An overview of promising evolutionary algorithms

The practical use of Evolutionary Computing

Martin Visser (mjvisser@cs.vu.nl)
Free university, Amsterdam

August 2003

Abstract

This paper intends to give an overview of real-world and academical problems that can be successfully solved by using an evolutionary computing (EC) approach. A bird's eye view is taken to do this whilst hard academical evidence proving the superiority of evolutionary algorithms (EAs) of other methods is still scarce. From this perspective a selection of problems that are currently investigated with evolutionary techniques are described and it is pointed out why EAs can work well on these problems. We conclude with a practical description of what the advantages and disadvantages of EAs are in real-world problems and how their paradigm might be exploited further.

0. Table of contents

1. Introduction & acknowledgement

This paper has been written for the "BWI werkstuk" course at the Free University (VU) in Amsterdam. This course is meant as a literature research in the final year of the study Business Mathematics & Computer Science[1].

After following the course "Evolutionary computing" (EC) taught by Guszti Eiben I became inspired by the way this (r)evolutionary approach tackles large, complex problems. The different viewpoint and the natural interpretation of such algorithms seem to be very flexible and capable of solving problems while remaining simple and elegant as an algorithm. In cooperation with Guszti Eiben the subject of reviewing promising applications of this powerful solver arose.

Of course the list of problems for which evolutionary algorithms (EAs) are possible solvers laid out in this paper is not (at all) exhaustive. It intends only to give an impression of the sheer possibilities of using an evolutionary technique. Furthermore we will not go into implementation issues and specific EC-bound theory to keep the document open for people from outside the field of EC. In this paper we stress the *possibilities* and *practical use* of EC and not the shortcomings or theoretical background of EC.

For people already familiar with the concept of EC the next chapter may be skipped. It sketches the history of EC, the basic concepts in building an EA and discusses the types of problems where to it can be applied. In chapter 3 we determine a scope for chapter 4. In this chapter a categorized list is given of promising application areas. For all of the categories we state the nature of the problem and how EC can be incorporated as a solver to get better solutions.

[1] Bedrijfswiskunde & Informatica (BWI) in dutch

2. Explaining evolutionary algorithms

2.1. A short history

Evolutionary computing (EC) is a relatively new field in the Artificial Intelligence (AI). If we look at its history we see that around the forties and fifties the initial ideas for using an *evolutionary* approach for solving problems where developed. In the sixties these ideas materialized into different algorithmical structures all sharing an evolutionary component. Three different evolutionary streams formed all invented by separate researchers. Not in any specific order these where Evolutionary Programming (EP) by Lawrence Fogel, Genetic Algorithms (GA) by J. Holland and Evolution Strategies (ES) by I. Rechenberg/H. Schewefel. Though debated continuously throughout EC history which algorithm had the most potential as a problem solver, these streams displayed huge resemblances in their way of solving things but all of them had their own typical features and successful problem solving characteristics (see [4]). In the nineties Genetic Programming (GP) by J. Koza was born which can be seen as a generalization of GAs. Only from then on these four streams blended into one specific area in the AI, Evolutionary Computing.

The main difference between these four types of Evolutionary Algorithms (EAs) is the representation they use to model decision problems, something that turned out to be vital for the effectiveness of the algorithm [8]. What they have in common is their approach to tackle problems by using the Darwinian principle of natural selection. As Darwin noticed in the mid nineteenth century, biological organisms seem to evolve over time. This evolvement or evolution seems to create environmentally adapted (and thus "better" or as Darwin proposed "fitter") species, as each species would breed new generations. The adaptation process is accomplished through a process of mutation, crossover and natural selection (see also [5]). We will now show how EC adopted this principle to solve decision problems.

2.2. The basic principles

We will first describe the setup of a standard EA, valid for all the four streams within EC that we have identified. Thereafter we will work out the typical properties that make up such streams, but first some terminology. In each EA (hybrid, memetic, co-evolutionary, multi-objective or self-adaptive) there exists a *population* of individuals. Each individual, called a *genome* according to the biological interpretation of EC, forms a solution to the problem at hand. In this interpretation all the genomes together can also be referred to as the *gene pool*. If we, for example, have a problem that consists in finding a real number minimizing a certain function, individuals encode real numbers lying in the problem domain. Likewise, if for some municipal park an optimal type of tree is sought to plant, the individuals should consist of tree definitions. We call the way in which a solution is stored the *representation* of an individual. For the first example the representation seems straightforward, we would like to have reals representing reals. In the case of the park we see that the representation is ambiguous; we can define the trees by, for example, their latin names but we also can state their height, width, depth and surface for which they

effectively catch sun. Both representations are correct but, as might be obvious, can influence the performance of the algorithm drastically.

Finding an exact/mathematical way of encoding a solution is only one of the two specifications needed to define a problem. The second specification is describing the goal or task of the problem. In EC creating a fitness function that maps all possible instances of the chosen representation to a single real number does this. This number now embodies the so-called fitness, correctness or quality of that solution. Fitness is a very general quantity and therefore each problem should have its own interpretation of how a solution's quality is assessed. For example, if we face a constrained problem we do not need to alter the original (natural) representation of the problem to satisfy the constraint. We could simply merge it into the fitness function by subtracting an arbitrary fitness value from the original value if the constraint is broken. Another possible implementation of the constraint would be to prevent the creation of individuals who break the constraint.

Having the problem defined we introduce Darwin's "survival of the fittest". As in biological processes we want our algorithm to have the ability to increase and decrease the diversity in the population (and thus solutions) as a whole. In Darwin's evolution of species this can be seen as the birth and death of creatures belonging to that a particular species. In EC, we are going to manipulate the gene pool using *genetic operators*. These operators, as was the case with representation, need to be adjusted in order to be able to work correctly for a given problem. Note that this can be done in various ways but we will only give a brief description of the general characteristics of such operators and thus skip any implementation issues, see also [2] and [3].

- *Recombination* or *crossover*. Can be conceived as the birth of one or more offspring from two or more parents through exchanging or sharing information between the parents. The idea here is that the information that is under optimization built up in the parents should be exchanged in some way. Such information can be seen as several building blocks building towards an optimal solution. These building blocks are exchanged in order to find the set of building blocks creating the optimal solution.
- *Mutation*. The random, undirected change of an individual. The mutation rate defines the mean differences before and after this transformation. A compromise between exploration and exploitation and thus respectively large and small mutation rates should be sought. Together with the recombination operator this is the main operator (re)introducing diversity in the population. Too large mutation rates will lead to erratic behavior of the algorithm and have a trial-and-error search performance whereas too small mutation rates will result in only finding local optima instead of the desired global optima.
- *Selection*. The only diversity decreasing operator. This operator can be found at two stages in the algorithm:
    - Just before the recombination step there is a selection of which parents will create the offspring. In general it is thought that fitter parents create better offspring.
    - After the manipulation of some individuals there should be some selective pressure pressing the population as a whole towards a better population in

the next generation. Roughly there are two ways of doing this. The (μ+λ)-strategy selects the new generation out of the original population before any manipulation *plus* the offspring resulting from the mutation and recombination operators. The (μ,λ)-strategy has bigger selective pressure because this operator selects the population for the next generation *only* from the offspring generated by the genetic operators. Depending on how the genetic operators work this usually leads to a brand new population in each generation cycle the algorithm makes.

How these operators work together can be seen in the following piece of generic pseudo-code on how a standard EA works, taken from [2].

```
INITIALZE population with random candidate solutions
COMPUTE FITNESS of each candidate
     while not STOP-CRITERION do
          SELECT parents
          RECOMBINE pairs of parents
          MUTATE the resulting offspring
          COMPUTE FITNESS of new candidates
          REPLACE some parents by some offspring
     od
```

The various streams within EC work corresponding to this main framework but all have their typical differences. We note that these differences are mere observations and that the specific kinds of EAs are not tied down to them. As the field of EC evolved these distinctions appeared and thus can be seen as helpful. We present a non-exhaustive list.

- *Genetic algorithms*. Frequently have a bit string representation of fixed length but can as well have a real valued representation. Recombination is implemented through bit-exchanges between parents which is the main diversity increasing operator. The mutation operator, usually implemented as random bit-flips, is a secondary operator. Selection is mainly done by the parent selection mechanism.
- *Evolution strategies*. Have a real-valued vector representation possibly extended with strategy parameters. It might have a recombination operator but the mutation is the main diversity increasing operator. Mutation is implemented by adding (multi-dimensional) normally distributed errors to the original individuals. On top of this a simple autoregressive model can be implemented in order to strategically vary the mutation rate (variance of the normal distribution) over the algorithm's run time. A regression is created which, in the initial phase of the algorithm, focuses on exploration (large mutation rates) and refines this behavior towards exploitation (small mutation rates). Selection is aggressive and therefore usually using the (μ,λ)-selection strategy.
- *Evolutionary programming*. Was traditionally concerned with evolving finite state automata designed for machine learning. Has the same representation and mutation technique as ES and uses no recombination operator at all. Unlike ES it uses a stochastic form of the (μ+λ)-selection strategy.

- *Genetic programming*. Has a tree-structure representation, resembling parse trees that can be naturally used to formulate logical expressions such a mathematical functions. Another interpretation of the tree structure can be that of syntactic expressions and thus making the individuals resemble computer programs. All the genetic operators are adapted to cope with the tree structure; further a similar approach is taken as in GA.

2.3. When to use an evolutionary approach

Surprisingly this relative simple framework seems to be able to approximate solutions of a very wide range of problems. First we note that the theoretical background of EC is small compared to that of other fields of science. The most interesting question that arises from the application of an EA is if it will succeed in finding the global optimum. It can be proven that a correctly formulated EA with certainty (probability 1) will indeed find this optimum. Though a nice theoretical result it does not serve much practical use because it does not take into account the convergence velocity. This velocity defines the speed of the algorithm convergence towards the global optimum (see [13]). For several streams within EC some convergence velocity theory work has been done but the results so far are not sufficient. The practical use of a convergence velocity theory is the mapping of a solution's quality to the execution time of the algorithm. We see that there is as of yet no general framework wherein we could derive convergence velocity formulas for *all* problems where EAs are used as solvers. Currently these velocities are known for several, simpler, fitness landscapes but not for complex problems. Ironically for complex problems execution times are of major importance. For some this lack in theoretical background renders EC into a "trial-and-error" approach of solving problems. It is stressed that this is not the case, which might clear if we view paragraph 2.2, there is a clear push towards optimal solutions. In the author's opinion this lack of theory results in a field that is very much problem-driven. This pull-factor comes from well-known unsolved[2] problems in other fields, for example the Traveling Salesman Problem (TSP). These unsolved problems are usually of complexity class NP[3] and have ambiguous solution methods. Heuristic and algorithms from other fields together with EAs come with their specific shortcomings like large execution times or non-optimal solutions. Because of their unsolved nature these problems are extremely interesting for researchers. The author also observed a small push-factor generated by EC. Because it finds its use in so many problem areas sometimes a problem is *founded* and accordingly solved by EC. Good examples of such foundation of new problems can be seen in paragraph 4.9.

[2] Unsolved should by no means be explained as that there are no methods (heuristics or algorithms) for solving these problems. Unsolved as meant here does mean that there is no overall accepted best method to use when dealing with these complex problems.

The question of when to use an EA extends the problem of not knowing much about the mathematical properties of an EA. In general we see that if a lot of mathematical understanding of a problem already exists and a good algorithm that calculates the optimal solution can be built, this is preferable over EAs. Unfortunately for a lot of problems, typically some much harder NP-hard or NP-complete decision problems[3], algorithms like that are not present. In these cases approximating heuristics are used to give some indication of the optimal solution. The problem of these heuristics is that they implement some *domain knowledge* of the problem. This leads to a vast number of different heuristics for solving a vast number of different problems, for example adding a simple capacitation constraint creates a totally different problem (see 4.1.5.). A basic EA on the contrary is typically not problem related but uses a methodology which is much more general. This leads to a more flexible, easily implementable and adaptable way of solving problems.

Though this does not promise us the perfect, unambiguous algorithm that can solve everything fast and accurate. As the No Free Lunch (NFL) theory states about "black-box" optimizers [6]: "If some algorithm a_1's performance is superior to that of another algorithm a_2 over some set of optimization problems, then the reverse must be true over the set of all other optimization problems". In this light a random search method is intrinsically as good as any other method, like EC. It must be noted that this statement is only true when reviewing *all* optimization problems. Such statements do not hold when the problem has a structure (which it does have) and this structure is *exploited*. Through exploiting the structure the NFL theory does not hold because it is now only stated that an algorithm works better for problems which posses this specific structure.

This means that EAs will have an average performance when implemented in basic form but can be strong(er) optimizers if the structure of the problems relates to that of the EA designed to solve it. The exploitation can be done in EAs via several means:

- Representation. As mentioned earlier representation affects the quality of an EA drastically. For example when solving a tree-like problem, implementation of a binary string representation generally yields worse results than keeping a tree-like structure in the representation of the genome. The domain of the problem solutions is also intrinsically determined in the possible representations. This can be exploited to not only to avoid redundancy but also to adjust the performance of genetic operators. For example if integers are encoded into bit strings for the use within a GA this could be done directly by using a bin-to-dec calculator or one could use a construction method to create a maximal hamming distance between the encoded integers. For the latter approach it is proven that a bitflip mutation operator will work better and more smoothly.
- Genetic operators. As direct results from the representation genetic operators should be adapted to handle such representation. For example the operators

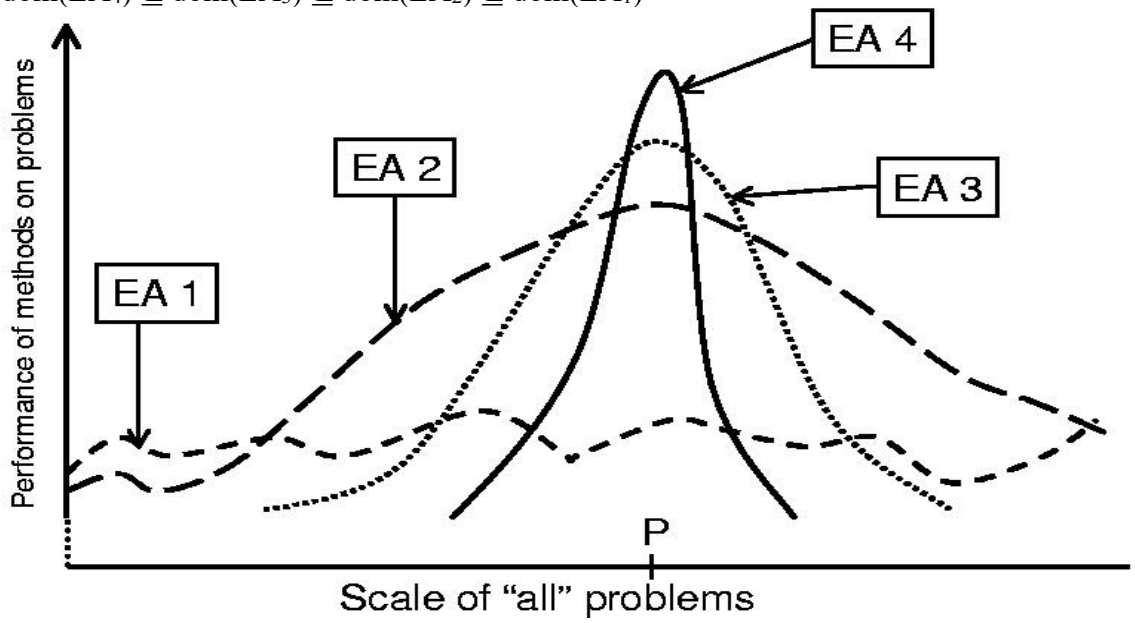
[3] An NP or Non-deterministic Polynomial-time is the set of decision problems solvable in polynomial time on a nondeterministic Turing machine. This means that given a solution its correctness can be calculated in polynomial time. It is believed but not proven that this means that finding such solution cannot be done in polynomial time (for P-problems this solution can be found in polynomial time). See [7], NP, complexity class P and NP.

should only create offspring that lies in the feasible solution domain.

- Hybridization. The implementation of a domain specific operator that is known to work well on a specific problem. For example the use of a human expert to determine some initial values for the EA or running a local search algorithm after the use of a genetic operator to ensure an even chance on fitness for the offspring made by these genetic operators. Hybridized EAs are usually referred to as memetic algorithms. We see that their interpretation differs slightly from normal EAs. In memetic algorithms an individual is more than just a possible solution; it can be seen as an agent tactically finding a resolution method for the problem.
- Codification. The translation between the phenotypic and genotypic features[4] of a genome. For example if some ordered list is represented by a genome the ordering can be done at phenotypic or genotypic level.

We conclude that EAs can certainly not be successfully applied to any problem. As Michalewicz mentiones in [8], instances of EAs that are not problem specific but general are said to be weak methods whereas problem specific instances are strong methods. We see that this is a logical result from the NFL theory. If we compare the ability to produce good results against the domain of problems where they are applicable to we see the following.

Figure 1: comparison of the performance of four EAs against their domain (dom) with $dom(EA_4) \subseteq dom(EA_3) \subseteq dom(EA_2) \subseteq dom(EA_1)$



We will now first review possible application fields of EAs and try and point out why an EA has advantages/disadvantages. In our conclusions (chapter 5) we will wrap up the benefits of EAs into a final list of when to use EAs.

[4] Phenotype space is the set of possible solutions of the real problem. Genotype space is the set of possible solutions in the EA and thus consists of all possible representations. One genotype can generally be mapped to multiple phenotypes but not vice versa creating a 1:N mapping.

3. Scope

Firstly, we will only state promising applications of EAs. It is not always trivial if an algorithm's application is successful or if it has potential but we will try and give some criteria for preferring an algorithm over others. Of course these criteria are not exhaustive and are placed in random order:

- The algorithm has a better performance
- The algorithm has a lower complexity and consumes less computational resources
- The algorithm is intuitively (more) correct
- The algorithm is more robust
- The algorithm's performance can be explained and its output is human interpretable
- The algorithm is easy to implement and thus has less development costs

We see that several of these criteria are intrinsic to EAs, though we saw from the NFL theory that e.g. a better performance can never be guaranteed *a priori*. If there is indeed an EA that gives a good performance we note that this performance is not always proven to be consistent throughout various instances of the same problem. It is known that special instances of some problems prove to be harder/easier to solve than a general instance. This means that one canonical[5] problem should be seen as a multitude of problems when some algorithm is solving it because only for a set of instances of homogenous algorithm behavior comparison is feasible. Generally not much theory exists on good classifications of problem types. The behavior of an EA depends solely on the form of the fitness surface it is exploring. Therefore to get a good grip on the quality of an EA one needs to classify this surface and have some sort of convergence velocity measure. We noticed that a sound, common-grounded methodology for describing fitness surfaces and theory on convergence velocity is non-existent (see [9]) and thus the comparative statements about success are hard to establish (e.g. "better performance").

Another way of establishing an algorithm's success is mentioned in [10]. Here a more abstract view on how to define success is taken. It skips the technical comparison issues and argues that an algorithm is successful if it generates results competitive with a human-produced result. In [10] it is stated that some results can be some as human competitive if one or more of the following criteria is met:

- The result was patented as an invention in the past, is an improvement over a patented version, or would qualify today as a patentable new invention.
- The result is equal or better than a result that was accepted in a peer-reviewed scientific journal.
- The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.
- The result is publishable in its own right as new scientific result – *independent* of

[5] A canonical form is a form (taken from [11])
reduced to the simplest and most significant form possible without loss of generality
conforming to orthodox or recognized rules

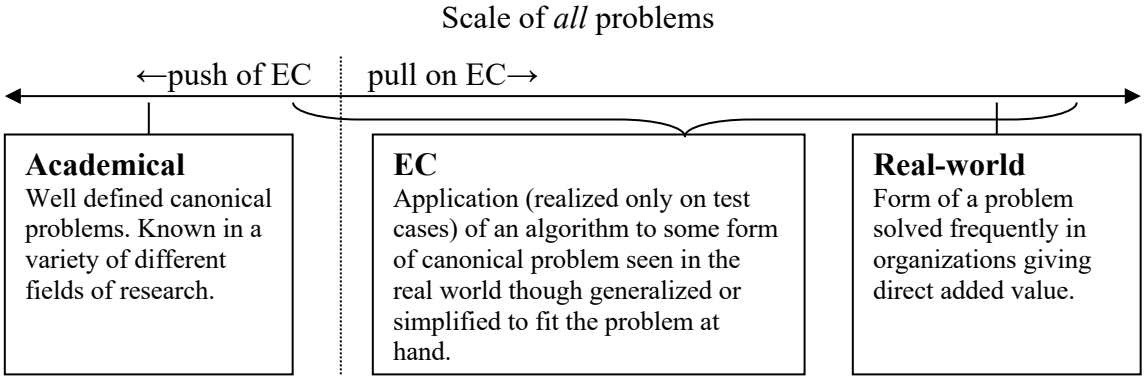
the fact that the result was mechanically created.

- The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.
- The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.
- The result solves a problem of indisputable difficulty in its field.
- The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).

This represents a clear vision on when an algorithm / solution method outperforms another. A list of twenty-four results is given in [10] all of which satisfy one or more of the above criteria. We will not use this approach however in the following categorization of promising EAs. Selecting application areas with this definition of success as criterion seems very time-consuming. One must first be familiarized with the field to determine an algorithms behavior compared to others. Then ambiguous properties like "difficulty" of a problem or scientific acceptance need to be fulfilled or a patent should be equaled or perfected. All of these criteria do not seem to be acquired in a reasonable amount of time and are a mere extension of common-grounded beliefs in science.

Secondly, we intend to discriminate application of EAs in academic problems versus real-world problems. As we saw the problem-types in EC are mostly defined by a pull-mechanism from other science fields. We can explain this behavior by supposing that the field of EC needs to prove itself amongst other known methods. A proof can be formulated in two ways. We distinguish the strong, theoretical proof of its success and the empirical or practical proof. A theoretical proof gives a sound explanation why algorithm *a* works better than algorithm *b*, given certain properties of both. From the specifications of the algorithms and the knowledge about them a mathematical interpretation can be given of the quality of an algorithm's capability of solving a problem. In reality such proofs are scarce. Usually lower and upper bounds of the solutions given by them are known but no further proof can be found. This is the case for EC as well. Thus a practical proof of an algorithm's use should be established made through standardized experimenting and comparing the results. Unfortunately the practical proof is less suitable for solving academical problems because of the lack of sound explanation of the ECs behavior and the gained success. We can see this because as a rule of thumb the solution method should be strongly correlated with the given problem. In other words if we have a theoretical or academical problem *a*, we should try and solve *a* in the paradigm where it is stated in to yield scientifically justifiable results. This seems to explain the observed tendency in EC of using a practical proof method on practical or real-world problems. This does not mean that a categorization cannot be made within the problem range where EC is applied to nor that EC is solely applied to solving real-world problems. Though as figure 2 might point out our intended differentiation seems infeasible for EC:

Figure 2: The correlation of the push and pull factors in EC and their problem domains on the scale of all problems.



In the next paragraph we will categorize some of the problems found in EC literature by an ad-hoc categorization. The categorization classes we will be stated in some general form though as we noted most of the instances in such category will have a real-world character. Mind that some instances would definitely fall into more than one category but to avoid redundancy will be stated in the class most applicable. The ad-hoc nature of the classes can be explained by their overlapping canonical form (for example a data-mining application can be interpreted as a static signal processing application or a timetabling problem can as well be interpreted as a scheduling problem/graph coloring problem) but apparent differences in the motivation behind the research of such problems should in those cases make them fall into a certain category. For example if a Brazilian school's *timetable* is being constructed (see timetabling 3) it is indeed apparent where facing a timetabling problem.

4. Categorization of EC research

4.1. Combinatiorial optimization

Of all subfields in EC research combinatorial optimization (CO) is arguably the largest and most important. It focuses on optimization of combinatorial problems thus problems where an optimal combination is sought. These problems are well known in various scientific fields like discrete mathematics and operations research. CO is a widespread problem field because a lot of real-world applications fall in this category. These problems usually have a NP-hard or NP-complete complexity class. Not necessarily[5] this means that these problems are therefore hard to solve but practically if the problem would be attacked by an exact brute force method (which is completely enumerating all the possible solutions) the performance of such algorithm would be exponential to the problem's input size. Considering the number of problems that reside in this category (see also [12]) we adopt a further classification.

4.1.1. Scheduling

Scheduling and timetabling problems can be viewed as different instances of the same problem. The problem of both is planning a sequence of operations using scarce resources in which the goal is planning all these operations with minimal cost. Usually this cost factor is calculated through the current planning scheme's usage of a resource (like time or number of personnel) or the breaking of constraint within the scheme. Formally we could view this problem as finding an optimal permutated triple {R,O,C} where R are the resources need to complete the operations O and C the optimality criteria. The different domain knowledge and different types of constraints that needs to be implemented in the problems in order to fulfill the optimality criteria cause differences in the two problems. In timetabling problems, for example, there exists a determined number of time-slots which can be filled up. Scheduling problems have a more sequential nature where the amount of time available is not defined.

Apart from these differences and resemblances we will roughly divide scheduling and timetabling by what they plan. Where in case of scheduling this usually is 'machine operations' and for timetabling, logically, timetables (see 4.1.2.). For scheduling the interpretation of the planning problem is finishing *n* products or jobs for which several operations on different machines are required. Usually there exist two constraints considering precendence and operation types which a solution must fulfill in order to be a feasible schedule. Firstly a certain operation can only be done on specific machine(s). For simplicity usually a one-on-one relation is modeled (thus exactly one machine for one operation) which might be extended to some M-to-N relation. Secondly it is likely that there exists an ordering in the sequence of the operations that need to take place before the job is finished. In scheduling this is interpreted as a directed task graph G=(T,A) where T are the tasks or operations needed and A are the directed arcs which indicate the precedence within the tasks (scheduling 1). Problems where this precedence in operations is the same for all the jobs being processed are usually referred to as Flow Shop problems (see scheduling 2,5,7). If precedence is present these problems are called the harder Job

Shop problems (scheduling 3,4,6,7). For further classifications within the field of scheduling see scheduling 7.

There exist a wide variety and enormous size of scheduling problems in real-world applications and corresponding with that a lot of research has been undertaken to tackle these problems. EC is becoming a more and more accepted application to use for these problems because it yields fast and robust results. We note that speed in these algorithms is becoming more important due to globalization and corresponding increase in number of machines, operations and jobs to be optimized. Furthermore seemingly small changes in the problem structure (for example do we have one or two machines capable of handling a certain operation) can change the problem and algorithmical results drastically. Therefore a range of other, non-EC related, means of solving scheduling problems exist all which work well on their own small range of specific problem instances. We see that in EC usually less discrimination exists between these different instances because an EA is thought to be general enough to solve them uniformly. Uniformly means that an EA can incorporate several algorithms and can coordinate when to use which to generate an optimal resolution method. We see that if an EA is hybridized and fine-tuned to one specific real-world case (see scheduling 8) the results can be very competitive and the execution time is generally reduced when compared to other means of solving.

1. S. Esquivel, C. Gatica, R. Gallard. *Performance of evolutionary approaches for parallel task scheduling under different representations*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 41-50, 2002.
2. S. Esquivel, G. Leguizamón, F. Zuppa, R. Gallard. *Performance comparison of alternative heuristics for the flow shop scheduling problem*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 51-60, 2002.
3. L.W. Cai, Q.H. Wu, Z.Z. Yong. *A genetic algorithm with local search for solving Job Shop problems*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 107-116, 2000.
4. E. Hart and P. Ross. *A systematic investigation of GA performance on Job Shop scheduling problems*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 277-286, 2000.
5. C.L. Chen, S. Vempati, Venkateswara, N. Aljaber. *An application of genetic algorithms for Flow Shop problems*. European journal of operational research, vol. 80, p. 389-396, 1995.
6. H.L. Fang, P. Ross, D. Corne. *A promising genetic algorithm approach to Job-shop scheduling, rescheduling, and open-shop scheduling problems*. In S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, p. 375-382, 1993.
7. H.L. Fang. Genetic Algorithms in Timetabling and Scheduling. PhD thesis, Department of Artificial Intelligence. University of Edinburgh, Scotland, 1994.
8. Nutech case studies for instance the scheduling of oil extraction for ChevronTexaco or the Unilever's scheduling of supplying stores for future product demand, see URL: http://www.nutechsolutions.com/case_studies/default.asp.

9. N. Urquhart, K.J. Chisholm, B. Paechter. *Optimising an Evolutionary Algorithm for Scheduling*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 307-318, 2000.

4.1.2. Timetabling

The interpretation of a timetabling problem is one of a planning problem where a timetable needs to be optimized. The timetable under optimization should satisfy a set of hard and soft constraints. These constraints can include a variety of features the timetable must posses. For example in a school's timetable one can construct a soft constraint stating it is preferable to have some diversity in the subjects of daily courses or that scholars should not have lunch all at the same time because this would overcrowd the lunchroom. Hard constraints can be modeled likewise. A very common hard constraint is to restrict the application of some resource. As in scheduling there should be a one-to-one relation of resources (e.g. a classroom, a professor) to operations (e.g. a class of students). Furthermore it is common to include a hard constraint stating that all the operations must be handled and thus for example that all the students have to attend a predetermined amount of classes.

Because of the fixed time-slots laid out in the problem's constraints the timetabling problem can also be viewed as an assignment problem where resources and operations are assigned to each other. This viewpoint lets the problem be translated into a set covering problem which is formulated as finding a subset of columns in a zero-one *m*x*n* matrix A. This should be done in such a way that the columns cover all the rows of the matrix at a minimum cost, calculated by multiplying A with a cost-vector (see timetabling 2,9). It also leaves room for modeling the problem as a graph colouring problem (see scheduling 10 and 4.1.3.)

As for scheduling problems, variety and size of the different timetable problem instances is huge. The timetabling problem finds its practical application in schools (colleges, universities, high school, see timetabling 3,4,5), flight scheduling, re-scheduling (see timetabling 7,8), employee scheduling to satisfy employee contracts and minimal staffing levels (see timetabling 1,2,6,9) and various other problems where resources need to be allocated on a timed basis. For timetabling problems EC earns its advantage over other methods due to the same factors as the scheduling problems but also inherits the same constraints of its generalized application. EC is usually a less time-consuming approach to tackle these problems since they can be very complex to model. EC provides a robust and generic ability to handle constraints and model the problem making it a flexible algorithm with potential.

1. P.V.G. Bradbeer, C. Findlay, T.C. Fogarty. *An ambulance crew rostering system*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 267-276, 2000.
2. E. Marchiori, A. Steenbeek. *An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 367-381, 2000.

3. G.R. Filho, L.A.N. Lorena. *A constructive evolutionary approach to school time-tabling*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p.130-139, 2001.
4. M. Bufé et al. *Automated solution of a highly constrained school timetabling problem - preliminary results*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 431-440, 2001.
5. C. Di Stefano, A.G.B. Tettamanzi. *An evolutionary algorithm for solving the school time-tabling problem*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p.452-462, 2001.
6. M. Gröbner, P. Wilke. *Optimizing employee schedules by a hybrid genetic algorithm*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p.463-472, 2001.
7. T. Grosche, A. Heinzl, F. Rothlauf. *A conceptual approach for simultaneous flight schedule construction with genetic algorithms*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 257-267, 2001.
8. M. Love, K.R. Sorensen, J. Larsen, J. Clausen. *Disruption management for an airline - rescheduling of aircraft*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 315-324, 2002.
9. M. Finger, T. Stützle, H. Lourenço. *Exploiting fitness distance correlation of set covering problems*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 61-71, 2002.
10. Tatties II program by Ben Paechter is an evaluation version of a simple timetable planner designed for the use in schools or universities. URL: http://www.dcs.napier.ac.uk/~benp/summerschool/summerschool2.htm

4.1.3. Graphs

In the mathematical graph-theory founded in discrete mathematics several real-world problems find a matching interpretation. In general we think of a graph problem as the pursuit of some goal, for example an optimal route, in a graph G = (V,A). As we noted earlier some scheduling problems can be modeled as directed graphs but are not categorized here because of the coherency within scheduling problems and the sheer number of specific scheduling problems. Neither will we state any arc routing problems here (see 4.1.5) for the same reasons. This leaves us with the following non-exhaustive set of graph problems:

- *Graph coloring*. One wishes to color the vertices V of G with a minimum number of colors, such that no two adjacent vertices receive the same color. If now *k* colors where needed for the coloring we see that the problem is equivalent to finding a disjunct, non-empty, partitioning of size *k* of the vertices in G. Because the partitions are disjunct the partitioning can be adopted to solve a simple timetable problem, interpreting the colors as available time slots and the arcs A as the constraint that a one-on-one relation between all the time-slots and resources/operations exists. Solving the coloring problem then gives the number of time-slots needed to plan all these events (see scheduling 7 and graphs 2,5,6). The technique of graph coloring can be generically applicable to assignment problems where the colors represent the resource that needs to be assigned, for

example mobile telephone networks or a navy fleet. It seems that an EA in its canonical form is not capable of solving coloring problems because of the symmetry of the solution space. The symmetry renders the standard genetic operators useless. In hybridized form with applicable genetic operators an EA does seem capable of solving coloring problem (see graphs 7,9).

- *K-connectivity*. This problem arises typically from physical network optimization problems. The world is becoming more interconnected and thus a network's number of nodes that need to be connected increases as well as the amount of data passing through these nodes. The optimization problem lies in minimizing the cost of adding connections in the network in order to keep up with the enlargement of the network as a whole together with the resulting communication costs in the network. If we translate this problem to a graph problem this means creating a k-connectivity graph where a k-connectivity graph is defined as a graph in which at least k arcs must be removed in order to disconnect the graph. Typical examples of such networks are telephony networks, intra-nets, the internet or power generators in the US. There exists some fair results of the application of EAs to such problems (see graphs 1,3,8) though the form of the EA needs to be adapted (again especially its genetic operators) to fit the problem possibly making EAs not the most natural approach.
- *C-means clustering*. The problem consists in finding c non-empty clusters of vertices in a graph where each vertex is assigned to exactly one cluster. The criteria for the belongingness of a vertex to one of the c clusters can be adapted to the practical problem at hand. The problem is difficult because solutions found are often degenerated partitions in which amount of data points per cluster is unequally spread. A balance between minimal sensitivity and local extrema should be pursued. The clustering problem is frequently faced in exploratory data analysis where some abstraction of the data is sought in order to make better assessments of its meaning. For example in data-mining this technique can be used to derive classifications of explanatory variables and in image analysis to find correlating features (see also 4.3 and 4.6). In general some segregation is sought which is appropriate for the data at hand. EAs seem to be a powerful, fast solver for these problems (see graphs 4) because of its adaptability and flexibility. The balance in the segregation that is sought can be regulated the parameters of the EA. Given a good fitness function the algorithm might even be able to meta-evolve its own strategic parameters to do so.

1. S. Kersting, G.R. Raidl, I. Ljubić. *A memetic algorithm for vertex-biconnectivity augmentation*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 102-111, 2002.
2. D.A. Fotakis, S.D. Likothanassis, S.K. Stefanakos. *An evolutionary annealing approach to graph coloring*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 120-129, 2001.
3. I. Ljubić, G.R. Raidl. *An evolutionary algorithm with stochastic hill-climbing for the edge-biconnectivity augmentation problem*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 20-29, 2001.

4. L. Meng, Q.H. Wu, Z.Z. Yong. *A faster genetic clustering algorithm*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 22-31, 2000.
5. P.M. Pardalos, T. Mavridou, J. Xue. *The graph coloring problem: a bibliographic survey*. In D.Z. Du et al., editors, Handbook of Combinatorial Optimization, vol. 2, p. 331-395, 1998.
6. N. Barnier, P. Brisset. *Graph coloring for air traffic flow management*. Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, p. 133-147, 2002.
7. P. Galinier, J.-K. Hao. *Hybrid evolutionary algorithms for graph coloring*. Journal of Combinatorial Optimization, vol. 3, p. 379-397, 1999.
8. L. Davis, S. Coombs. *Optimizing network link sizes with genetic algorithms*. In M. Elzas et al., editors, Modeling and simulation methodology: Knowledge systems paradigms, p. 317-331, 1989.
9. A.E. Eiben, J.K. van der Hauw, J.I. van Hemert. *Graph coloring with adaptive evolutionary algorithms*. Journal of Heuristics, vol. 4, no. 1, p. 25-46, 1998.

4.1.4. Knapsack

The interpretation of this problem is that a knapsack needs to be filled with items but only a limited amount of weight can be fitted into it. The goal is to fit as much profit in the knapsack as possible. Thus from the set of all items given their amount, weight and profit they generate we need to select an optimal item set. Next to the main goal of maximizing the profit in the knapsack other constraints can be formulated. Usually the amount of constraints in knapsack problems is high thus making the feasibility of finding *any* solution an important topic (see knapsack 2). This problem can be applied in every real-world problem where for a scare resource a practical upper bound is known and an optimal allocation is sought. For example if an investor wants to invest a certain amount of money in three possible projects he would like to invest it in such a way that an optimal (expected) profit is generated. An alternate constraint for this example would be setting a maximum amount of money that can be invested in each project separately. We can also identify the multi-dimensional knapsack problems where not one but *n* knapsacks are filled making the problem much harder (see knapsack 3). EC should only be applied for such complex problems, heavy constrained or extremely large, because for those problems few heuristics exist. EAs seem very capable of handling such problems compared to other methods (see knapsack 1,3).

1. J. Levenhagen, A. Bortfeldt, H. Gehring. *Path tracing in genetic algorithms applied to the multiconstrained knapsack problem*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 40-49, 2001.
2. J. Gottlieb. *On the feasibilty problem of penalty-based evolutionary algorithms for knapsack problems*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 50-59, 2001.
3. S. Khuri. *The zero/one multiple knapsack problem and genetic algorithms.* In E. Deaton et al., editors, Proceedings of the 1994 ACM Symposium of Applied Computation, p. 188-193, 1994.

4.1.5. Arc routing

Arc routing problems are probably the most famous combinatorial optimization problems around. Strictly the problem is a graph problem but is categorized separately here because of the amount of the research in this field. The most common arc routing problem is the Traveling Salesman Problem (TSP), which is the problem of finding the shortest closed tour through a given set of *n* cities visiting each city exactly once. In a graph these cities can be seen as the vertices of the graph and the arcs as the possible routes between cities. The length or weight of each arc is interpreted as the distance between the cities it connects. In a standard TSP a closed tour should be found and the weights of the arcs should equal their geometrical lengths and that thus the problem can be drawn accordingly in Euclidian space making it a Euclidian-TSP. Of course the modeling of practical problems may require the use of open tours or non-Euclidian metrics for the lengths of the arcs. As a generalization of the TSP there exists the multiple TSP (mTSP). An mTSP differs from a standard TSP by letting *k* instead of one sales agents travel past the *n* cities creating *k* separate cycles in the graph. The problem of mTSP is again find *k* closed circuits visiting all the *n* cities while minimizing the path weights. mTSP is a variant of the k-TSP problem where it is also required that all the closed circuits have one common base city.

If we interpret the cities being visited not by sales agents but by for example delivery trucks we wish to also model a limit of the number of goods the truck can carry. For this purpose the capacitated TSP is constructed where the standard TSP was not capacitated. Of course an arbitrary number of other constraints can be set up, making the solving of the problem generally harder. One can also try to make a large, difficult TSP problem easier. It is know that touring cities very close to each other can be solved with a simple nearest neighbor algorithm equally well as with a complex TSP. Especially for clustered data a combination of simple and complex approaches works very well. One of those methods is called the street based routing method. If for example we need to optimize the delivery of mail to all the households in a suburb we could model this as a TSP with a high number of cities but this would make the problem needlessly complex. Because we know that all the households in one (part of the) street will be very close to each other it is spilled computational time to let the TSP algorithm approximate the optimal route for those households. We much rather apply a simpler algorithm to the intra-street households and the more complex TSP on the inter-street optimization. This is what has been done in street based routing where the intra-street mini-problems are solved by a deterministic procedure (see arc routing 1,3).

Applications of arc routing problem can be found in abundantly in real life. The physical routing problem includes indeed the traveling salesman but in general everything that needs to pass along several points on a map in preferably the fastest or shortest way like mail, packages, garbage collections, deliveries of supplies, picking of orders within a warehouse, public transport, routing of network traffic etc. (see for example arc routing 4). In practice a TSP problem can, apart from the different models described above, also have a dynamic component whereas the situation changes the algorithm quickly computes the optimal strategy for the changing conditions. Especially in routing network traffic and delivery of supplies this feature can be a must-have (we would think of nodes

in the network being deleted/created or a priority supply which must be handled immediately). This is where EC can provide extra functionality over other algorithms. We can roughly state that optimal values calculated by EAs and non-EAs are more or less the same for any standard TSP problem. Though the computation time in EC is very scalable compared to other methods. At any time during an EAs execution the optimum so far can be given together with an arbitrary set of other possible solutions making dynamic TSPs possible. Also because of this feature a major increase in the number of cities would not have disastrous effects on the computation time making such problem still solvable in reality, something which is definitely not the case for several other ways of solving.

1. D. Sofge, A. Schultz, K. de Jong. *Evolutionary computational approaches to solving the multiple traveling salesman problem using a neighborhood attractor schema*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 153-162, 2002.
2. R Baraglia, J.I. Hidalgo, R. Perego. *A parallel hybrid heuristic for the TSP*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 193-202, 2001.
3. N. Urquhart, B. Paechter, K. Chisholm. *Street-based routing using an evolutionary algorithm*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 495-504, 2001.
4. A.Vasilakos, C. Ricudis, K. Anagnostakis, W. Pedrycz, A. Pitsillides, X.Gao. *Evolutionary- fuzzy prediction for strategic ID-QoS: routing in broadband networks*. International Journal of Parallel and Distributed Systems and Networks, vol. 4, p. 176-182, 2001.
5. K. Bryant. *Genetic Algorithms and the Traveling Salesman Problem*. PhD thesis, Department of Mathematics, Harvey Mudd College, Claremont, 2000.
6. P. Jog, J. Suh, D. Gucht. *Parallel Genetic Algorithms Applied to the Traveling Salesman Problem*. SIAM Journal of Optimization, vol. 1, no. 4, 1991.
7. M. Land. *Evolutionary Algorithms with Local Search for Combinatorial Optimization*. PhD Thesis, Computer Science & Engineering Department, University of California, San Diego, 1998.
8. An online demo by K. Boukreev of a genetic algorithm solving a TSP problem instance. URL: http://www.codeproject.com/cpp/tspapp.asp
9. J.T. Alander. *An Indexed Bibliography of Genetic Algorithms and the Traveling Salesman Problem*. Department of Information Technology and Production Economics, University of Vaasa, 2000.

4.2. Design

One can think of a vast number of things, conceptual or physical, that need to be designed or might be designed better. In the case of physical design one might think of architects designing buildings or industrial designers designing all sorts of everyday objects. What happens in such designs is that a number of design issues are being solved simultaneously. We separate functional design dealing with the use and functionalism of the designed object, aesthetic design dealing with the appearance of an object and the construction design identifying the materials the object should be made of and how it should be built. We note that construction design enables as well as constrains the other design factors. For example an architect may have come up with some wild aesthetically very correct design that is not feasible to build physically. This is a typical example of a construction design's constraining nature. Furthermore we see that the fitness of a construction design is completely determined by accepted laws in nature (gravity, law of effect etc.) and could be expressed in mathematical terms. On the contrary in functional and surely ethical design the fitness cannot be determined univalenty.

Conceptual designs have a much broader scope then physical design because they embrace ideas instead of physical things. Aesthetical, functional and constructional factors still exist in such design but should be interpreted differently. For the intended constructional factors we still observe its objective, univalent nature because constructions of concepts always exist in some paradigm (they are for example stated in a language) and usually refer to and/or make use of existing conceptual frameworks (if someone would want to prove a new mathematical theorem one would use old, proven theory to do so).

If we would apply an EA to a design problem we need to be able to determine the fitness of a structure, preferably on an unambigious and deterministical way. In EC therefore a design problem is a problem of optimizing the construction of a design. Of course what type of structure or idea is optimized determines the base for fitness evaluation wholly (see also 4.8 and 4.9 for a completely opposite approach to fitness). For the construction of possible solutions suitable building block are combined with some means of connecting them. Thereafter their fitness should be calculable.

A good example of a conceptual design would be designing an ARMA model. In such model a balance between the Moving Average (MA) and Autoregressive (AR) components is sought. Putting these together in the ARMA formula creates a (time-) series prediction model. In these models the fitness is evaluated by, of course, their predictive behavior together with the generality of the model. Adding components will always increase predictive behavior in ARMA models but is only until some extent valuable to maintain generality (see design 3). Designs in computer science can also be optimized. If we see a program running on a computer as a black box we see that it is feeding the computer machine code and getting output back from it. We might want to optimize the program to create better output. A good example of computer program design is the design of a protocol for communication between computers. Such protocols should be able to let computers communicate and minimize the response times and

number of faults while doing this. It also should avoid network congestion or network collapse and thus be robust. Such a protocol can be seen as a program and thus as a mere piece of machine code that can be (and certainly needs to be) optimized (see design 3,4,5). In these articles this approach did not yield the intended results, which is competing with current protocols, but they did come up with working protocols. This means that the method being used is viable for later success.

For physical design the optimization goal usually lies in natural features of a design such as strength, hardness, aerodynamicy or speed. In airplane design for example, aerodynamics plays a vital role. Planes fly by using an airfoil (wing) to generate an upward lift caused by differences in airspeed on the upper and lower side of the airfoil. It is known that at different altitudes different types of air are found (different amount of oxygen, air-pressure) and thus different wings should be designed to make the best use of these different properties (see design 1). This makes the design difficult because only one airfoil can be used which subsequently needs to fulfill all these requirements.

We see that constructional design problems are still hard to solve because they are in general also hard to model correctly (especially conceptual design problems). In terms of intuition the use of EC does seem applicable in this field because its interpretation in biology is similar. In real life creatures created through evolution also can be seen as smart designs certainly capable of performing certain tasks. Furthermore if a design problem is stated clearly we see that solution domain is usually theoretically infinite and practically thus huge. This property makes an EA suitable too because of its known ability to handle these out-of-bound problems very well.

1. D.J. Doorly, S. Spooner, J. Peiró. *Supervised evolutionary methods in aerodynamic design optimization*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 357-366, 2000.
2. T. Minerva, I. Poli.*Building ARMA models with genetic algorithms*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 335-342, 2001.
3. F. Corno, M. S. Reorda, G. Squillero. *Evolutionary techniques for minimizing test signals application time*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 183-189, 2002.
4. N. Sharples, I. Wakeman. *Protocol construction using genetic search techniques*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 235-246, 2000.
5. N. Ireson, Y.J. Cao, L. Bull, R. Miles. *A communication architecture for multi-agent learning systems*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 255-266, 2000.
6. Peter J. Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufmann. 1999
7. Peter J. Bentley, David W. Corne, editors. *Creative Evolutionary Systems*. Morgan Kaufmann. 2002

4.3. Image analysis

The analysis of images is important for the mechanisation of processes. The problem lies not in seeing images but solely in interpretating images. Humans rather unconsciously perform this task on a day-to-day basis but formally describing this task in order to implement it in computers seems a very complex matter. Mechanization is needed because for some problems the quantity of the images that need to be reviewed is too much to be done economically by humans themselves. Also we could think of some qualitative expert knowledge that is needed to perform the analysis task. Because such experts might be scarce and not always accurate a computer can assist in such analysis.

The problem of precision vs. generality seems to be key question for these type of problems. An algorithm should be general enough to see “the big picture”, that is identifying certain objects which spread out over a substantial area of the image though also be precise enough to identify the outlines of such objects and identify small anomalies. In image analysis we can separate feature-based methodes, voxel intensity based methods and image filters. Image filters (see image analysis 4) do not interpret the data in an image but pre-process the picture before the main algorithm will interpret it. Constructed methods for image analysis usually rely on perfect pictures but of course this can never be achieved by modern technology because there is always a certain physical limit to the pixel-count in a picture. The picture should then be filtered in order to emphasize areas or intensity levels in pictures to make analysis more accurate.

Feature-based methods search for features in an image. This is usually done through comparing known template images with the image being interpreted. This can be done in numerous ways. If for example we look at a production line producing beer in glass bottles one would need to check for stains or tears in the bottles. Because of the amount of bottles this cannot be done manually. If we have a template of a good beer bottle then the algorithm calculates the differences between the beer bottle image being reviewed and the template. If the correlation exceeds a certain bound the analyzed beer bottle should be carefully checked (see image analysis 1). For this simple problem the key lies in learning an algorithm the differences between the two images.

In voxel intensity based methods there exist no template images. This is necessary if the exact form of the phenomenon we want to analyse is not known a priori. Also for pictures on different scales (see image analysis 2) this can be useful. We see that the problem’s difficulty increases dramatically if no a priori knowledge is available. The algorithms in this category label certain points in the pictures that the algorithm finds of interest. A point of interest is a point where the coloring of the image is different than points lying next to it or points that are more or less identical to points already labelled interesting which lie in the neighborhood making that point likely part of the same object. We see that in hospitals this method can be of value raising the accuracy of detecting diseases. In a hospital we could find images from x-ray tomography (CT-scans), Magnetic-resonance imaging (MRI-scans), mammograms, single-photon-emission tomography (SPECT-scans) or positron-emission tomography (PET-scans). Detection of diseases on these pictures (like cancer) can be hard, even for an expert eye, because it involves noticing

minor details. Therefore automated image analysis can be used to assist these experts (see image analysis 5,6).

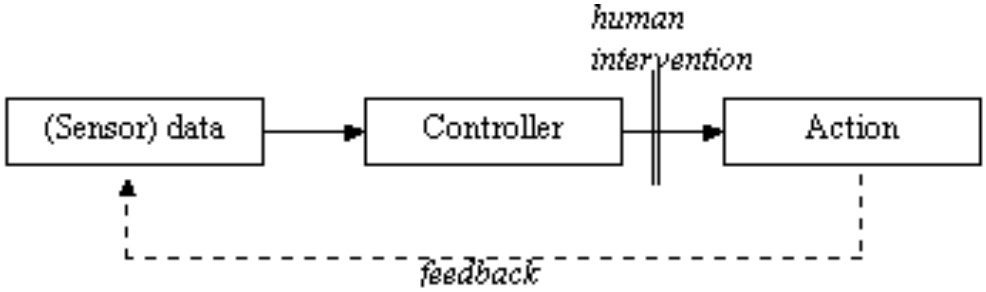
The real advantage of EAs comes in play when the voxel intensity based methods are used for analysis. Because of their domain independent knowledge, like EAs, they can be used for all kinds of image analysis jobs (see also image analysis 2,3) focussing on the recognition of objects. The flexibility of EC enables the voxel intensity method to learn to perform a certain analysis correctly. Especially fine-tuning the parameters of the EA does this. These parameters intrinsically tell the EA how to perform its search function and how to evaluate a fitness value making the trade-off between generality and precision. This makes the EA capable of performing a very specific job though the original modelling of the algorithm can still be said to be problem domain independent.

1. E. Piazza. *Surface movement rader image correlation using genetic algorithm*. In E.J.W. Boers et al., Applications of Evolutionary Computing, 2001, p. 248-256.
2. D. Prabhu, B.P. Buckles, F.E. Petry. *Scene interpretation using semantic nets and evolutionary computation*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 34-44, 2000.
3. D. Howard, S.C. Roberts, C. Ryan.*The boru data crawler for object detection tasks in machine vision*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 222-232, 2002.
4. L. Sekanina. *Image filter design with evolvable hardware*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, Applications of Evolutionary Computing, p. 255-266, 2002.
5. Y. Fan, T. Jiang, D.J. Evans. *Medical image registration using parallel genetic algorithms*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 304-314, 2002.
8. A. Bevilacqua, R. Campanini, N. Lacnonelli. *A distributed genetic algorithm for parameters optimization to detect microcalcifications in digital mammograms*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 278-287, 2001.

4.4. Decision support systems / controllers

A decision support system can be viewed as follows.

Figure 3: a schematic decision support system



We see that the system generates actions based on (sensor) input that is processed by some algorithm capable of making decisions; hereafter we will call such system a controller. We can view this process as an ongoing decision process with or without feedback. Feedback controllers are called closed-loop controllers and controllers without feedback open-loop controllers. We can also discriminate in the use of human intervention in the decision process. Naturally we see that for critical, complex applications more human intervention and guidance is required then for standard controlling functions. For applications where a large number of decisions are required in a short time frame human intervention can only be limited.

Open-loop controllers make decisions on the same input each time a decision cycle is made. The decision algorithm can be stochastic so the output would not be the same for each cycle but will have some of the same overall properties. Closed-loop controllers show a very different behavior. In each decision cycle where a decision is made and a corresponding action is performed this action changes the sensor inputs of the next decision cycle. Thus the decision algorithm will respond to altering conditions in the system as a whole. As a rule of thumb therefore closed-loop systems are used if a lot of decisions need to be made throughout time and where the system undergoes a certain dynamic. Because open-loop systems would not perform well under changing external conditions they are used in more static environments. Typically in these evironments decisions are taken less frequently. Corresponding to the amount of decisions both types of controllers have to take in a run of several cycles ,the complexity of the decisions to be made is usually higher in an open-loop controller than it is in closed-loop controllers. This goes hand in hand with the amount of human intervention in the system. For open-loop controllers the output of the decision system is a mere guidance for the user who will make the decision based not only on the output of the controller. In closed-loop controllers the controller makes the decision by itself and the user would only be warned and possibly interfere if some dangerous system state would emerge and thus the controller is much more unsupervised than is the open-loop controller.

We find the application of controllers all throughout everyday life. Simple examples of such systems are sliding doors, elevators, cruise control in cars, peripheral controllers in

computers, routers, VCRs, traffic lights, camera's etc. All of these everyday examples work on feedback and thus seem to us humans to be automatic processes. Because all of these examples seem to work fine, the role of EC lies in the decision problems that are much harder to control. The hardness of such problems for the open-loop controllers usually lies in the lack of knowledge of the correlation between input signals and the desired output. For the closed-loop controllers the problem is a large amount of input data and the small timeframe in which a decision should be made. For the first case a simulation could be written in order to predict the output of a certain decision. For the latter case usually a frame-work of rules is derived from historical data which is then projected on new input signals. We now observe a categorization problem. On one hand we see that the system has learned from former input data in order to make its choices at the moment needed, we could thus say this is actually learning (see 4.5 for a definition). On the other hand we see that a set of rules is generated from historical data and that typically the number of input variables is large. This knowledge-generation is usually called data-mining (see 4.6). We again note that the categorization here is ad-hoc based, though we intend to categorize applications here where both the statical component (data-mining, a one-time generated set of rules) and the dynamical component (learning from prior events) are integrated. We will try to analyze the two components separately in the next two sections (4.5 and 4.6).

We see that in the case of motorway control these two components interact with each other. We see that the circumstances on motorways can change rapdily from very scarce and high speed traffic at night to very dense traffic at peak hours, usually around 8 am and 5 pm. To be able to control traffic detection, prediction of journey times or traffic lights the system needs to be very adaptable to changing situations (see also decision support systems 2,3,5). In the case of robotics (see decision support systems 4) the feedback and thus dynamic nature is apparent. If we would want the robot to walk using its feet it must continously re-balance itself in order to achieve the goal not to fall but to move. It does this through the use of its equilibrium sensors. The problem at hand again needs an adaptable algorithm in order to deal with previously unencountered problems but even more needs to be computationaly quick. As all humans learn through *experience* losing your balance (and consequently falling if the situation is not dealt with) can take place in a single second.

The advantage of using EAs for these types of problems is their natural implementation of learning and their robustness. The learning component is accomplished by storing a population in the algorithm. Theoretically the population as a whole should contain fitter solutions (in the case of a controller, "actions") as the EA progresses. For a system under control we can say it is a system which can only be in a finite number of states, for example the robot can be in balance, out of balance with tendency to fall backwards, forward etc. Now we see that each state has its own optimal response, respectivly go in the desired direction, put one feet backward, forward etc. Such (simplified) examples can directly be translated into the population where the population contains a collection of those (near-)optimal state-bound actions and wherein optimal moves for certain states are continously sought but where the other moves from different states are still preserved. In a sense we could call this mechanism a sort of experience building.

1. J. Dorado, J.R. Rabunal, J. Puertas, A. Santos, D. Rivero. *Prediction and modelling of the flow of a typical urban basin through genetic programming*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 190-201, 2002.
2. D. Howard, S.C. Roberts. *The prediction of journey times on motorways using genetic programming*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 210-221, 2002.
3. S.C. Roberts, D. Howard. *Detection of incidents on motorways in low flow high speed conditions by genetic programming*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 245-254, 2002.
4. B. Andersson, P. Svensson, M. Nordahl, P. Nordin.*On-line evolution of control for a four-legged robot using genetic programming*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 319-326, 2000.
5. A Nutech case study for the dutch ministry of traffic (ministerie van verkeer en waterstaat). URL: http://www.nutechsolutions.com/pdf/cs_dutch_min_traffic.pdf.

4.5. Machine learning

A lot of areas in AI are related to the technique of machine learning, therefore the use of the term has become broader and broader as the field of AI expands. As we saw earlier decision support systems are said to learn when feedback of a controller's actions is given back to it and the controller adjusts its output to it. Extending this discussion EC can as well be viewed as a learning tool in the way we explained in 4.4. It is not the author's intention to view machine learning like this, we will only review applications here that explicitly learn an EA a concept.

To learn something we must first ask ourselves what learning is. Learning can be interpreted as simply the gaining of knowledge, though this prolonges the question to what knowledge is. We can separate knowledge roughly into two categories namely declarative knowledge and procedural knowledge. Declarative knowledge find its use (apart from the discussion whether knowledge has use at all) in giving information *about* things (for instance, that "A" is the first letter of the alphabet). Procedural knowledge gives information on *how to do* things (for instance, following steps A,B and C will dig a hole). Apart from only gaining knowledge learning has the vocation that it is done at least partly by the entity (here the computer) rather than entirely by a teacher or knowledge provider (see [7], "machine learning"). The term *learning* normally implictly refers to the learning of humans. The main objective in learning is the learning itself and the ability to reproduce the learned patterns. Sometimes the behavior emitted by a cognitive system[6] is seen as the goal of learning though strictly this is not the ultimate, but merely a practical, goal of learning.

[6] A system which has cognition. Cognition can be seen as a mental process in any form of creature like most naturally humans, animals, aliens and even non-life forms like computers. It is defined as the ability to learn or investigate something.

As for all EA instances in learning there should always be a task stating what should be learned, a certain quantification defining if the learning went well and a representation for knowledge. In the most generally accepted case the task is learning a concept and the representation of knowledge is rule-based (see learning 2,5). We see that knowledge is attained if by applying the rules on input data the rules discriminate well between negative and positive instances of the concept which can be any concept of choice. These rules can be generated using first-order logic, as formulated in the REGAL system (see machine learning 4), which indicate that if certain events in the input occur simultaneously this results in a certain reaction. By adjusting these rules to fit the desired output the system is trained.

We can also learn a computer how to operate certain (simulated) architectures, for example we can learn a computer how to move a (simulated) arm by using a couple of virtual muscles (see learning 1,3). In learning 3 the arm is a real prosthetic arm used to help disabled people. The arm receives input from the user's and the arm then needs to learn what the user means with this input and act accordingly. What is learned is only the meaning of the users input, not learning the consequenences of the emitted behavior. In learning 1 we see a similar approach for moving a simulated baby's arm towards some predefined spot. Without knowing the spot and even without knowing there is such a spot at all the algorithms starts moving the arm and should learn how to get to the sweet spot and stay there.

We see from the examples above that the learning problem is general in nature. The definition of knowledge can be implemented in numerous ways into algorithms and the representation of knowledge is ambigous because of different types of knowledge. We see that EAs also have a general nature and thus might prove to be well at solving such problems. The difficulty of using EAs for these problems is the representation of knowledge in the algorithm. As in done in learning 3 we could evolve series of logic gates who instantly give a solution to the problem. We see that the knowledge is fully represented by such gates and that from the arrangenment of logical operators created by the EA a certain knowledge or rule(s) can be extracted. Though if a learning problem would increase drastically in complexity this method may not work anymore or yield uninterpretable logic gate constructions. In learning 1 we saw that the knowledge lay in the performed actions of the hand. As in standard EAs a series of possible actions (solutions) was generated and by reviewing their simulated behavior the algorithm learned which action to take. This method seems to yield searching all the possible actions and finding the best action to implement. The interpretation of knowledge attained while doing this seems non-existent. According to the author the most preferable approach to deal with learning problems in EA (and definetely a promising one) is the use of the REGAL system which encodes each genome as a logical rule (see learning 2,5). The REGAL system then uses a distributed approach to identify a set of rules for the task to be learned at hand. A supervising algorithm controls the distributed EAs to accommodate cooperation amongst them. The interpretation of each distributed EA is that it will find a niche in the set of rules generated so far and will create a new rule to increase the fitness of the set of rules as a whole. Therefore the population in each specific distributed EA is competing for the creation of new rules as well as the

distributed EAs amongst each other. Because of the clear representation of knowledge through a rule-based system this seemes the best algorithm to implement knowledge. This is confirmed by similar AI applications which usually also apply rule-based systems to implement knowledge. The extra evolutionary approach of the REGAL method could make it a serious competing algorithm for these type of problems.

1. S. Delepoulle, P. Preux, J.C. Darcheville. *Selection of behavior in social situations. Application to the development of coordinated movements*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 384-393, 2001.
2. F. Neri. *A study on the effect of cooperative evolution on concept learning*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 414-420, 2001.
3. J. Torresen. *A dynamic fitness function applied to improve the generalisation when evolving a signal processing hardware architecture*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 267-279, 2002.
4. A. Giordana, F. Neri. *Search-intensive concept induction*. Journal of Evolutionary Computation, vol. 3, no. 4, p. 375-419, 1995.
5. F. Neri. *Evolutive modeling of TCP/IP Network traffic for intrusion detection*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 214-223, 2000.

4.6. Data-mining

Data-mining is the practice of searching large amounts of data for patterns. If found these patterns are translated into (usually rule based) knowledge or as a mathematical model. It can be seen as a learning algorithm because analysis might result in a set of rules though this is not the same learning as sketched in paragraph 4.5. In data-mining we either start analyzing a list of explanatory variables in order to predict a certain effect (implemented as a nominal class variable like “success”/”fail”) which is called supervised learning or we apply unsupervised learning where such an explanatory variable does not exist and where we are only interested in knowledge extraction. In this context we see that data-mining is not learning with respect to a predefined concept but is finding knowledge for the sake of explanation/prediction.

The demand for data-mining applications exploded in the early nineties, together with the rise of internet and the increased use of the computer as an information system where all data in a business chain is gathered in order to analyze the chain as separate parts or preferably as a whole. The need is apparent because of the huge amounts of data this generates and no clear sense of their meaning is present.

Originally data-mining is not an evolutionary technique but EC can still be applied very well to these problems. Either in canonical form, hybridized or in combination with specific data-mining techniques like bagging or boosting. If we compare the fields of data-mining and EC we see numerous similarities. Like EC, a data-mining technique is also used on an ad-hoc, rule of thumb, basis where not much theory exists on when and how to use certain building blocks making up the algorithm. As in EAs the parameterization in data-mining is a very important fine-tuning phase. We see that this is

the first main application of EAs in data-mining. As theory on data-mining does not supply us with a sound methodology on how to parameterize a model efficiently EAs can be used to do this.

A more challenging task for EAs is to guide the search for knowledge in a data-mining problem. A too straightforward way of doing this would be using a GP to try and identify a mathematical function describing the data. Of course this approach is not useful in real-world problems where we face the difficulty of multiple regressions within a set of explanatory variables, noisy data and nominal attributes sometimes mixed with numeric attributes. The generally accepted way to implement knowledge for these problems is rule-based knowledge. This can be done in the form of a tree which classifies the data according to some characteristics represented in the nodes of the tree or by if-then constructs. An EA can now implement solutions as the total set of rules in each individual, just one rule per individual or in a tree structure like in GP. For the implementation of these rules the genetic operators and their interpretation certainly needs to be altered in a similar way as in paragraph 4.5.

The practical applications consist of several applications in the health sector (see data-mining 2,3,4) where patient diagnosis is one of the most important domains. For several diseases like cancer or hepatitis it proves very hard for doctors to make a sound diagnosis with the data they gather. Computers might be able to help if for a given set of input variables like age, MRI scans or family disease history they can extract some knowledge of when the disease in question is likely to exist. Furthermore we can think of examination of general patient features, marketing, consumer decision-making and epidemiologic surveillance.

Other applications areas are found where data is gathered and an interpretation is sought, like student course performance or in businesses like stock markets and car manufacturers (see data-mining 1,5,6). We see that EC works well in data-mining too because of its flexibility. We note that EC is not purely a data-mining technique and therefore only through hybridization it can be (very) effective. We see that an EA usually fulfills the role of supervisor in the knowledge search process. In such a role it can regulate different data-mining building blocks and respective parameters in order to fit the data-mining process to its task. Furthermore we see that data taken from real-world situations *always* has considerable noise in them. Through some statistics we observe that EAs are well capable of handling such data (data-mining 2).

1. P.L. Hsu, R. Lai, C.C. Chiu. *The hybrid of association rule algorithms and genetic algorithms for tree induction: an example of predicting the student course performance*. To appear in the Journal of Expert Systems with Applications, Elsevier science, 2003.
2. K.C. Tan, Q. Yu, C. M. Heng, T.H. Lee. *Evolutionary computing for knowledge discovery in medical diagnosis*. Artificial Intelligence in Medicine, vol. 27, no. 2, p. 129-154, 2003.

3. P.S. Ngan, M.L. Wong, W.Lam, K.S. Leung, J.C.Y. Cheng. *Medical data mining using evolutionary computation*. Artificial Intelligence in Medicine, vol. 16, no. 1, p. 73-96, 1999.
4. J.H. Holmes, D.R. Durbin, F.K. Winston. *The learning classifier system: an evolutionary computation approach to knowledge discovery in epidemiologic surveillance*. Artificial Intelligence in Medicine, vol. 19, no. 1, p.53-74, 2000.
5. M. O'Neill, A. Brabazon, C. Ryan, J.J. Collins. *Evolving market index trading rules using grammatical evolution*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 343-352, 2001.
6. Nutech case study "major US automaker", see URL: http://www.nutechsolutions.com/case_studies/default.asp.
7. A.A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, 2002.

4.7. Geometrical optimization / search

Geometrical optimization is explained here as searching a space defined by some metric, usually Euclidian. The goal is placing a number of points in this space based on optimality criteria defined for the points their simultaneous behavior. This means that not the properties of all the separate points need to be optimized but the combination of points as a whole. We are aware of the fact that a lot of problems can be defined in such a context but we are only interested in applications where the point-like nature is apparent by a problem's description. Of course one could interpret a solution of a set of optimal points as the carcass of a design, or even an image analysis where the points represent features in an image. It should be clarified that the context of these problems is different (or extended) from that of placing points in an optimal way.

A well-known application of these problems lies in the field of drug/molecule design. The object here is configuring atoms, molecules or clusters of molecules in a certain predefined way. This definition embraces the goal for which this task needs to be performed. This can be for example the design of a drug that needs to fit on a certain receptor in order to work properly. For practical use of such configurations the created substance needs to be in a stable energetic state otherwise it will react and loose its desired features. Physical laws, either classical or quantum mechanical, have the ability to predict the energetic potential level a molecule is in by calculating the force fields caused by atoms that reside in each others attraction field.

In this case the placements must be done on a 3D-map which means that if a lot of atoms need to be placed (which is especially the case in drug design) the search space would explode. On top of that the more exact quantum mechanical calculation of energetic potential is complex and takes considerably more to time than conventional calculations. In drug design sometimes the problem is therefore simplified by identifying building blocks for which it is known that their energetic state is approximately zero. Through this aggregation the number of object that need to be rearranged can be decreased and with it the original problem's complexity (see geometrical optimization 1).

Other applications for these problems can be found in placing physical items on a map in a certain position, which is relative to each other. We see that for example this needs to be done at airports where airplanes need to be placed at certain specified positions or at finding lucrative locations for placing pubs, bar and restaurants (see geometrical optimization 2).

Typically these geometrical placements have a lot of constraints imposed on them making them complex problems. Also, especially in the case of molecular design, the number of placements and the precision of the placement is a huge though essential problem. EAs are proven to be very successful in these applications because of their adaptability in implementing problems with large search spaces. In the case of molecular design using a quantum mechanical calculation procedure no practical method has being developed as of yet. To enhance computational power an EC approach can be used in a distributed form, which is certainly needed if these problems are to be solved some time in the future, and is therefore an advantage of EC.

1. R.L. Johnston, T.V. Mortimer-Jones, C. Roberts, S. Derby, F.R. Manby. *Application of genetic algorithms in nanoscience: cluster geometry optimization*. In S. Cagnoni et al., editors, Applications of Evolutionary Computing, p. 92-101, 2002.
2. J. Pfalzgraf, K. Frank, J. Weichenberger, S. Stolzenberg. *Design, implementation, and application of a tool for optimal aircraft positioning*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 382-394, 2000.

4.8. Arts

We see that the creation of art is not really a problem but a human desire as a form of expression. If art is seen as the expression of the author's emotion it seems very unlike that computers can in any way produce images, sounds or texts that can be seen as art. Though if art is seen as the creation of something in some medium that has a certain aesthetical value it might be argued if computers are capable to create art. Apart from this philosophical discussion we observe that there is a need for computerized art and that some people seem to like a computer's "creativity". We will continue to use the words art and creativity to describe the artificial counterparts of the same phenomena humans express.

The creativity the computer uses to reproduce art is the same creativity (or better stochastic transformations) EAs use to search a solution space. It can be said therefore that each EA embraces this creativity although in art applications it is poured in a medium making it directly audible or visible. The crucial aspect of making art with EAs lies in the fact that the fitness function cannot be interpreted as a deterministic, unambiguous function. In the paradigm of artificial art the goal lies in making aesthetically correct pictures, sounds etc.. Because, according to common-sense, a computer has no aesthetical values this can only be done by humans. Thus a human should perform each fitness evaluation in the algorithm. This means that the aesthetic

properties of the art being created are representing the user's own aesthetic values, creating a very "personal" piece of art.

Making computerized art in this novel way certainly stresses the creativity and versatility of an EA. Judging by the commercial success such applications have and the availability of them on the web, artificial art seems to be successful though you really need to experience it yourself to find out (arts 1 and 3 are links to visual arts where arts 6 is a link to musical arts and arts 7 is a link to evolutionary poetry, the other links are here for reference).

1. A number of links to evolutionary art applications can be found at URL: http://www.accad.ohio-state.edu/~mlewis/aed.html
2. A.L. Wiens, B.J. Ross. *Gentropy: evolving 2D textures*. Computer and Graphics Journal, vol. 26, p. 75-88, 2002.
3. J.I. van Hemert, A.E. Eiben. *Mondriaan art by evolution*. URL: http://www.wi.leidenuniv.nl/~jvhemert/mondriaan
4. A.E. Eiben, R. Nabuurs, and I. Booij. *The Escher evolver: Evolution to the people*. In P.J. Bentley and D.W. Corne, editors, Creative Evolutionary Systems, pages 425-439, 2001.
5. Peter J. Bentley, David W. Corne, editors. *Creative Evolutionary Systems*. Morgan Kaufmann, 2002
6. Genetic music, the makers of the genetic music client that enables users to create their own genetic music. URL: http://kyselak.hfg-karlsruhe.de:8080/genetic-music/index.html
7. David Rea's genetic poetry site where user from all over the internet can vote on which poems will be kept in the poetical gene-pool. URL: http://www.codeasart.com/poetry/darwin.html

4.9 Other

As explained in paragraph 4.8 EC involves creativity. Typically the algorithm uses this creativity to solve a problem but from time to time authors use it to find awkward, interesting applications. As in art, there is no doubt about it that they have a use in the real world…

1. H.O. Nyongesa. *Generation of time-delay algorithms for anti-air missiles using genetic programming*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 243-247, 2001.
2. N.R. Harvey, S. Perkins, S.P. Brumby, J. Theiler, R.B. Porter, A.C. Young, A.K. Varghese, J.J. Szymanski, J.J. Bloch. *Finding golf courses: the ultra high tech approach*. In S. Cagnoni, Real-world Applications of Evolutionary Computing, p. 54-64, 2000.
3. H.-S. Kim, S.-B. Cho. *Application of interactive genetic algorithm to fashion design.* Engineering applications of Artificial Intelligence vol. 13, no. 6, p. 635-644, 2000.

4. M. Köppen, B. Nickolay, H. Treugut. *Genetic algorithm based heuristic measure for pattern similarity in Kirlian photographs*. In E.J.W. Boers et al., Applications of Evolutionary Computing, p. 317-324, 2001.

5. Conclusions

We conclude that the list of problems EC can solve is extremely large. We see that an EA is not only an optimization method (though it still is its main application area) but can search, learn or in general *be creative*[7]. As is shown in each paragraph of chapter 4, we have identified some properties of EAs that are complimentary to other algorithms used to solve the mentioned problems. If we sum them up we see a familiar list arising:

- *Creative*. Though artificial it exists clearly in the internal workings of an EA and in its way of tackling problems.
- *Adaptable*. Additional constraints can be implemented straightforward into an EA. In representations virtually any kind of structure can be implemented. Also, unlike many other methods, multiple objectives and mixing of continuous and discrete parameter can be implemented without loss of structure.
- *Easily hybridized.* EC should preferably be seen as a paradigm instead of a strict algorithm. In can consistently incorporate other non-EC related algorithms to improve performance. Such EAs are called memetic algorithms where an evolutionary process directs other search methods behavior creating an optimal resolution method. Because of the adaptable nature of EAs they are compatible with just about any other algorithm.
- *Capable of handling "out of range" problems*. Fitness surfaces are searched efficiently through balancing exploration (roughly searching general characteristic of the fitness surface) versus exploitation (local optimization in order to find the global optimum). It is even possible to explicitly or implicitly tell the EA when the balance should be shifted towards either of the search types. This can be utilized to solve so-called "out of range" problems that are impossible to solve via direct analytical techniques. This includes for example problems with many variables, many local optima or moving goal posts.
- *Robust*. Because EAs are population based the cope very well with noisy, inaccurate or incomplete data.
- *Implicit parallelism*. Unlike many other methods an EA can easily be distributed over multiple processors spreading the total load and thus creating a more efficient and powerful algorithm. Because of their population based approach one can co-evolve separate species and ultimately combine the best individuals.
- *Cheap to implement, modular*. The implementation of an EA consists mainly in building blocks regulated by parameters. Representation of a problem is independent from the solving method of the problem, see also *easily hybridized*. Usually EAs are implemented in JAVA and are highly object oriented increasing portability and reusability.

Looking at all these nice characteristics one might think we are dealing with the perfect algorithm. Apart from some fine-tuning and implementation issues, it is said to be

[7] We refer to paragraph 4.8 for a small discussion on the validity of calling a non-human creative

problem domain independent and thus the advantages sketched above can be accomplished fairly quickly. Though this description is widespread throughout EC research we stress that such statements are paradoxical. It is true that a standard EA, as we can simply see in paragraph 2.2, is problem domain independent. And besides that it does seem true that EAs have all the features sketched above. The paradox lies in the fact that on the one hand we see that if we apply a standard EA to a hard problem in the real world we see that it most definitely will fail in achieving good solutions. On the other hand if we face an easy or medium-sized problem the EA generally does produce good results but still we would still prefer other means of solving such problems over EAs. Appropriate other means *usually* have the advantage of being deterministic, extensively researched, samples of implementations are available and the solving method is seen as straightforward. Combining these two findings we see that EAs find their practical use in hard real-world problems and that for such problems domain independency just cannot be accomplished. The algorithm would give fair results but conventional means of solving would probably outperform such EAs in multiple ways. Therefore we tend to see the pursuit of domain independent EAs as a nice philosophy on a "solver of everything" but without any practical use if such algorithms always are a second-best choice. A nice quote on this from Guszti Eiben is "Evolutionary Algorithms are the second best solver for any problem" meaning that the best solvers are always domain dependent but an evolutionary domain independent approach still comes in second best.

The use of hybridisation is one of the key possibilities of implementing domain knowledge into EAs and we observe a definite increase in the use of this method. The author believes that through the use of knowledge (in the form of representations, heuristics or algorithms) that is already attained by other fields of science large performance gains can be made with relatively few resources. Because these heuristics have been practically proven to work for those *specific* kinds of problems it can be ensured that they would also work in an EC context. Because there always is convergence in EAs the hybridized result will work comparable if not much better than the original heuristic.

Theoretically through hybridization, fine-tuning of the algorithm and the creation of an optimal set of EA building blocks an EA can be the best algorithm for a (large) set of narrowly defined problems. We note that this scenario has not taken place as of yet. This is caused partly due to the problems that arise whilst trying to academically justify that the results of an EA are better and together with this problem the problem of how a problem is defined (see 2.3, no free lunch theorem) or practical problems with other justifications (see 2.3, human competitive results). Furthermore due to the fact that the potential of EAs has not yet being used to its full extent, or better: EC can evolve much more. The author believes that this potential is certainly present.

6. References

1. J. Heitkötter and D. Beasley, editors. *The hitch-hiker's guide to evolutionary computation (FAQ for comp.ai.genetic)*. URL: http://evonet.dcs.napier.ac.uk/evoweb/resources/flying_circus/tutorials/hhgtec/index.html, 1998.
2. A.E. Eiben. *Evolutionary computing: the most powerful problem solver in the universe?* Dutch Mathematical Archive (Nederlands Archief voor Wiskunde), vol. 5/3, nr. 2, p. 126-131, 2002.
3. A.E. Eiben and M. Schoenauer. *Evolutionary computing*. Information Processing Letters, nr. 82, p. 1-6, 2002.
4. D. Fogel. *Evolutionary computation: the fossil record*. IEEE Press, 1998.
5. D.L. Hull. *Darwin and his critics: the reception of Darwin's theory of evolution by the scientific community*. University of Chicago Press, 1983.
6. D.H. Wolpert, W.G. Macready. *No free lunch theorems for optimization*. IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, p. 67-82, 1997.
7. The wikipedia free encyclopedia. URL: http://www.wikipedia.org
8. Z. Michalewicz. *A Hierarchy of Evolution Programs: An Experimental Study*. Journal of Evolutionary Computation, vol. 1, nr. 1, p. 51-76, 1993.
9. A.E. Eiben, M. Jelasity. *A critical note on experimental research methodology in EC*. Proceedings of the 2002 Congress on Evolutionary Computation, p. 582-587, 2002.
10. J.R. Koza, M.A. Keane, J. Yu, F.H. Bennett III, W. Mydlowec. *Automatic creation of human-competitive programs and controllers by means of genetic programming*. Journal of Genetic programming and Evolvable Machines, vol. 1, nr. 1/2, p.121-164, 2000.
11. WordNet 1.7.1. Princeton University, 1997. URL: http://www.cogsci.princeton.edu/cgi-bin/webwn
12. The compendium of NP optimization problems URL: http://www.nada.kth.se/~viggo/problemlist/compendium.html
13. T. Bäck an H.P. Schwefel. *An overview of evolutionary algorithms for parameter optimization*. Journal of Evolutionary Computation, vol. 1, no. 1, p. 1-23, 1993.
14. EvoNet and its workgroups (STIM, IASP, SCONDI, TEL, FLIGHT, COP and ROB). URL: http://evonet.dcs.napier.ac.uk/
15. Peter J. Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufmann. 1999
16. Peter J. Bentley, David W. Corne, editors. *Creative Evolutionary Systems*. Morgan Kaufmann. 2002
17. A.A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, 2002.
18. A number of links to evolutionary art applications can be found at URL: http://www.accad.ohio-state.edu/~mlewis/aed.html