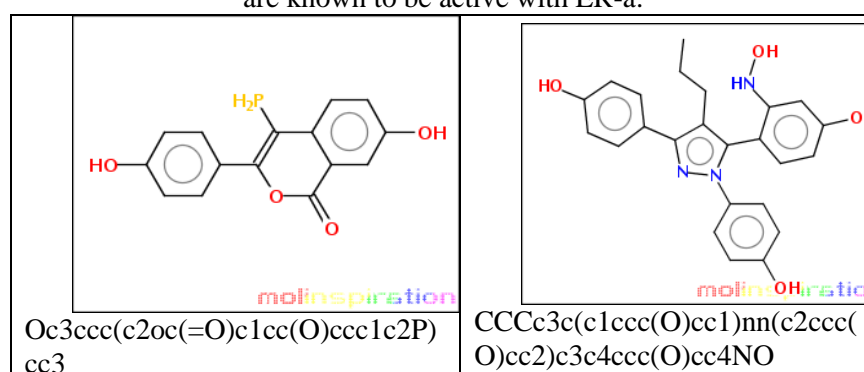
Table 6
Compounds that algorithm's resulting compounds must not be similar to. These compounds are known to be active with ER-b.
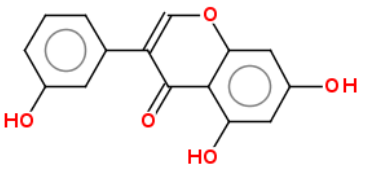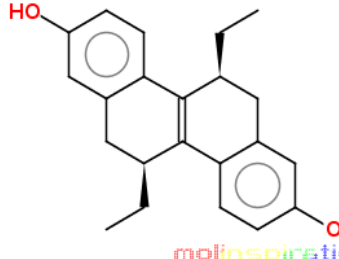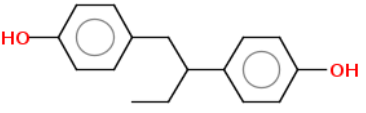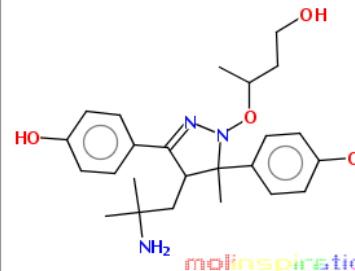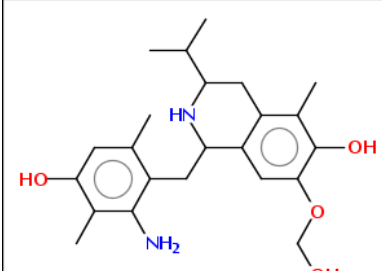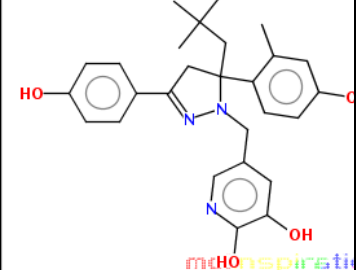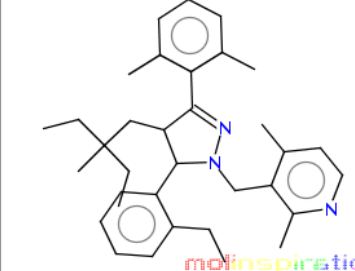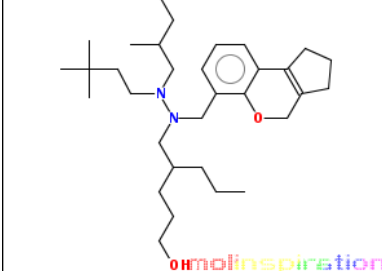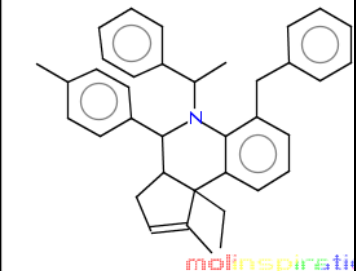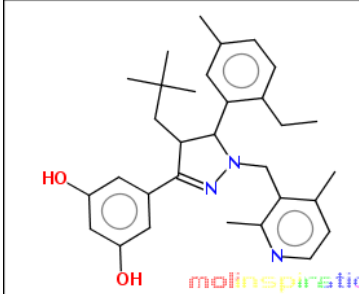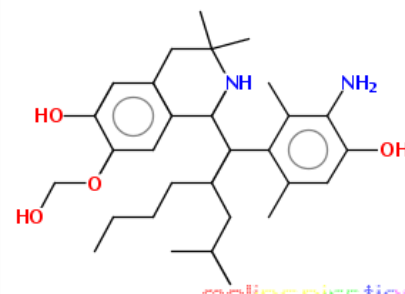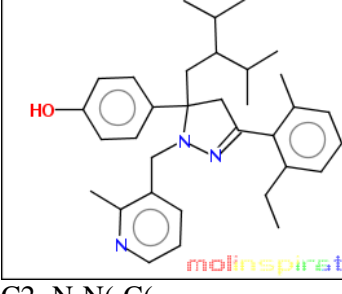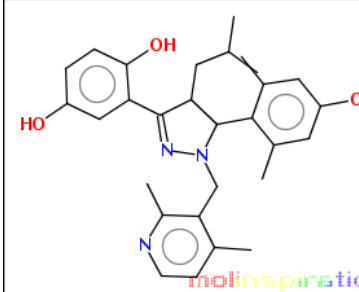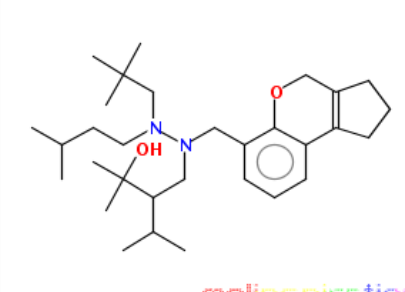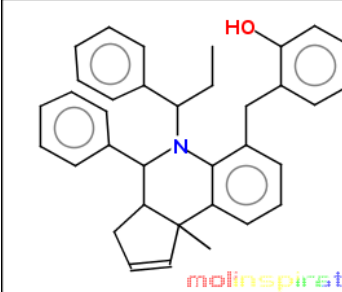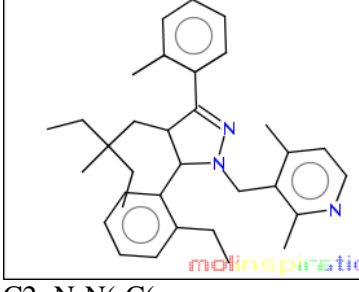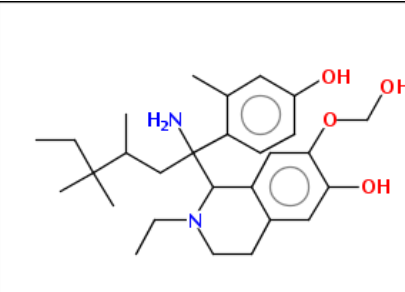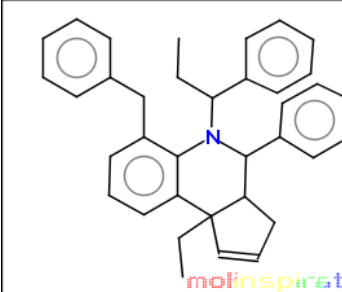
Oc3cccc(c2coc1cc(O)cc(O)c1c2=O)c3	CC[C@H]2Cc1cc(O)ccc1C4=C2c3ccc(O)cc3C[C@@H]4CC
CCC(Cc1ccc(O)cc1)c2ccc(O)cc2	

Table 7
Random sample of proposed solutions taken from MEGA and PMEGA.

MEGA	PMEGA with 2 Subpopulations	PMEGA with 4 Subpopulations
C1(-c2:c:c:c:c(:c:c:2)-O)=N-N(-C(-c3:c:c:c:c(:c:c:3)-O)(-C-1-C-C(-N)(-C)-C)-C)-O-C(-C-C-O)-C (MP_72_M_70)	c1(:c(:c(:c(:c2:c(:c:1)-C(-N-C(-C(-C)-C)-C-2)-C-c3:c(:c(:c(:c:c:3-C)-O)-C)-N)-C)-O)-O-C-O (SP_0_192_M_9)	c1(:c(:c:c(:c:c:c:1)-O)-C)-C2(-N(-N=C(-c3:c:c:c(:c:c:3)-O)-C-2)-C-c4:c:c(:c(:c(:n:c:4)-O)-O)-C-C(-C)(-C)-C (SP_2_177_M_11)
c1(-C2=N-N(-C(-c3:c(:c:c:c:c:c:3)-C-C)-C-2-C-C(-	c12:c(-C3=C(-C-O-1)-C-C-C-3):c:c:c:c:2-C-N(-N(-C-C(-C-C)-C)-	c12:c(-C3(-C(=C-C-C-3-C(-c4:c:c:c:c(:c:c:4)-C)-N-1-C(-

C-C)(-C-C)-C)-C- c4:c(:n:c:c:c:4-C)- C):c(:c:c:c:c:1-C)-C (MP_189_M_35)	C-C-C(-C)(-C)-C)-C-C(-C-C-C-O)- C-C-C (SP_1_187_M_7)	c5:c:c:c:c:c:5)-C)-C- C):c:c:c:c:2-C-c6:c:c:c:c:c:6 (SP_0_180_M_2)
C1(-c2:c:c(:c:c(:c:2)-O)-O)=N- N(-C(-c3:c(:c:c:c(:c:3)-C)-C-C)- C-1-C-C(-C)(-C)-C)-C- c4:c(:n:c:c:c:4-C)-C (MP_196_M_6)	c1(:c(:c:c(:c(:c:c:1-C)-O)-N)-C)-C(- C2-c3:c(:c:c(:c(-O-C-O):c:3)-O)-C- C(-N-2)(-C)-C)-C(-C-C(-C)-C)-C-C- C-C (SP_0_172_M_14)	c1(-C2=N-N(-C(- c3:c:c:c:c(:c:c:3)-O)(-C-2)-C-C(- C(-C)-C)-C(-C)-C)-C- c4:c(:n:c:c:c:4)-C):c(:c:c:c:c:1- C)-C-C (SP_1_194_M_1)
c1(-C2=N-N(-C(- c3:c(:c:c(:c:c:3-C)-O)-C)-C-2-C- C(-C)-C)-C-c4:c(:n:c:c:c:4-C)- C):c(:c:c:c(:c:1)-O)-O (MP_179_M_66)	c12:c(-C3=C(-C-O-1)-C-C-C- 3):c:c:c:c:2-C-N(-N(-C-C(-C)(-C)- C)-C-C-C(-C)-C)-C-C(-C(-O)(-C)- C)-C(-C)-C (SP_0_157_M_44)	c12:c(-C3(-C=C-C-C-3-C(- c4:c:c:c:c:c:4)-N-1-C(- c5:c:c:c:c:c:5)-C-C)- C):c:c:c:c:2-C-c6:c(:c:c:c:c:6)-O (SP_3_188_M_16)
c1(-C2=N-N(-C(- c3:c(:c:c:c:c:3)-C-C)-C-2-C-C(- C-C)(-C-C)-C)-C- c4:c(:n:c:c:c:4-C)- C):c(:c:c:c:c:1)-C (MP_162_M_69)	c1(:c(:c:c(:c:c:1)-O)-C)-C(-C2- c3:c(:c:c(:c(-O-C-O):c:3)-O)-C-C-N- 2-C-C)(-C-C(-C(-C-C)(-C)-C)-C)-N (SP_1_199_M_4)	c12:c(-C3(-C=C-C-C-3-C(- c4:c:c:c:c:c:4)-N-1-C(- c5:c:c:c:c:c:5)-C-C)-C- C):c:c:c:c:2-C-c6:c:c:c:c:c:6 (SP_2_164_M_22)

| c1(-C2=N-N(-C(-c3:c:c(:c:c:c:3)-C)-C-2-C-C-C-C(-C)-C)-C-c4:c(:n:c:c(:c:4)-C)-C-C):c(:c:c:c:c:1-C)-C (MP_198_M_71) | c1(:c(:c:c2:c(:c:1)-C(-C(-c3:c(:c(:c(:c(:c:3)-C)-O)-N)-C)-C(-C-C(-C)-C)-C)-N-C-C-2)-O)-O-C-O (SP_1_190_M_48) | c12:c(:c(:c:c:c:1-C3-C=C(-C(-C-3-C(-c4:c(:c:c:c:c:4)-C)-N-2-C-c5:c(:c:c:c:c:5)-C)-C)-N)-C)-C-c6:c:c:c:c:c:6 (SP_1_53_M_0) |
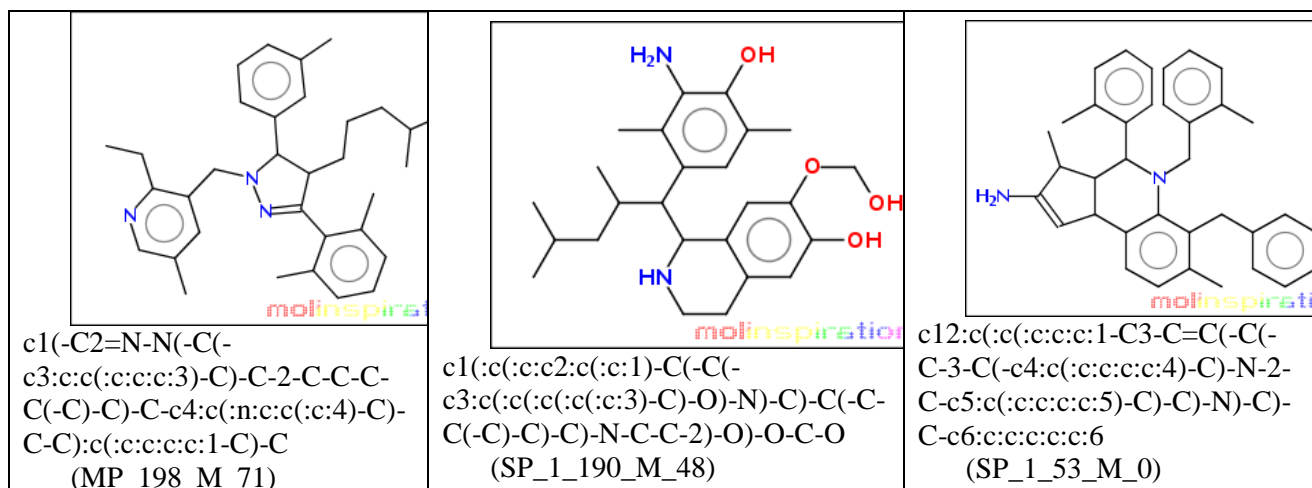
Table 8

Summary table for the experiments held on the two cores CPU.

		MEGA	PMEGA 2 Subpopulations	PMEGA 4 Subpopulations
1st Experiment (population 100, iterations 200)	Average Execution Time	0:39:15	0:22:44	0:25:50
	Speedup		1.727206062	1.519243174
	Efficiency		0.863603031	0.759621587
2nd Experiment (population 150, iterations 200)	Average Execution Time	1:04:44	0:34:48	0:40:33
	Speedup		1.859696728	1.596464785
	Efficiency		0.929848364	0.798232392
3rd Experiment (population 200, iterations 200)	Average Execution Time	1:22:20	0:49:22	0:54:22
	Speedup		1.667604366	1.514253602
	Efficiency		0.833802183	0.757126801
Summary	Average Speedup		1.8	
	Average Efficiency		0.9	

5.2 MEGA versus PMEGA on 4 core CPU

This section summarizes the experimental results from the execution of MEGA versus the process-enabled version of PMEGA using a four core CPU machine.

The specification of our testing machine is:

➢ Machine 2:

▪ CPU: Inter Core 2 Duo Quad Q6600 @ 2.4 GHz (4 Physical Cores)

▪ Memory: 4 Gbytes

The experiments in this section follow the same pattern as the previous experiments. The population sets used was 100, 150 and 200 individuals. The algorithms run for 200 iterations and 5 runs per experiment. The main purpose of these experiments was to compare the gained speedup with four cores with the one gained using 2 cores, also to check the effect of smaller subpopulations on the resulting solutions. These experiments have the double subpopulation number, than before, resulting into smaller size subpopulations.

5.2.1 1st Experiment – MEGA vs. PMEGA (4 Subpopulations) vs. PMEGA (8 Subpopulations)

The first experiment contacted on the four core CPU had a population of 100 distributed into four and eight subpopulations, 200 iterations and executed for 5 runs.

As on previous experiments, the objectives and datasets are the same used at the experiment described in section 5.1.2.

Table 9 shows the execution time of the experiment. As on the previous experiment, Table 9 is showing, the time results of PMEGA using 4 subpopulations with 25 individuals and with 8 subpopulations having 12-13 individuals.

Figure 28 is a graphical demonstration of the solutions of a run of the MEGA algorithm. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red spots represent the individuals of the working populations over a period of iterations.

Figure 29 is a graphical demonstration of the solutions of a run of the PMEGA with 4 Subpopulations algorithm on the same problem.

Figure 30 is a graphical demonstration of the solutions of a run of the PMEGA with 8 Subpopulations algorithm on the same problem.

Figure 31 compares all Pareto fronts of the MEGA and PMEGA algorithms from all runs performed.

The graphs show a very nice distribution of the proposed solutions regardless the algorithm used. From graph in Figure 31 is easy to notice that the solutions have very similar diversity across algorithms, both in parameter and objective space.
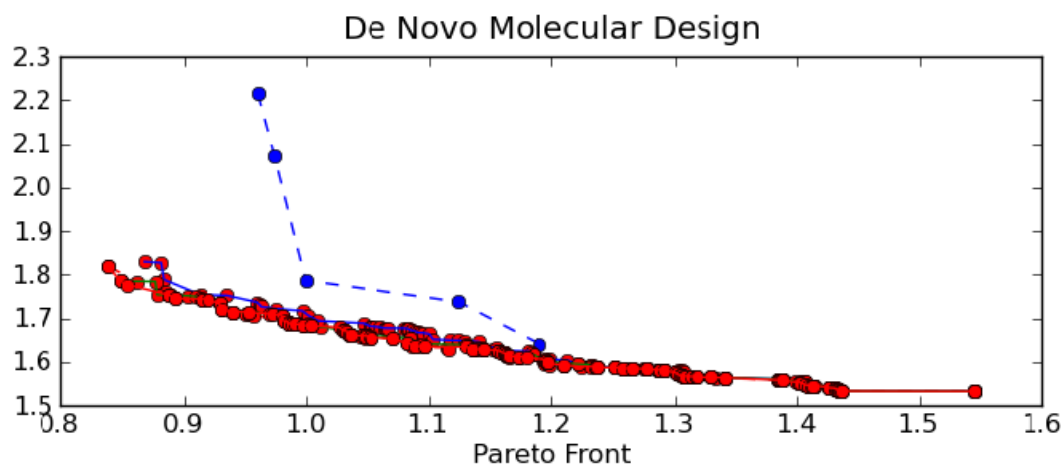

Figure 28: A graph of Pareto fronts taken from one of the runs for MEGA. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working populations over a period of iterations. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
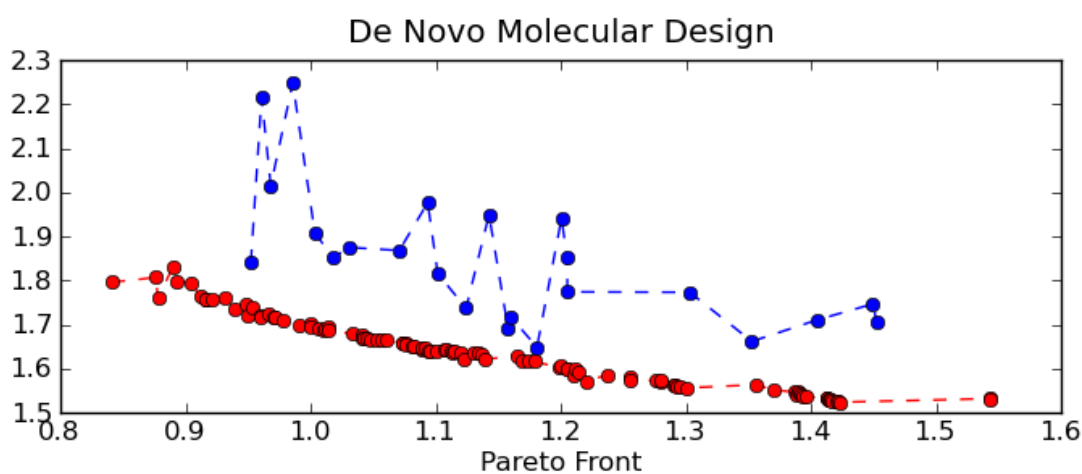
Figure 29: A graph of Pareto fronts taken from one of the runs for PMEGA with 4 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
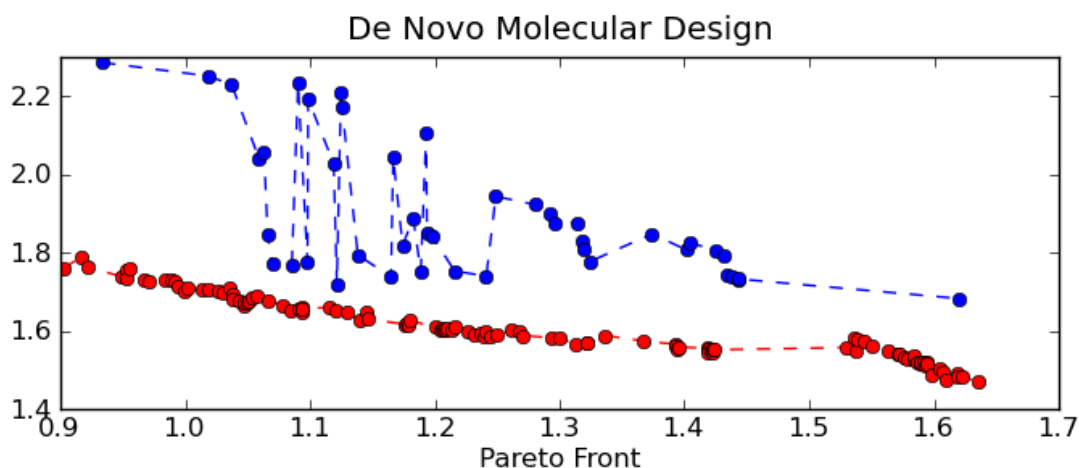
Figure 30: A graph of Pareto fronts taken from one of the runs for PMEGA with 8 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
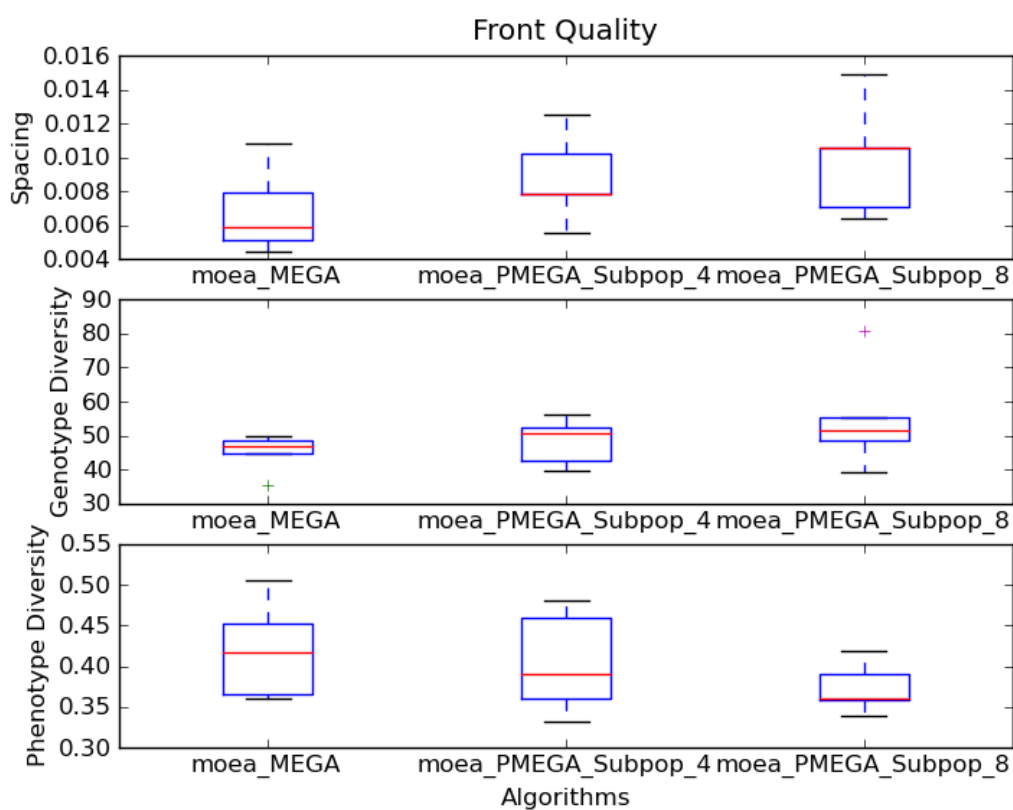
Figure 31: A comparison of MEGA, PMEGA with 4 Subpopulations and PMEGA with 8 Subpopulations Pareto fronts.

Table 9
Time results from the experiment, calculations for Average execution time, Speedup and Efficiency.

	MEGA	PMEGA 4 Subpop	PMEGA 8 Subpop
Run 0	00:57:29	00:19:37	00:21:50
Run 1	00:55:22	00:19:09	00:18:41
Run 2	00:57:26	00:23:06	00:22:25
Run 3	00:53:18	00:22:44	00:21:58
Run 4	00:55:02	00:20:19	00:20:32
Total	04:38:37	01:44:55	01:45:26
Max	00:57:29	00:23:06	00:22:25
Min	00:53:18	00:19:09	00:18:41
Average	00:55:57	00:20:53	00:21:27
Speedup		2.678191489	2.60880829
Efficiency		0.669547872	0.652202073

Time measured is Wall Clock Time (Total Execution Time).

Time format is HH:MM:SS.

Average is calculated as (Total – Max – Min) / (5 - 2).

Speedup is calculated as Time of serial / Time of parallel.

Efficiency is calculated as Speedup / Number of processes.

5.2.2   2nd Experiment – MEGA vs. PMEGA (4 Subpopulations) vs. PMEGA (8 Subpopulations)

The second experiment contacted on the four core CPU had a population of 150 distributed into four and eight subpopulations, 200 iterations and executed for 5 runs.

As on previous experiments, the objectives and datasets are the same used at the experiment described in section 5.1.2.

Table 10 shows the execution time of the experiment. As on the previous experiment, Table 10 is showing, the time results of PMEGA using 4 subpopulations with 37-38 individuals and with 8 subpopulations having 18-19 individuals.

Figure 32 is a graphical demonstration of the solutions of a run of the MEGA algorithm. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red spots represent the individuals of the working populations over a period of iterations.

Figure 33 is a graphical demonstration of the solutions of a run of the PMEGA with 4 Subpopulations algorithm on the same problem.

Figure 34 is a graphical demonstration of the solutions of a run of the PMEGA with 8 Subpopulations algorithm on the same problem.

Figure 35 compares all Pareto fronts of the MEGA and PMEGA algorithms from all runs performed.

The graphs show a very nice distribution of the proposed solutions regardless the algorithm used. Also from graph in Figure 35 is easy to notice that the solutions have very similar diversity across algorithms, both in parameter and objective space.
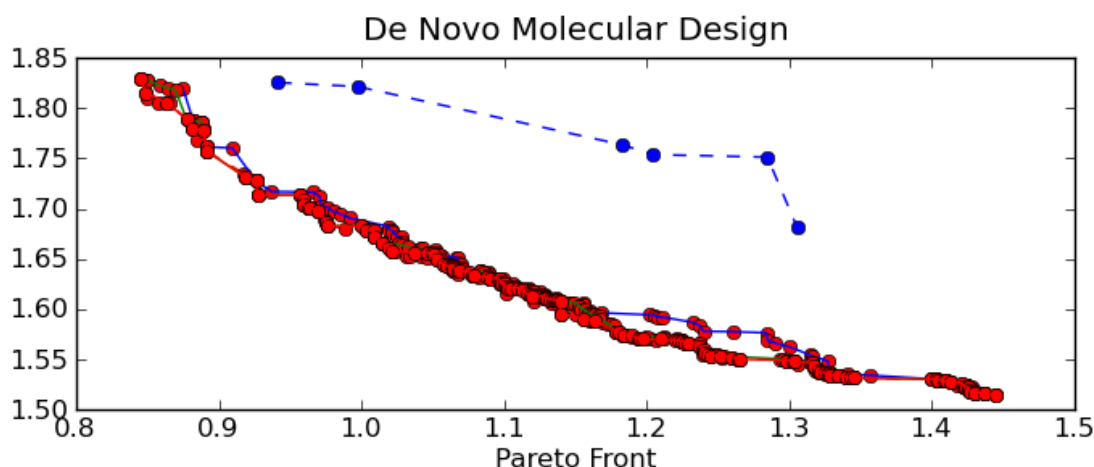
Figure 32: A graph of Pareto fronts taken from one of the runs for MEGA. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working populations over a period of iterations. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
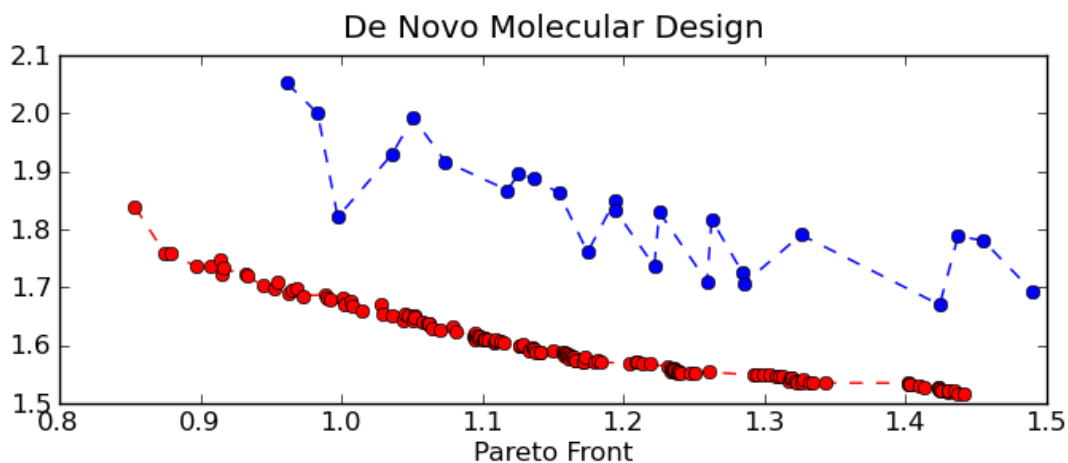
Figure 33: A graph of Pareto fronts taken from one of the runs for PMEGA with 4 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
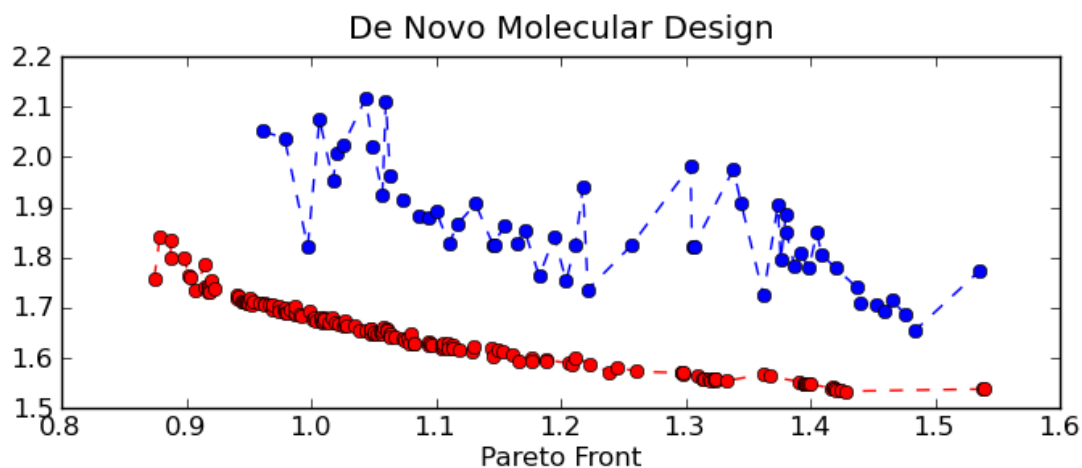
Figure 34: A graph of Pareto fronts taken from one of the runs for PMEGA with 8 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
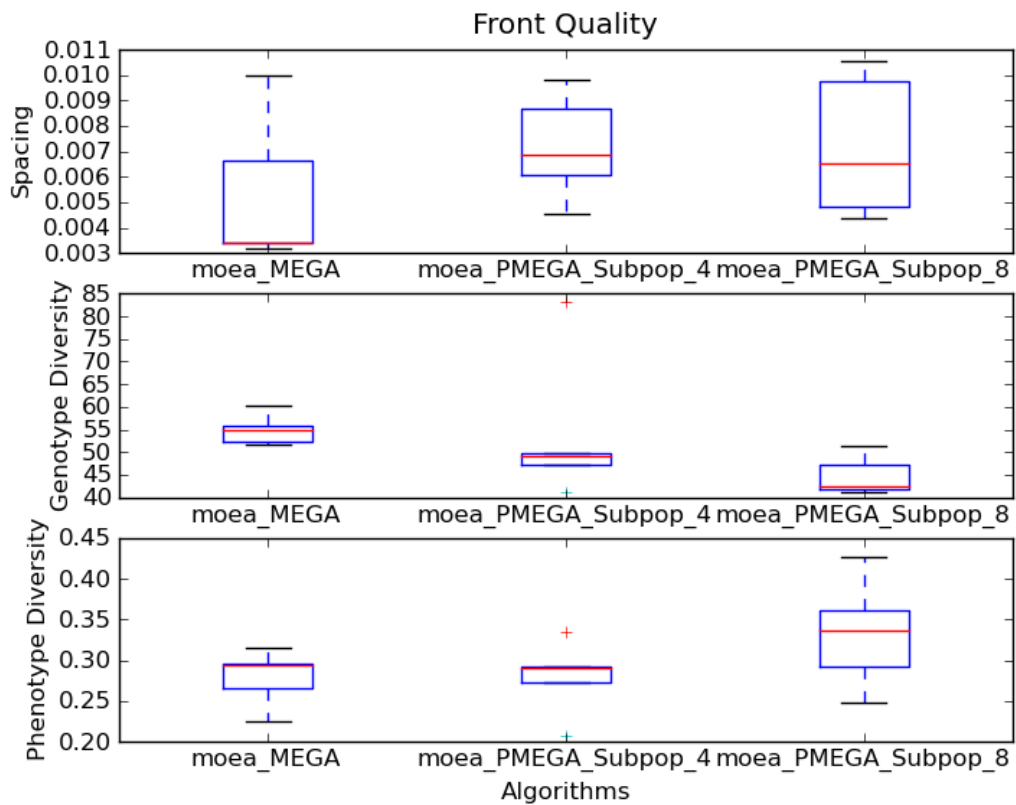
Figure 35: A comparison of MEGA, PMEGA with 4 Subpopulations and PMEGA with 8 Subpopulations Pareto fronts.

Table 10

Time results from the experiment, calculations for Average execution time, Speedup and Efficiency.

	MEGA	PMEGA 4 Subpop	PMEGA 8 Subpop
Run 0	01:26:52	00:30:41	00:32:48
Run 1	01:22:54	00:31:49	00:30:25
Run 2	01:26:33	00:39:17	00:34:03
Run 3	01:27:47	00:32:20	00:33:20
Run 4	01:25:28	00:31:57	00:30:29
Total	07:09:34	02:46:04	02:41:05
Max	01:27:47	00:39:17	00:34:03
Min	01:22:54	00:30:41	00:30:25
Average	01:26:18	00:32:02	00:32:12
Speedup		2.693895248	2.679489391
Efficiency		0.673473812	0.669872348

Time measured is Wall Clock Time (Total Execution Time).

Time format is HH:MM:SS.

Average is calculated as (Total – Max – Min) / (5 - 2).

Speedup is calculated as Time of serial / Time of parallel.

Efficiency is calculated as Speedup / Number of processes.

5.2.3   3rd Experiment – MEGA vs. PMEGA (4 Subpopulations) vs. PMEGA (8 Subpopulations)

The 3rd, and last, experiment contacted on the four core CPU had a population of 200 distributed into four and eight subpopulations, 200 iterations and executed for 5 runs.

As on previous experiments, the objectives and datasets are the same used at the experiment described in section 5.1.2.

Table 11 shows the execution time of the experiment. As on the previous experiment, Table 11 is showing, the time results of PMEGA using 4 subpopulations with 50 individuals and with 8 subpopulations having 25 individuals.
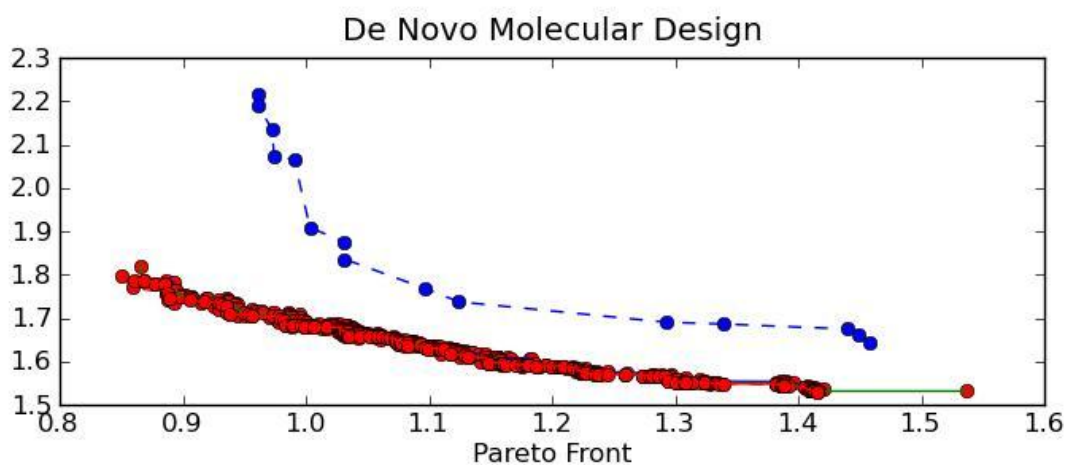
Figure 36 is a graphical demonstration of the solutions of a run of the MEGA algorithm. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red spots represent the individuals of the working populations over a period of iterations.

Figure 37 is a graphical demonstration of the solutions of a run of the PMEGA with 4 Subpopulations algorithm on the same problem.

Figure 38 is a graphical demonstration of the solutions of a run of the PMEGA with 8 Subpopulations algorithm on the same problem.

Figure 39 compares all Pareto fronts of the MEGA and PMEGA algorithms from all runs performed.

The graphs show a very nice distribution of the proposed solutions regardless the algorithm used. Also from graph in Figure 39 is easy to notice that the solutions have very similar diversity across algorithms, both in parameter and objective space. This is very nice since is cross validated in all three experiments of section 5.2.


Figure 36: A graph of Pareto fronts taken from one of the runs for MEGA. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working populations over a period of iterations. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
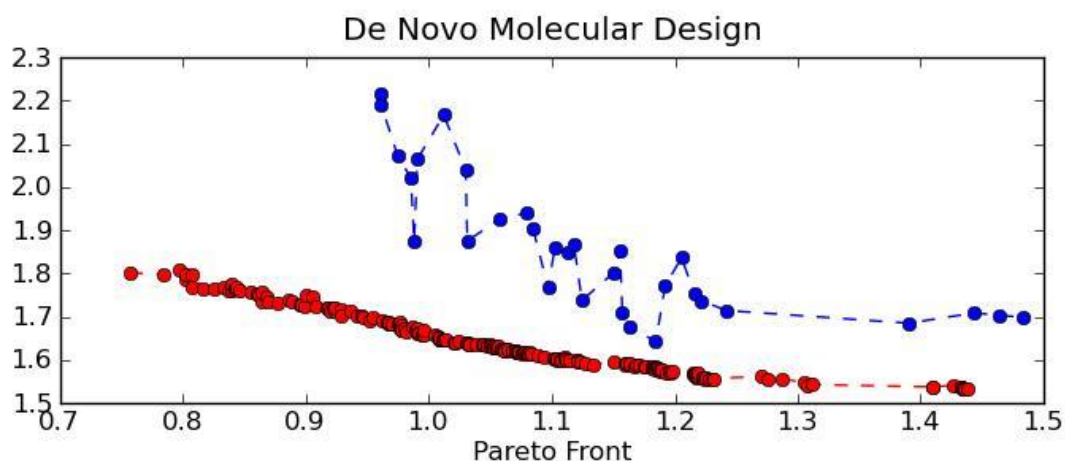
Figure 37: A graph of Pareto fronts taken from one of the runs for PMEGA with 4 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
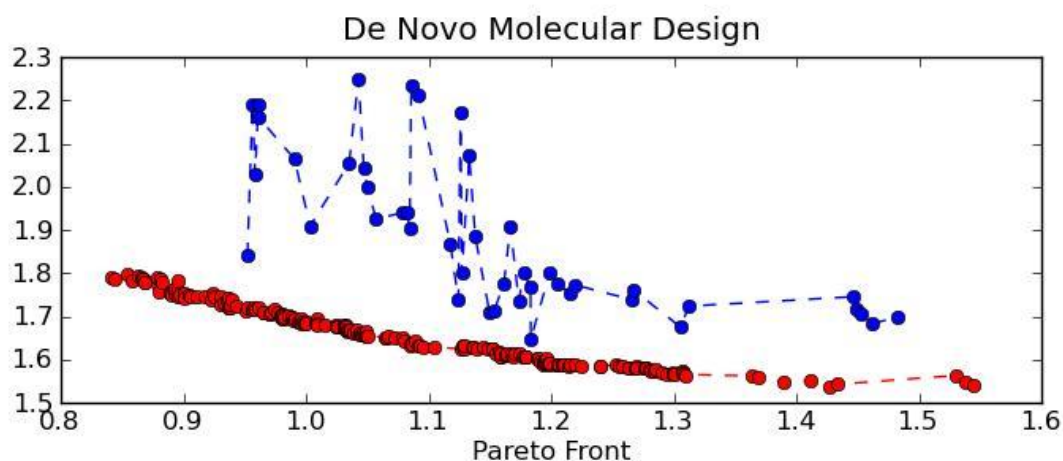
Figure 38: A graph of Pareto fronts taken from one of the runs for PMEGA with 8 subpopulations. The X and Y axis represent the two objectives. Note that both objectives need to be minimized. The blue dots on the blue dashed line represent the individuals of the starting working population. The red dots represent the individuals of the Pareto Fronts over a period of iterations.
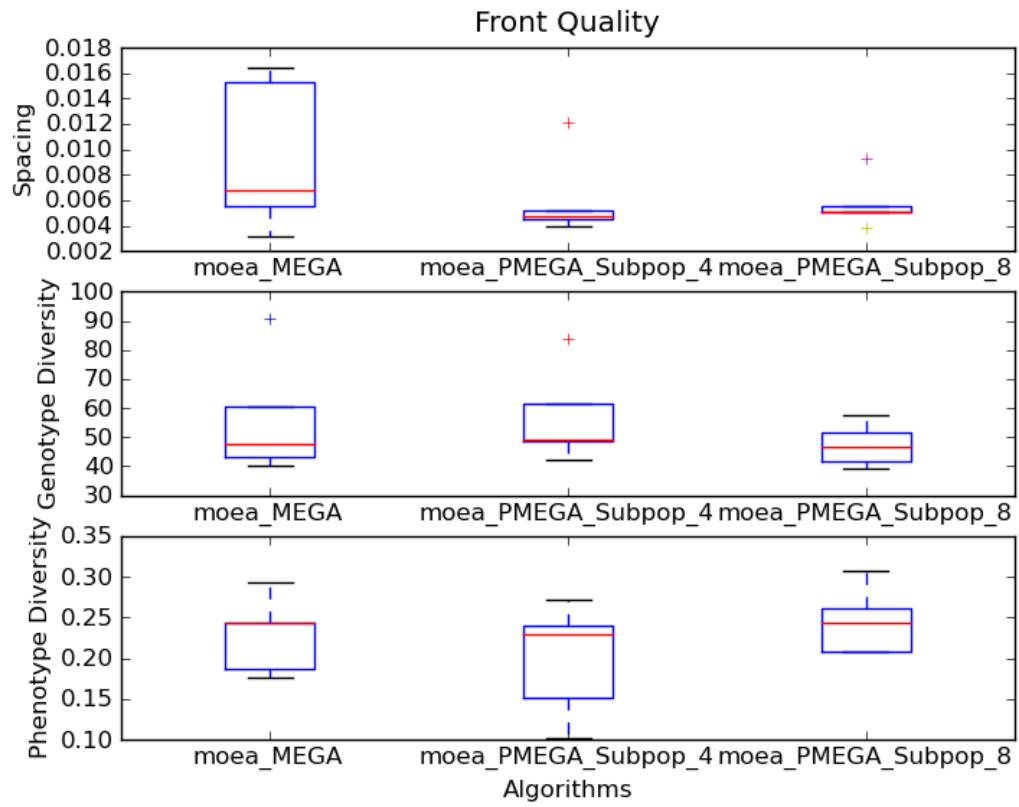
Figure 39: A comparison of MEGA, PMEGA with 4 Subpopulations and PMEGA with 8 Subpopulations Pareto fronts.

Table 11
Time results from the experiment, calculations for Average execution time, Speedup and Efficiency.

	MEGA	PMEGA 4 Subpop	PMEGA 8 Subpop
Run 0	02:00:38	00:45:49	00:42:13
Run 1	01:50:20	00:35:38	00:42:17
Run 2	01:55:42	00:48:17	00:42:43
Run 3	01:57:31	00:42:06	00:40:45
Run 4	01:57:03	00:40:02	00:40:22
Total	09:41:14	03:31:52	03:28:20
Max	02:00:38	00:48:17	00:42:43
Min	01:50:20	00:35:38	00:40:22
Average	01:56:45	00:42:39	00:41:45
Speedup		2.73752768	2.796540253
Efficiency		0.68438192	0.699135063

Time measured is Wall Clock Time (Total Execution Time).

Time format is HH:MM:SS.

Average is calculated as (Total – Max – Min) / (5 - 2).

Speedup is calculated as Time of serial / Time of parallel.

Efficiency is calculated as Speedup / Number of processes.

5.2.4   Summary

This section summarizes the results from the experiments that were executed on the four cores CPU. A summary of the quality of the solutions is shown, comparing a random sample of the proposed solutions with the requested objectives. Also a summary of the gained speedup is shown.

The graphs displayed in the experiments sections show a general trend of the solutions, but choosing a random sample of the proposed solutions and comparing them with the

requested objectives gives a more detailed view of how well the algorithms work. Molecules 2D representations were designed by the online tools of MolInspiration [MOLINSP].

Table 12 shows the ligands that are known to be active with ER-a, which are used for the similarity objective. Table 13 shows the ligands that are known to be active with ER-b, which are used for the dissimilarity objective.

Table 14 shows a random sample of the algorithm's proposed compounds. By just a look at the compounds displayed in this table is easy to find similar compounds in different algorithms. This shows that MEGA and PMEGA can produce similar solutions.

Table 15 shows a summary of the experiments executed on the four cores CPU. From measured speedup on all the experiments the average speedup is calculated to be used with Amdahl's Law for the calculation of the maximum expected speedup of the parallel algorithm. The average measured speedup for this set of experiments is 2.7 and the efficiency 0.6.

Table 12

Compounds that algorithm's resulting compounds must be similar to. These compounds are known to be active with ER-a.
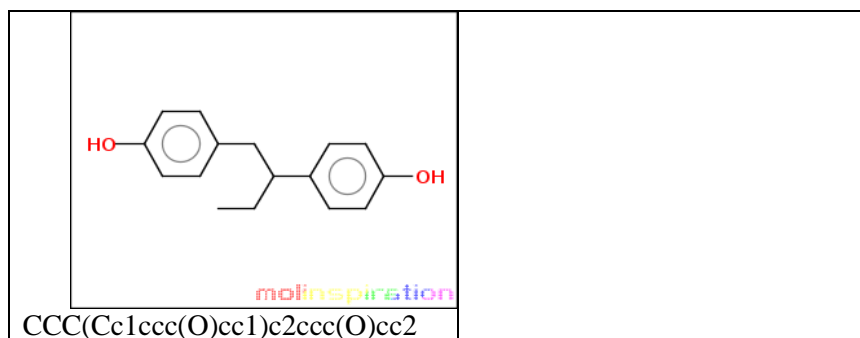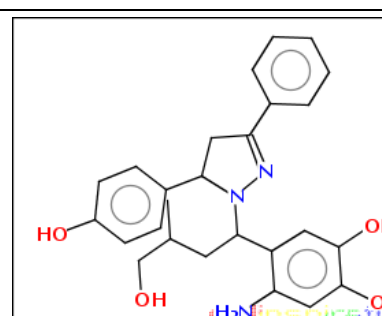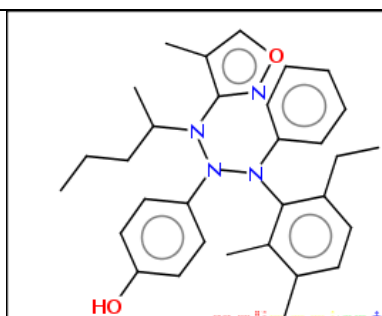

| Oc3ccc(c2oc(=O)c1cc(O)ccc1c2P)cc3 | CCCc3c(c1ccc(O)cc1)nn(c2ccc(O)cc2)c3c4ccc(O)cc4NO |
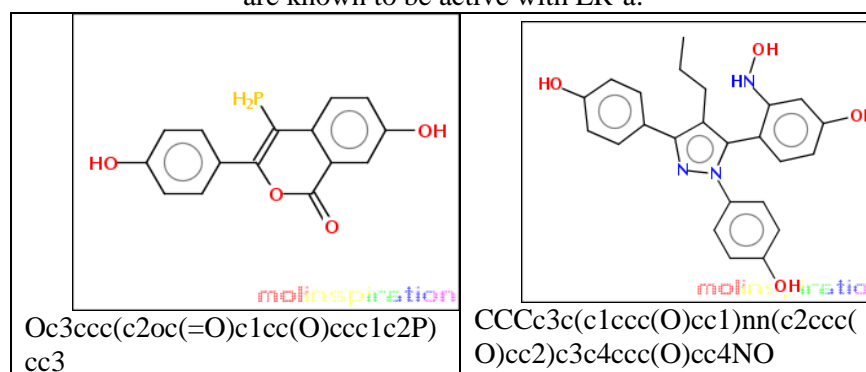
Table 13

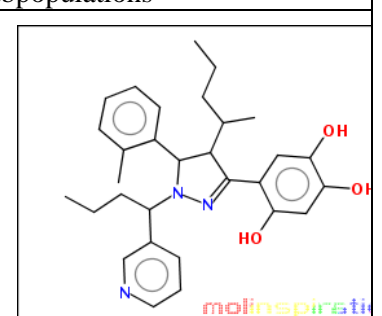Compounds that algorithm's resulting compounds must not be similar to. These compounds are known to be active with ER-b.

| Oc3cccc(c2coc1cc(O)cc(O)c1c2=O)c3 | CC[C@H]2Cc1cc(O)ccc1C4=C2c3ccc(O)cc3C[C@@H]4CC |

CCC(Cc1ccc(O)cc1)c2ccc(O)cc2
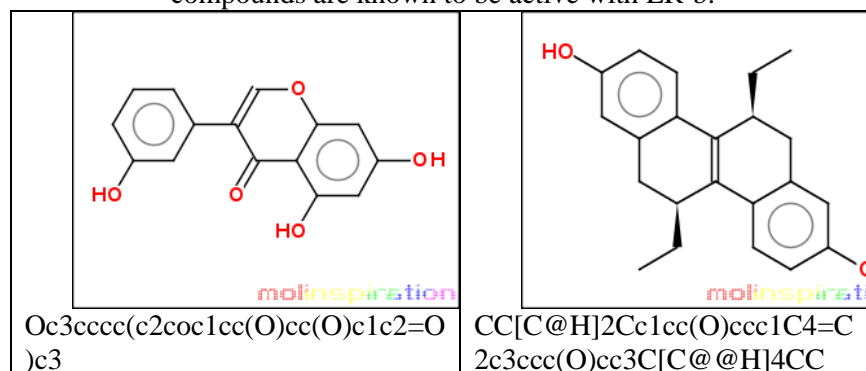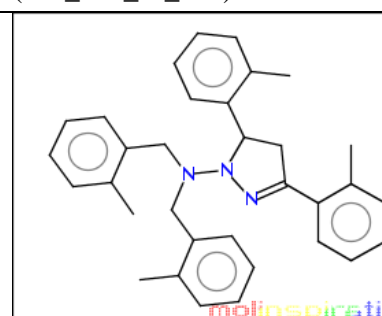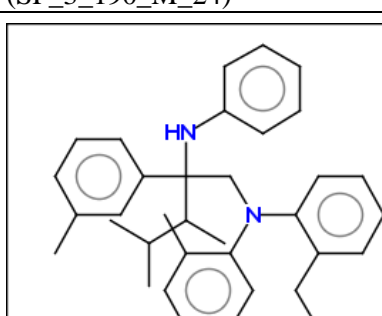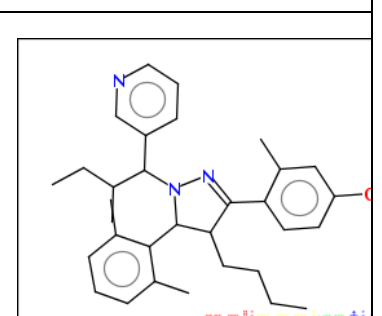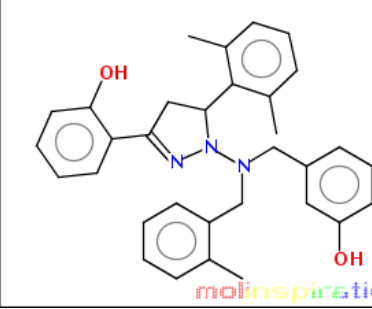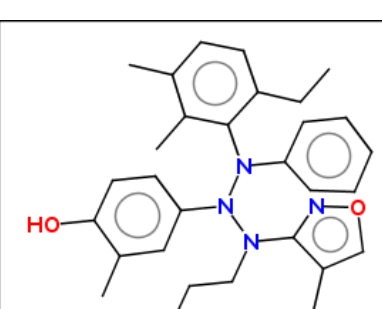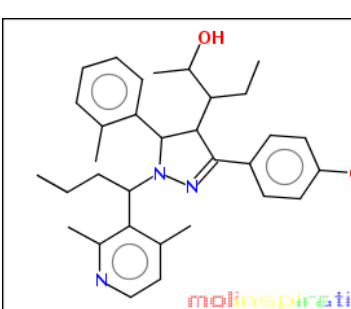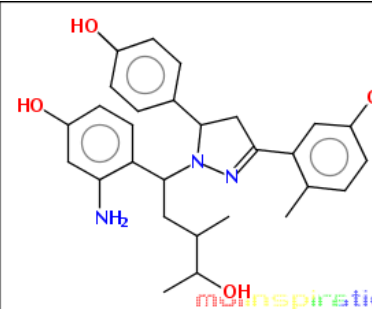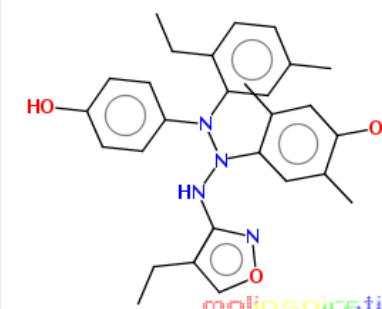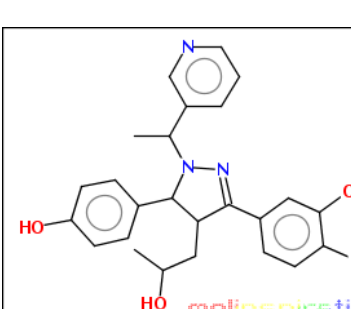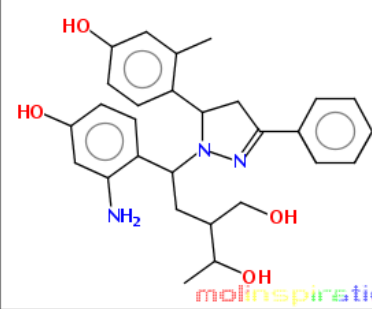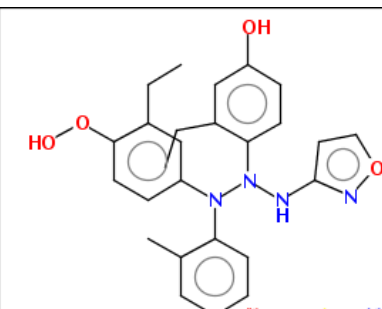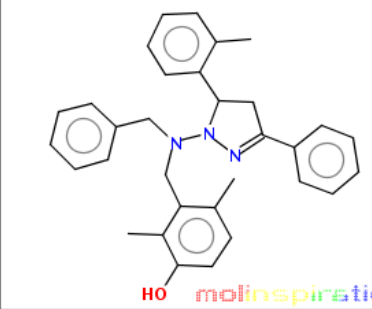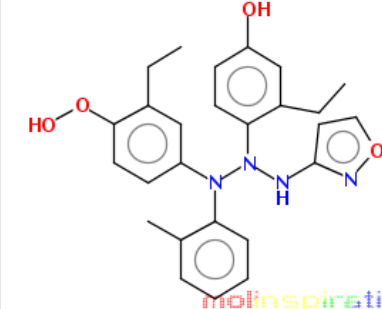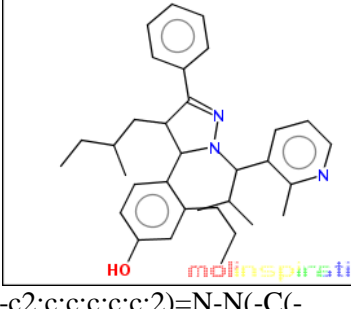
Table 14
Random sample of proposed solutions taken from MEGA and PMEGA.

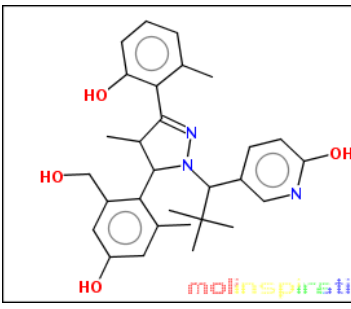| MEGA	PMEGA with 4 Subpopulations	PMEGA with 8 Subpopulations
c1(-C2=N-N(-C(-c3:c(:c:c(:c(:c:3)-O)-O)-N)-C-C(-C-O)-C)-C(-c4:c:c:c(:c:c:4)-O)-C-2):c:c:c:c:c:1 (MP_174_M_183)	c1(:c(:c(:c:c:c:1-C-C)-C)-C)-N(-c2:c:c:c:c:c:2)-N(-c3:c:c:c(:c:c:3)-O)-N(-c4:c(:c:o:n:4)-C)-C(-C-C-C)-C (SP_3_190_M_24)	c1(-C2=N-N(-C(-c3:c(:c:c:c:c:3)-C)-C-2-C(-C-C-C)-C)-C(-c4:c:n:c:c:c:4)-C-C-C):c(:c:c(:c(:c:1)-O)-O)-O (SP_2_147_M_6)
c1(-C2=N-N(-N(-C-c3:c(:c:c:c:c:3)-C)-C-c4:c(:c:c:c:c:4)-C)-C(-c5:c(:c:c:c:c:5)-C)-C-2):c(:c:c:c:c:1)-C (MP_161_M_191)	c1(:c(:c:c:c:c:1)-C-C)-N(-c2:c(:c:c:c:c:2)-C)-C-C(-c3:c:c(:c:c:c:3)-C)(-C-C(-C(-C)-C)-C)-N-c4:c:c:c:c:c:4 (SP_2_164_M_13)	c1(-C2=N-N(-C(-c3:c(:c:c:c:c:3-C)-C)-C-2-C-C-C-C)-C(-c4:c:n:c:c:c:4)-C(-C-C)-C):c(:c:c(:c:c:1)-O)-C (SP_1_174_M_18)

c1(-C2=N-N(-N(-C-
c3:c(:c:c:c:c:3)-C)-C-
c4:c:c(:c:c:c:4)-O)-C(-
c5:c(:c:c:c:c:5-C)-C)-C-
2):c(:c:c:c:c:1)-O
(MP_192_M_8)

c1(:c(:c(:c:c:c:1-C-C)-C)-C)-N(-
c2:c:c:c:c:c:2)-N(-c3:c:c(:c(:c:c:3)-
O)-C)-N(-c4:c(:c:o:n:4)-C)-C-C-C
(SP_1_172_M_27)

c1(:c(:n:c:c:c:1-C)-C)-C(-N2-
N=C(-c3:c:c:c(:c:c:3)-O)-C(-C-
2-c4:c(:c:c:c:c:4)-C)-C(-C(-O)-
C)-C-C)-C-C-C
(SP_3_173_M_6)

c1(-C2=N-N(-C(-c3:c(:c:c(:c:c:3)-
O)-N)-C-C(-C(-O)-C)-C)-C(-
c4:c:c:c(:c:c:4)-O)-C-
2):c(:c:c:c(:c:1)-O)-C
(MP_162_M_119)

c1(:c(:c:c:c(:c:1)-C)-C-C)-N(-
c2:c:c:c(:c:c:2)-O)-N(-
c3:c(:c:c(:c:c:3)-C)-O)-C)-N-
c4:c(:c:o:n:4)-C-C
(SP_0_160_M_15)

C1(-c2:c:c(:c(:c:c:2)-O)-O)=N-
N(-C(-c3:c:c:c(:c:c:3)-O)-C-1-
C-C(-O)-C)-C(-c4:c:n:c:c:c:4)-C
(SP_5_173_M_19)

c1(-C2=N-N(-C(-c3:c(:c:c(:c:c:3)-
O)-C)-C-2)-C(-c4:c(:c:c(:c:c:4)-
O)-N)-C-C(-C(-O)-C)-C-
O):c:c:c:c:c:1
(MP_132_M_74)

c1(:c(:c:c:c:c:1)-C)-N(-c2:c:c(:c(-O-
O):c:c:2)-C-C)-N(-c3:c(:c:c(:c:c:3)-
O)-C-C)-N-c4:n:o:c:c:4
(SP_3_177_M_2)

c1(-C2=N-N(-C(-
c3:c:n:c(:c:c:3)-O)-C(-C)(-C)-
C)-C(-c4:c(:c:c(:c:c:4-C)-O)-C-
O)-C-2-C):c(:c:c:c:c:1-C)-O
(SP_7_199_M_1)

c1(:c(:c(:c:c:c:1-C)-O)-C)-C-N(-

c1(:c(:c:c:c:c:1)-C)-N(-c2:c:c(:c(-O-

C1(-c2:c:c:c:c:c:2)=N-N(-C(-
c3:c(:c:c(:c:c:3)-O)-C-C-C)-C-

N2-N=C(-c3:c:c:c:c:c:3)-C-C-2-c4:c(:c:c:c:c:4)-C)-C-c5:c:c:c:c:c:5 (MP_187_M_100)	O):c:c:2)-C-C)-N(-c3:c(:c:c(:c:c:3)-O)-C-C)-N-c4:n:o:c:c:4 (SP_3_177_M_2)	1-C-C(-C-C)-C)-C(-c4:c(:n:c:c:c:4)-C)-C(-C)-C (SP_1_166_M_0)

Table 15
Summary table for the experiments held on the four cores CPU.

		MEGA	PMEGA 4 Subpopulations	PMEGA 8 Subpopulations
1st Experiment (population 100, iterations 200)	Average Execution Time	0:55:57	0:20:53	0:21:27
	Speedup		2.678191489	2.60880829
	Efficiency		0.669547872	0.652202073
2nd Experiment (population 150, iterations 200)	Average Execution Time	1:26:18	0:32:02	0:32:12
	Speedup		2.693895248	2.679489391
	Efficiency		0.673473812	0.669872348
3rd Experiment (population 200, iterations 200)	Average Execution Time	1:56:45	0:42:39	0:41:45
	Speedup		2.73752768	2.796540253
	Efficiency		0.68438192	0.699135063
Summary	Average Speedup		2.7	
	Average Efficiency		0.7	

Chapter 6

Discussion

From the collected and calculated results about the measured speedup of the experiments we can use the Amdahl's Law to calculate the estimated percentage of the parallelism on the algorithm we use and further on the estimated speedup using more CPUs.

The average Speedup measured is $S_{avg} = 1.8$.

Using $P_{estimated} = \dfrac{\frac{1}{S}-1}{\frac{1}{N}-1}$ with $S = 1.8$ and $N = 2$ the $P_{estimated}$ equals to 0.86, meaning that 86% of the algorithm is executed in parallel.

Using the $P_{estimated}$ in Amdahl's Law we can estimate the speedup of the algorithm using more CPUs.

Using Amdahl's Law for parallelization $\dfrac{1}{(1-P)+\frac{P}{N}}$, replacing P with $P_{estimated}$ calculated before and N with 4, 8, 16, 32, etc, shown in Table 16. Table 17 acts as a cross-reference to Table 16, using as reference the four cores CPU.

Table 16

Calculate Estimated Speedup and Efficiency using more cores. Reference used is the two cores CPU.

N	$S_{estimated}$	E
2	1.8	0.9
4	2.8	0.7
8	4.0	0.5
16	5.1	0.3
32	5.9	0.2
64	6.4	0.1

$P_{estimated}$	$P_{estimated} = \dfrac{\frac{1}{S}-1}{\frac{1}{N}-1}$ $\rightarrow$ 0.86	Estimated parallel percentage of the algorithm.
$S_{estimated}$	$\dfrac{1}{(1-P)+\frac{P}{N}}$	Estimated Speedup
N		Number of cores
E		Efficiency
S_{max}	$\dfrac{1}{(1-P)}$	7.0

Table 17

Calculate Estimated Speedup and Efficiency using more cores. Reference used is the four cores CPU.

N	$S_{estimated}$	E
4	2.7	0.7
8	3.8	0.5
16	4.7	0.3
32	5.4	0.2
64	5.8	0.1

$P_{estimated}$	$P_{estimated} = \dfrac{\frac{1}{S}-1}{\frac{1}{N}-1}$ $\rightarrow$ 0.84	Estimated parallel percentage of the algorithm.
$S_{estimated}$	$\dfrac{1}{(1-P)+\frac{P}{N}}$	Estimated Speedup
N		Number of cores
E		Efficiency
S_{max}	$\dfrac{1}{(1-P)}$	6.3

From these estimations we can conclude that the algorithm can have a maximum Speedup around 6.6 (average maximum Speedup). Something very important to notice is the Efficiency

shown in Tables 16 and 17, as we add more cores the efficiency drops to very low numbers since the algorithm gets allocated more resources than those it really needs.

From all these we conclude to the fact that using more than eight cores to run the current algorithm (PMEGA) is inefficiently due to low efficiency and small speedup from that point and further on. This is also supported by Amdahl's Law, which states that if an algorithm can be parallelized at a degree of 90% then the maximum speedup it can achieve regardless of the number of CPUs is 10, shown in figure 9. And in this case the estimated parallelized part of the algorithm is 85% so by applying Amdahl's Law and setting the number of CPUs to infinite, then the maximum estimated speedup is 6.6.

When comparing PMEGA with itself, using different number of subpopulations at the same problem, a minor reduction to the speedup gained is observed. This happens due to the fact that after the independent evolution o a subpopulation, the child process responsible for the evolution reports the results back to the main process, which in its turn gathers them and merges them with other results that were delivered. This action of gathering and merging requires some computation for the main process, thus delaying for a small amount of time the child process to get assigned a new task.

Regarding the quality of the solutions of PMEGA compared with MEGA, from the visualization of the sample of proposed solutions and the graphs showing the quality of the solutions for each experiment, was clearly shown that:

- PMEGA can provide similar solutions with MEGA.
- PMEGA's subpopulations can provide solutions that are different among subpopulations. Despite the fact that subpopulations were selected at random without the use of any knowledge.

Regarding the programming experience, Python helps the programmer by having an already implemented class for creating a pool of processes and a well defined API. Also gives the programmer the freedom to make his/her own experiments with the various classes and components provided. Despite the fact that Python limits the use of real multithreading with the use of GIL, there are ways to overcome this obstacle by the use of multiprocessing.

Chapter 7

Concluding Remarks and Future Work

The experimental results produced lead to the following conclusions:

- With respect to the quality of the solutions produced, MEGA and PMEGA behave comparably. The differences observed between the final Pareto-front approximations produced, are partly due to the way PMEGA splits the working population into subpopulations. The current PMEGA implementation splits the population in a random fashion without using any knowledge related to the morphology of the Pareto-approximation and the density of solutions at any region of the search space. A potentially better way to split the population we are planning to explore will use clustering methodologies.

- From the experiments and the use of Amdahl's Law is clearly shown that the PMEGA can achieve a maximum speedup of 6.6. This does not seems good if we consider that is achieved when the number of CPUs used reaches infinite. But on the other hand is stated clearly that when using a two cores CPU, PMEGA completes in the half time that MEGA needs and when using a four cores CPU, PMEGA completes almost in one third of the time that MEGA needs.

- Number of subpopulations has an effect on the quality of the results obtained. With more subpopulations the more uniform the resulting solutions are, at least in objective space. In parameter space this has little to no effect since similar solutions are located in the same cluster group, thus the more similar solutions does not change the diversity of the cluster groups, it only changes the size of the cluster groups.

The above conclusions show that using PMEGA, a modified PEA version of MEGA, can provide us with equivalent solution sets in substantially less time.

Future work on PMEGA will focus on:

- Algorithmic improvements in the way subpopulations are selected with the aid of knowledge-driven approaches in order to improve the quality of the optimization search and reduce the number of iterations needed for convergence.

- Also an investigation if further parallelization is possible in order to achieve a better maximum speedup.

Bibliography

[1] M. Tomassini, Parallel and Distributed Evolutionary Algorithms: A Review, Institute of Computer Science, University of Lausanne.

[2] Bäck T., *Evolutionary Algorithms in Theory and Practice.* Oxford University Press 1996

[3] http://en.wikipedia.org/wiki/Optimization_%28mathematics%29 (accessed December 18, 2009)

[4] http://en.wikipedia.org/wiki/Charles_Darwin (accessed December 1, 2009)

[5] Darwin, C. R. 1876, *The origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*. London: John Murray. 6th ed, with additions and corrections.

[6] http://anthro.palomar.edu/synthetic/default.htm (accessed December 1, 2009)

[7] C. A. Nicolaou, J. Apostolakis,, and C. S. Pattichis, "De Novo Drug Design Using Multiobjective Evolutionary Graphs", J. Chem. Inf. Model., vol. 49(2), pp. 295-307, 2009.

[8] Baringhaus, K.-H.; Matter, H. Efficient strategies for lead optimization by simultaneously addressing affinity, selectivity and pharmacokinetic parameters. In Chemoinformatics in Drug DiscoVery; Oprea, T., Ed.; Wiley-VCH: Weinheim, Germany, 2004; pp 333-379.

[9] Schneider, G.; Fechner, U. Computer-based de novo design of druglike molecules. Nat. ReV. Drug DiscoVery 2005, 4, 649–663.

[10]   Xu, J.; Hagler, A. Chemoinformatics and drug discovery. Molecules 2002, 7, 566–600.

[11]   Bohacek, R. S.; Martin, C.; Guida, W. C. The Art and Practice of Structure-Based Drug Design: A Molecular Modelling Approach. Med. Res. ReV. 1996, 16, 3–50.

[12]   Evolutionary Algorithms 8 Population models – Parallel Implementations, http://www.geatbx.com/docu/algindex-07.html (accessed August 20, 2009).

[13]   P. Adamidis, Parallel Evolutionary algorithms: A Review, Dept. of Applied Informatics, University of Macedonia.

[14]   http://en.wikipedia.org/wiki/Speedup (accessed December 1, 2009).

[15]   http://en.wikipedia.org/wiki/Gene_Amdahl (accessed December 1, 2009).

[16]   http://en.wikipedia.org/wiki/ Amdahl's_law (accessed December 1, 2009).

[17]   Fonseca, C. M.; Fleming, P. J. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In Proceedings of the Fifth International Conference on Genetic Algorithms; Forrest, S., Ed.; Morgan Kaufmann: San Mateo, CA, 1993; pp 416-423.

[18]   Y. Colette, P. Siarry, Multiobjective Optimization: Principles and Case Studies; Springer-Verlag: Berlin, Germany, 2004.

[19]   http://www.python.org (accessed August 20, 2009).

[20]   http://www.parallelpython.com (accessed September 23, 2009).
[21]   http://code.google.com/p/mpi4py (accessed September 23, 2009).
[22]   http://en.wikipedia.org/wiki/CUDA (accessed September 23, 2009).
[23]   http://en.wikipedia.org/wiki/OpenCL (accessed September 23, 2009).
[24]   http://pypi.python.org/pypi/pycuda (accessed September 23, 2009).
[25]   http://pypi.python.org/pypi/pyopencl (accessed September 23, 2009).
[26]   http://en.wikipedia.org/wiki/Global_Interpreter_Lock (accessed September 23, 2009).
[27]   http://docs.python.org/library/multiprocessing.html (accessed September 23, 2009).
[28]   http://docs.python.org/library/threading.html #module-threading (accessed September 23, 2009).
[29]   J. Noller, R. Oudkerk, PEP 371 -- Addition of the multiprocessing package to the standard library, (accessed Mar 19, 2009).
[30]   http://en.wikipedia.org/wiki/Cluster_analysis (accessed December 16 2009)
[31]   [ITAB09]http://www.cs.ucy.ac.cy/itab2009/
[32]   http://en.wikipedia.org/wiki/Jaccard_index (accessed September 28, 2009).
[33]   Wheeler, D. L.; Barrett, T.; Benson, D. A.; Bryant, S. H.; Canese, K.; Chetvernin, V.; Church, D. M.; DiCuccio, M.; Edgar, R.; Federhen, S.; Geer, L. Y.; Helmberg, W.; Kapustin, Y.; Kenton, D. L.; Khovayko, O.; Lipman, D. J.; Madden, T. L.; Maglott, D. R.; Ostell, J.; Pruitt, K. D.; Schuler, G. D.; Schriml, L. M.; Sequeira, E.; Sherry, S. T.; Sirotkin, K.; Souvorov, A.; Starchenko, G.; Suzek, T. O.; Tatusov, R.; Tatusova, T. A.; Wagner, L.; Yaschenko, E. Database resources of the National Center for Biotechnology Information. Nucleic Acids Res. 2006, 34, D173–D180.
[34]   http://www.noesisinformatics.com (accessed September 28, 2009).
[35]   http://molinspiration.com (accessed December 16 2009)
[36]   Handl, J.; Kell, D. B.; Knowles, J. Multiobjective Optimization in Bioinformatics and Computational Biology. IEEE/ACM Trans. Comput. Biol. Bioinf. 2007, 4, 279–292.
[37]   Alberto, I.; Azcarate, C.; Mallor, F. & Mateo, P.M., Multiobjective Evolutionary Algorithms. Pareto Rankings.
[38]   A. Petrovski, J. McCall, Mult-objective Optimisation of Cancer Chemotherapy Using Evolutionary Algoritms, 2001
[39]   Yong Liang, Kwong-Sak Laung, Tony Shu am Mok, A Novel Evolutionary Drug Scheduling Model in Cancer Cemotherapy, 2004
[40]   H. S. Lopes, Evolutionary Algorithms for the Protein Folding Problem: A Review and Current Trends, 2008
[41]   C. M. V. Benitez, H. S. Lopes, A Parallel Genetic Algorithm for Protein Folding Prediction Using the 3D-HP Side Chain Model, 2009
[42]   E. Alba, Parallel Evolutionary Algorithms can achieve super-linear Performance, University of Malaga.
[43]   N. Matloff, F. Hsu Tutorial on Threads Programming with Python, 2007.
[44]   C. Nicolaou, C. Kannas, C. Pattichis, Optimal graph Design Using A Knowledge-driven Multi-objective Evolutionary Graph Algorithm In Proceedings of the Ninth International Conference on Information Techonology and Applications in Biomedicine, Larnaca Cyprus, November 5 - 7, 2009