Chapter 3
Survey of EC in Dynamic Environments

3.1 Introduction

The domain of EC in dynamic environments can be broken up into four main areas. This chapter is therefore divided up into four sections so as to comprehensively detail the prior art in the field as it stands. Section 3.2 focuses on the definition of dynamic environments. It examines the different types of changes that can occur as well as the features of each type of change that differentiate it from the other. Section 3.3 identifies the various approaches researchers have adopted in attempting to tune the EC paradigm to dynamic environments. Following from this, Section 3.4 describes the difficulties encountered when trying to measure the performance of an evolutionary algorithm placed in a dynamic environment. It also covers the types of metrics adopted to date. Section 3.5 looks at the different benchmark problems explored in the literature. The chapter then continues with a review of the findings of this survey in Section 3.6.

A number of conclusions emerge from this chapter. It is found that while most of the research effort on EC in dynamic environments has been aimed at adding memory, this comes at a computational cost with mixed results. At the same time, it overlooks the idea that problem-solving competency can be built into the solution population in a general manner. The chapter finds that efforts to maintain a diverse or dispersed population are of significant importance for dynamic environments. However, of lesser importance is the field's emphasis on developing benchmark problems and metrics. Too many have been developed and all use bounded search spaces aimed at traditional GA fixed-length individuals while often also having a known optimum. These points are elaborated upon in Section 3.7.

3.2 Dynamic Problems

3.2.1 Background

A dynamic problem can most broadly be defined as a problem in which some element under its domain varies with the progression of time. Dynamic

I. Dempsey et al.: Foundations in Gramm. Evol. for Dyn. Environ., SCI 194, pp. 25–54.
springerlink.com

problems have received a growing degree of attention in recent years as authors have highlighted the fact that dynamic problems share more in common with real-world problems than static problems [78, 81, 27, 13]. Even though Goldberg first highlighted this link in 1987 [78], the momentum of research in the area only picked up towards the end of the 1990s. To date there are in the region of 200 publications available in the field of dynamic optimisation problems [59] compared to many thousands of publications in the area of EC as a whole [75].

This section will continue by surveying the relevant literature and categorising the types of problems that can occur in dealing with dynamic environments. Following from this a more detailed insight will be developed by incorporating the types of changes that can occur into a unifying table. Having analysed the characteristics of dynamic problems, the section then closes with an analysis of the differences between dynamic and static problems.

3.2.2 Dynamic Environment Categorisations

Taxonomies are useful categorisations that provide a conceptual framework for discussion, analysis, or information retrieval. A taxonomy should be discriminatory by allowing the separation of a group into mutually exclusive subgroups. This section examines two taxonomies relevant to EC in dynamic environments: types of dynamic problems and types of change.

Categories of Dynamic Problems

Trojanowski and Michalewicz (T&M), in their works [215, 216], develop a clear taxonomy of the range of problems to which EC is applied, including both static and dynamic. By splitting the problem in two and placing the solution landscape in one column and the constraints of the problem in the second, they discriminate between six categories of problems. Under the solution landscape or objective function, two possible states can be represented: either static or variable/dynamic. The constraints of a problem can then take on one of three states. They are non-existent or null, static, or variable. Table 3.1 lays out the various permutations.

The first two categories of problem in Table 3.1, represent the classes of problems to which the majority of EC research has been focused, where the objective function remains static over the lifetime of the problem. Problem categories 3–6 on the other hand, describe problems where some element in the problem domain changes, and form the area of study for this work. While this describes the types of problems that are tackled by researchers in EC, the variable elements within these categories can be broken down into the various types of change that can occur.

Table 3.1 Trojanowski and Michalewicz categorisation of problems split in two dimensions: objective function and constraints

No.	Objective Function	Constraints
1	static	null
2	static	static
3	static	var
4	var	null
5	var	static
6	var	var

Categories of Change

In comparison to categorisation of dynamic problems, categorisation of change has received more attention in the literature. De Jong derived a categorisation of different types of change in [52]. He describes four patterns of change:

i. Drifting landscapes: Here the landscape is undergoing small changes and it is the algorithm's job to track the optimum as it moves. An example under this category would be a manufacturing plant where the quality of materials may change over time, or machines may age and break down.
ii. The landscape undergoes morphological changes: This type of change includes landscapes where new regions of fitness may emerge where previously no optima resided. It is representative of competitive problems such as those seen in financial markets.
iii. Cyclic changes: This is where the change has an element of periodicity or the optima continuously reside within a certain region of the solution space. Problems under this category may include weather or political analysis.
iv. Abrupt and discontinuous changes: De Jong describes these changes as ones that may be cataclysmic and sudden, such as a power plant failure.

Branke too develops his own categorisation of types of change in [32, 33] that differ from De Jong's. He too describes four different types:

i. Frequency of change: How often does the solution landscape change?
ii. Severity of change: How far is it from the old optimum to the new?
iii. Predictability of change: Are the changes random or do they follow a pattern?
iv. Cycle length/accuracy: How accurate, and what is the period of a cycle in the solution landscape?

A final categorisation is presented by T&M [215] who propose three types of change:

i. Random changes: This describes changes that are not dependent upon the previous change or state.

ii. Non-random and non-predictable changes: These are changes that do depend on prior state and changes, but that occur as a result of relationships too complex to predict.
iii. Non-random and predictable changes: Here changes may be cyclical or follow some predictable pattern.

Of these three categorisations a certain amount of crossover and differences can be observed, with T&M presenting the most succinct classification. The T&M classification most closely constructs a taxonomy with the discriminatory factor being the degree of correlation in changes in the fitness landscape over time. Distinguishing types of change in this manner is useful as the EAs being developed will be expected to adapt their state or populations to these changes. Three different subgroups are identified by T&M. The first where the changes in the fitness landscape are Markovian in nature–entirely uncorrelated. The third where there is a predictable pattern to the change, where a correlation in changes in the landscape occurs over time and a deterministic prediction may be made. The second subgroup then is everything in between. This subgroup is large and may be refined into smaller subgroups to take into account endogenous and exogenous fitness functions. That is where algorithms take action as a result of their model or prediction and the consequences of the actions taken by the EA may not directly affect the changes occurring in the fitness landscape, or they do. The first and last subgroups only encompass exogenous fitness functions.

All three categorisations identify cyclical changes or changes that follow a pattern. De Jong and T&M could also be said to agree with regard to their identification of random changes; pattern 4 from De Jong with type 1 from T&M. De Jong describes pattern 4 as being abrupt and discontinuous indicating that it may be a change that would not have a dependency on the prior state and not leak information in the lead up to the change. Thus this satisfies the description given by T&M for their first type. This then leaves the non-random and non-predictable type of change from the T&M taxonomy and drifting landscape and morphological landscape changes from De Jong. Of these, the non-random and non-predictable type readily matches up with the morphological landscape given the example of financial markets provided by De Jong to describe this category. The final remaining type from De Jong can either be subsumed into the first or third types defined by T&M or fall somewhere in between the two. If the landscape is gradually shifting, with information on the shift's vector being leaked, then there are potentially predictive indicators that can be evolved placing this in T&M's third category. On the other hand, if information is not being yielded from the system then these gradual shifts are essentially the random changes identified in T&M's first category.

In examining the first two points from Branke, it is difficult to view frequency of change and severity of change as *types* of change. It is better rather to place them under the properties of a change that can be applied across all types of change, as they are measures of the rate and degree of the

change. T&M also suggest that change can be examined under the criteria of being discrete or continuous but this basically describes the rate at which the change is occurring, matching up with Branke's frequency description. It should be noted that even if the rate of change is continuous, for the purposes of analysis, it will be required that the data be discretised on some level so as to allow evolution to occur on a stable data set.

In unifying these different categorisations, the opportunity is then presented to provide new and more widely recognisable names for the various types of changes that occur. T&M's first category "Random Changes" essentially describes a Markovian process where the changes that occur are independent of what has preceded. Such problems may include, for example, a sudden major disruption to infrastructure in a job scheduling system. T&M's second category also matches with De Jong's second and describes a Complex System where the changes that occur are a result of relationships too complex to accurately predict, for example, where multiple parties are competing against each other with various different goals such as those in the financial domain. As already stated this category includes fitness functions that may be exogenous or endogenous in nature. An example of such a change from the financial domain that includes an exogenous fitness function is fund replication where a risk portability profile is developed to replicate the returns of a fund with more liquid securities. An example with an endogenous fitness function is the act of investing itself and attempting to uncover inefficiencies in the market. The final classification from T&M, "Non-Random and Predictable Changes" suggests that there is information available in the build-up to a change that makes it possible to explain and predict the change. The change may occur as part of a cycle, a pattern, or some incremental process, because there are dependency relationships across the variables. As a result problems with this type of change are essentially Deterministic, where the current state is dependent upon that which has gone before and past information can be used to predict future states; analysis of weather may be considered under this category. Table 3.2 displays the results of this unification.

The properties of change described above and others can then be applied in varying degrees to each type of change listed in Table 3.2. This table will be used through out this chapter to identify what issues the various authors are seeking to address in their work.

Table 3.2 Unification of De Jong, Branke and Trojanowski and Michalewicz categorisations of types of change. The unified names are given in the first column followed by the mapping from each of the authors.

	De Jong	**Branke**	**T & M**
Markov	1 & 4	3	1
Complex System	2	-	2
Deterministic	1 & 3	3 & 4	3

3.2.3 EC in Static Versus Dynamic Environments

The question to consider here is as follows: what is the essential difference between evolving solutions for a static environment versus a dynamic environment? Traditionally GAs and other EC algorithms have worked on problems where the goal has been to deterministically solve a particular problem or reach a close-to-optimal state. With regards to dynamic environments this focus shifts–the emphasis is no longer on an individual becoming a specialist or expert at a certain task. Instead despite changes in an individual's environment, the focus is to track the optimum point as closely as possible, as stated and demonstrated by Huang and Rocha [95]. In essence, the aim of an individual in a dynamic environment is to survive. In order to do so it must develop a robustness or *"plasticity"*, as described by Rand and Riolo [178], despite the fact that the changes listed in the previous section obfuscate this goal.

Branke observes that the primary issue with Evolutionary Algorithms is their tendency to *"... eventually converge to an optimum and thereby lose their diversity necessary for efficiently exploring the search space and consequently their ability to adapt to a change in the environment when such a change occurs ..."*. Yet if the inspiration behind GAs is evolution, which works in an inherently dynamic environment [178, 121], it is difficult to see how the application of artificially simulated evolution to problems is itself misguided. The problem therefore must lie in our interpretation or implementation of evolution. De Jong notes that researchers often *"over fit (their) algorithms to various classes of static optimisation problems with the focus on getting the representation and operators to produce rapid convergence to near optimal points"*. The next section will explore the avenues researchers have explored in attempting to bring the results of EAs more in line with those observed in nature: adding extra features to prevent a telescoping of diversity and making the algorithms more amenable to changes in the problem landscape.

3.3 Existing Approaches for Evolution in Dynamic Environments

3.3.1 Overview

Standard GA and GP have led to 36 human-competitive results as defined in [112] in a number of areas [75] with complexity scaling up to a billion-variable problem [80]. This clearly demonstrates the ability of an artificial-evolutionary process to produce solutions through the stochastic recombination and mutation of solutions deemed to be fit in the problem environment. When this problem environment becomes dynamic however, as far as the authors are aware, there are no published results that are comparative to the patentable works cited for static environments.

As already stated populations in GAs exhibit a tendency to converge to an optimum with a resulting loss in diversity. This impedes the algorithm from

efficiently exploring new areas in the solution space when the optimum shifts. This section looks at approaches researchers have adopted in overcoming this issue. Five approaches are examined:

i. **Memory:** Equip algorithms with a mechanism to recall previously effective solutions.
ii. **Diversity:** Prevent the noted problem of convergence within a population, resulting in the algorithm's failure to discover new solutions.
iii. **Multiple Populations:** Strike a balance between exploration and exploitation by assigning sub-populations to specific areas of the search space.
iv. **Problem Decomposition:** Break the problem down to its fundamental pieces.
v. **Evolvability:** Provide a representation that aids the population in producing offspring that are fitter than their parents.

Each approach will first be described, then analysed.

3.3.2 Memory

Of all the approaches investigated in attempting to make Evolutionary Algorithms' performance in dynamic environments comparable to that of natural evolution, the incorporation of memory into the algorithm has received the most interest [12, 29, 61, 100, 132, 131, 205, 214, 237, 238, 239, 240, 50, 49, 78, 82, 126, 144, 177, 189, 206, 95]. Within the application of this approach, Branke identified two distinct implementations [28]: Explicit Memory and Implicit Memory. Both of these are discussed next with respect to dynamic environments.

Explicit Memory: Background

Explicit memory involves the deliberate storing of genetic material in a memory cache. The philosophy behind this memory is that if a solution was useful in the past it may prove to be useful again at some point in the future. Various different strategies are employed in the selection, storage, and deletion of excess individuals to and from memory.

Louis and Xu [126] apply a memory feature to the shop scheduling problem, whereby at regular intervals of generations the best individual is stored. When the scheduling environment changes, for example, a faster machine is added or another machine breaks down, a new population is generated. However, 5–10% of this new population is seeded with individuals found in memory. The authors reported positive results with this level of seeding though saw a deterioration in performance due to either convergence with greater percentages or if the scheduling problem changed significantly (for example, if one job was deleted). Karaman et al. [100] also reinitialise their population, upon observing a change, with individuals from a stored memory.

With Ramsey and Grefenstette [177] the best performing individuals are again stored. In this case Ramsey and Grefenstette store the best-performing individuals in context at each change in the environment. They do this by measuring the state of the environment that the individual is being evolved and tested in. The state of the environment is measured through the use of a monitor module that captures the observeable parameters. When a change in the environment is recorded the population is reinitialised and 50% of the population is seeded with individuals from memory that have evolved in contexts similar to the current environment. Similarity is calculated by measuring the Euclidean distance between the observable parameters of the new environment and the previously recorded environments. Eggermont et al. [61] adopt a similar strategy and develop a case-based memory, which aims to keep track of interesting historical events and store individuals around these events. They test their solution on a periodically changing fitness function and reintroduce the stored individuals by randomly seeding the population after a change or, alternatively, by replacing the weakest individuals with memory individuals evolved in a similar context. Simoes and Costa [205] adapt the traditional GA paradigm to behave more like an immune system. The Immune System GA employs a memory population that attempts to remember prior pathogens and then selects the antibody cells that are most appropriate to battle against the pathogen in a process known as secondary response.

Mori et al. [132, 131] explore the use of a Thermodynamical Genetic Algorithm where at each generation the best individual is stored using a replacement strategy for the memory. The aim is to maintain a certain level of diversity, where diversity is the variance of bits in positions. This strategy attempts to maintain a wide representation of individuals across the solution space.

Branke [29] also follows the idea of maintaining a level of diversity in memory and adopts a dual-population paradigm. In this algorithm one population is re-initialised upon a change in the environment and the second population draws from a memory cache. The goal is to allow one population to explore potentially new areas of the solution landscape while the memory initialised population conducts search over previously discovered optima.

Bendsten and Kirk [12] create a dynamic model of memory. In their work an explicit memory population is initialised in parallel with the original population. At each iteration the individual from the memory population that is closest to the best performing individual from the solution population is selected and modified slightly to bring it closer to the solution individual on the solution landscape. This approach avoids the problems faced by others of adding to and selecting from the memory cache.

Trojanowski et al. [214] use a finite-size memory for each individual that stores successful ancestors in a FIFO queue. The better parent at each recombination is stored and the offspring inherits the memory of its parents. The memory is then employed when a previously unencountered obstacle is discovered.

Yang [237] employs a memory-based emigrants scheme, whereby the best solution in memory is used to seed new offspring and replace the weakest-performing members in the solution population. In further studies [238, 239, 240] Yang explores other forms of explicit memory. In conjunction with different forms of GAs, a population-based incremental-learning scheme stores the individuals working vector as well as the individual and an associative memory scheme.

Sebag et al. [198] introduce a negative memory with the concept of the *virtual loser*. The virtual loser is constructed by identifying the bits in the worst performers that are different from the best performers. Once this is done a flee-mutation operator is applied to new offspring that attempts to move the new individuals away from the virtual loser.

Explicit Memory: Analysis

The above approaches to the incorporation of memory into the evolutionary process are founded on the principle that it may be useful to remember previously successful solutions. Various different storage, selection and replacement schemes are explored and positive results are generally reported over a standard GA. However, problems examined also incorporate some element of periodicity, which is certain to favour any scheme that remembers previously successful solutions. The addition of memory does allow the algorithm to efficiently retrieve a previously successful solution. Through recombination with memorised individuals, or the seeding of new individuals with the genetic material of memorised solutions, it does provide a mechanism for shifting the population towards the neighbourhood of a previously discovered optimum. However this has been pointed out by Rand and Riolo [178] to have a negative effect in some cases, as it is based on the questionable premise that in a dynamic environment previous solutions will be useful in the future.

This addition of memory also comes at a cost in terms of computational effort due to the extra storage size of the memory individuals, which is generally the same size as that of the solution population. Derived from this, are the added evaluation costs of examining the memory population when a change occurs, measuring the environment, checking this metric against the stored contexts, and processing the steps involved in executing the chosen memory replacement strategy.

The intuitive use of a memory of previously discovered optimal solutions is appealing when considered for certain problems. Where a problem solution landscape is tightly bounded or undergoes a level of periodicity, the incorporation of an explicit memory, as shown by the above researchers, provides an efficient path to the rediscovery of optimal solutions or solution neighbourhoods.

The concept of an external explicit memory, however, does not readily exist in the natural evolutionary process. Though the study by Simoes and Costa [205] does present a parallel however, by placing the GA in the context of an

immune system. The purpose of an immune system is to recognise non-self cells and mark them for destruction. The human immune system is made up of two parts; an *innate* and an *adaptive*. The innate part is encoded in our genome and is static from birth until death. The adaptive part of the immune system is modified each time it encounters a new pathogen, so that it is able to recognise this pathogen again in the future [24]. However, during this process no ontogenetic process occurs and the patterns recognised by a parent's adaptive immune system are not be passed onto his/her offspring.

Implicit Memory: Background

Implicit memory differs from explicit memory in that it does not utilise an external memory cache of previous solutions or states, negating the need for extra selection, replacement, and storage strategies. Implicit memory instead is incorporated into the individuals of the population through degenerate genetic material. A number of authors have adopted different approaches in the creation and utilisation of implicit memory though most researchers focus on the development of multiploidy representations. Multiploidy is where more than one allele is used to represent each individual. Whereas in standard evolutionary computation representation one allele is used to represent the solution, this is known as haploid representation. Multiploidy includes a multitude of alleles to express a phenotype and diploid is where just two alleles are used. The use of a multiploidy representation requires some mechanism to enable the dominance of one allele over the other. The literature expresses almost as many dominance mechanisms as researchers investigating the mechanism.

Goldberg and Smith [78], and Smith [206] produced the earliest publications on a diploid representation for improved evolutionary results in non-static problems. Their results demonstrated that a diploid structure with an evolving dominance map provides superior results over a haploid representation. They show that the diploid structure is able to maintain alternating alleles in abeyance, thus affording a quicker evolutionary process to an alternative optima. Hadad and Eick [82], and Ng and Wong [144] also present diploidy representations with varying dominance mechanisms.

Dasgupta and McGregor [49, 49] implement a multiploidy approach where genes at higher levels are able to activate or deactivate the genes at the immediate lower level enabling a hierarchical structure to be evolved.

Ryan [189] develops a multiploidy representation where the dominance is performed through the summing of genes for a particular trait. If the result exceeds a certain threshold the phenotypic trait is 1, or 0 otherwise. This methodology was extended in Ryan and Collins [191], by introducing a lower threshold. Where the summation was below this the phenotypic trait was 0, in between the two bands the trait was generated randomly, and above the higher threshold it was 1, as in the previous study. This methodology produced superior results to the prior study.

Osmera et al. [170] found that for highly multi-modal problems the diploid structures were able to outperform their haploid counterparts. Dominance in this case was performed through an XOR operation between chromosomes. Callabretta et al. [37], similar to Osmera, also employ an XOR mechanism. In this case the first bits of the alleles are XOR-ed. If the result is 0, then the allele with the second bit equal to 0 is expressed, with the same algorithm being used if the result of the XOR is 1. If the second bit in both is the same, then the alleles are co-dominant and an average between them is used.

Collingwood et al. [45] adopt a mask to determine dominance. This specifices which of the chromosomes is dominant independent of which allele is present in that chromosome. Each individual in the population possesses its own mask that is also evolved through the application of genetic operators during reproduction.

Kim et al. [243] use a *winner take all* approach whereby each chromosome is evaluated with the phenotype being expressed, derived from the fitter chromosome. In this case reproduction occurs among the dominant chromosomes of parents to generate one offspring, and the recessive chromosomes to generate another.

In, [223], Uyar and Harmanci conduct an analysis of performance of different genotype-to-phenotype mapping schemes. However, having completed this survey unsatisfied they adopt their own dominance mechanism [224] that aims to maintain a balance between exploration and exploitation. It does this through adaptation of the dominance mechanism as a result of feedback from current phenotypic output. Lewis [119] also conducts a survey of multiploidy techniques and concludes that a simple multiploidy scheme does not provide a strong advantage over a haploid GA. When equipped with a dominance mechanism, multiploidy algorithms do perform better though similar to a haploid GA, which throttles mutation when a degradation in fitness is observed. For diploid representations, such as that presented by Ng [144], Lewis comments that their utility is largely limited to functions that oscillate between two optima.

Though the exploration of multiploidy and dominance mechanisms has received the majority of attention from researchers for the incorporation of degenerate genetic material other methods also have been examined.

Ohkura and Ueda [165, 166], present a string representation that contains inactive regions. These inactive regions allow for the possibility of neutral mutations to occur. Over the course of the evolutionary process active and inactive regions will change. They will present a higher level of diversity across the population along with the incorporation of memory in inactive regions that can be reactivated by evolution when a change occurs.

Huang and Rocha [93, 94, 95, 181, 182] explore the use of RNA editors in their studies. RNA editing occurs during the mapping process from genotype-to-phenotype and applies stochastic variations to this process. This results in genotypically equal candidates producing different phenotypes, meaning

a many-to-one mapping from genotype-to-phenotype and vice versa. RNA editing in nature occurs in the development process of an organism and so is not ontogenetic, meaning the changes that occur in the mapping process are not inherited by offspring. Huang and Rocha report greater plasticity and robustness of solutions as a result of the RNA editing over traditional GAs. The RNA editors can be classified as implicit memory because in [95] they develop an agent-based model where each individual evolves and possesses its own RNA editor.

In [148], O'Neill highlights the presence of degenerate genetic material in Grammatical Evolution. In this case a genotype-to-phenotype mapping occurs that involves a many-to-one relationship and, like the studies by Ohkura and Ueda, the ability to conduct neutral mutations is also present. Though GE has largely only been tested on static problems, with the exception of [156], considering the prior studies examined here it would suggest that these features give GE a plasticity of solution. The genotype-to-phenotype mapping in GE is discussed in greater detail in Chapter 2.

Implicit Memory: Analysis

The concept of diploid or multiploidy in EC is inspired by similar representations in the natural world. Each human cell contains 23 pairs of chromosomes, one from each parent. This cell structure is considered to be diploid. Different strands of chromosomes then, have the attribute of being either dominant or recessive and it is this relationship, for example, that dictates whether we have brown or blue eyes.

In surveying the work to-date on exploring the potential implementations of Implicit Memory, it is clear that a multiploidy representation has motivated the majority of researchers' efforts with mixed results. Comparative studies on multiploidy representations with different dominance schemes have demonstrated that their performance is equivalent to that of haploid GAs with higher or adaptable mutation rates. The central criticism of Lewis' study is that the multiploidy representations he examined were essentially only useful for problems that alternated between a couple of solutions [119], while Branke also makes a similar observation [27].

While this problem could also be considered an advantage in applying the *"horses for courses"* addage – in that there are certainly problems in the real-world to which a diploid structure may be applied quite successfully – there is still the problem of selecting a suitable dominance function. Of the work surveyed, no clear solution was presented that had received any degree of traction. This is evidenced by the diversity of dominance functions. It should also be noted that the representation of multiple versions of the genotype for each individual also comes at a computational cost.

Ohkura and Ueda, and Huang and Rocha provide alternative implementations of Implicit Memory, where redundant genetic material is not stored

in a diploid or multiploid structure thereby negating the need for a dominance function. With Ohkura and Ueda, the degenerate genetic material is represented by inactive regions on the genotype presenting a resilience to destructive mutations, while also providing memory when those regions are reactivated. Huang and Rocha develop an Implicit Memory mechanism with the addition of stochastic RNA editors. These editors allow a single individual to present multiple phenotypes each time the RNA editor is run, favouring one phenotype over the other by adapting the probabilities of the rules the RNA editors use.

While the representations expressed by researchers in the previous section do have a strong foundation in the natural world, the development of the various dominance functions does not mirror the natural parallel to the same degree. This may be partly down to the complexity of dominance in the natural world where there may in some cases be partial dominance or also co-dominance. Such variations present challenges when attempting to implement the relationship in algorithmic form.

Of course, what Explicit Memory and the Implicit Memory outlined here do not cover is the fact that there is also an Implicit Memory inherent in the population as a whole. The memory resident in the population itself becomes more important in dynamic environments, especially where changes are of a more gradual degree. Under such circumstances previously optimal solutions may slip in the rankings and stand less of a chance of passing on their genetic material once the optima have moved. However, they do provide the population with a means to quickly reproduce older useful solutions if the environment reaches a state similar to one previously visited.

Reflecting on the categories of change identified earlier in Table 3.2, it can be seen that through the incorporation of memory into the GA, the authors were seeking to address two types of change, Complex and Deterministic. In the case of Complex changes memory of past events can prove to be useful as they may give insights into future changes where similar conditions exist. Where the changes are Deterministic in nature, cycles or patterns may exist and previously discovered solutions can be readily recalled. In situations where the type of change is Markov, memory does not play as important a role as these changes are random and remembering previous states or good solutions will not aid in an algorithm's search.

Table 3.3 The addition of memory is aimed at helping systems remember past useful solutions for problems where there may be cycles or patterns

Problem Type	Memory
Markov	-
Complex System	X
Deterministic	X

3.3.3 Diversity

GAs can be considered to be parallel problem solvers, where a population of solutions spread out across the solution space attempt to find and converge to a global optima. While in the initial dispersed state, a wide coverage of the solution space is achieved with many local optima being explored. However, as the population converges, this search narrows in focus. This problem has been alluded to by a number of authors [32, 205, 74, 81] especially with respect to dynamic problems. Diversity itself can be measured in a number of ways [36]: through explicitly counting the number of distinct genotypes, phenotypes, and fitnesses; using entropy [185]; or implicitly, by monitoring the time-averaged-fitness of the population and associating a drop in this average with a narrowing of diversity.

Diversity: Background

As mentioned above, a common criticism of GAs is that the population converges to an optimum, leading to a drop in diversity and making the discovery of new optima difficult when the environment changes. Increasing the level of diversity after a change has occurred, or been noticed, is one way of addressing this issue. However, this fails to take into account the clustering of a diverse population and so dispersion of the population must also be considered.

The most popular solution is *hypermutation*, proposed by Cobb [44], where the rate of application of the mutation operator is increased at certain times. Cobb uses hypermutation to maintain an acceptable level of time-averaged best performance. When this average performance drops below a threshold, hypermutation is applied with the aim of maintaining a balance between exploration and exploitation. The implication of this increased rate of mutation is that a higher level of diversity is achieved as opposed to what could be achieved through crossover alone. The issue being addressed by hypermutation is that in a converged population, crossover between parents fails to increase diversity leading to the need for greater mutation.

Vavak et al. [225] also examine the time-averaged best performance as the trigger for the application of their *variable local search* (VLS). When this occurs, crossover and mutation are temporarily suspended. VLS is conducted by applying a shift bit register to the individuals selected for reproduction. The bit register is made up of bits that are set randomly and the value represented by this register is either added to or subtracted from the bit string of the individual. This enables a local search around the location in the search space represented by the solution. The register is applied to each member of the population and then the GA resumes with its normal operators. If performance is not improved after a number of generations and/or a period of time, the VLS is again applied with an extended area of search, in other words, the register is longer allowing a wider boundary. In [226], Vavak et al.

extend their study of VLS with the introduction of an incremental learning technique for controlling the boundary of search automatically.

As an alternative to the monitoring of time-averaged fitness, the explicit maintenance of diversity over the course of evolution also addresses the issue. Mauldin [128] presents an early example of the maintenance of diversity for static environments through setting an adaptive threshold of Hamming Distance between bit strings for new individuals to be introduced to the population, with this threshold decreasing as generations progress. Mauldin observed that this methodology produced more robust solutions when applied to the out-of-sample data.

Grefenstette [81] maintains diversity by introducing the concept of replacing a certain precentage of the population at each generation with randomly generated individuals, a process that has become known as *random immigrants*. This process maintains a level of diversity throughout a run by potentially introducing new genetic material to the population at each generation.

Cedeno and Vemuri [39] employ a *Multi-Niche Crowding GA* that promotes reproduction among solutions sharing similar traits, which collectively could be described as occupying a niche. The replacement strategy that is then adopted replaces the worst-performing individuals in this niche. This has the effect of essentially preventing a speedy convergence in the search space by encouraging similar solutions to reproduce in their niche.

Ghosh et al. [77] keep track of the ages of individuals in the population. In doing so they are enabling a selection bias based on the age of the individual in conjunction with its score derived from the objective function. The age-scoring function adopted in this study intuitively favours the middle aged over the young or old. Such a scheme provides less of a chance for selection of poor-performing young and old solutions.

Mori et al. [132, 133] aim to maintain a temperature or a level of free energy in the evolution of a population through the use of a Thermodynamical Genetic Algorithm (TDGA). This algorithm maintains this level with the expression

$$F = E - TH \tag{3.1}$$

where F is the energy level of the population, E is the average fitness of the population, H is the diversity of the population and T is the weighting placed on diversity. Individuals are introduced into the population after reproduction in such a way as to maintain a high level of F.

Simoes and Costa [204] develop a new genetic operator known as a *transformation* whereby genetic segments from previous solutions, which have left the population, are stored in a pool. These segments are then inserted into individuals at reproduction replacing the crossover genetic operator and avoiding the problem of crossover in a converged population.

In [135] Morrison employs *sentinels*, which are individuals uniformally dispersed [134] in the search space. These sentinels remain statically located

and play normal roles in selection and reproduction, while also providing the algorithm with the ability to maintain a level of diversity that is dispersed throughout the search space. When a radical shift in the environment causes the optimum to move to a different area, a sentinel will already be located in the general neighbourhood to provide genetic material that will help in moving the general population towards this new position.

Diversity: Analysis

Reflecting on nature where almost every animal on the planet possesses its own unique genetic makeup, diversity is seen to play a central role in the natural environment. It is this diversity that enables populations to produce solutions to new diseases or changes in their environment, where each problem is tackled in a massively parallel manner through a multitude of unique genetic makeups. In contrast, studies have shown [231, 127] that inbreeding and a reduction in diversity can have a negative effect on the fitness of organisms in nature, lending credence to approaches that place an importance on diversity in evolutionary algorithms.

Branke [27] identifies two ways in which researchers use diversity as an aid to EAs for dynamic environments: to increase diversity after a change has occurred and to maintain a level of diversity throughout the run. The maintenance of diversity can be further broken down into the active maintenance of diversity and the pre-emptive maintenance of diversity. Both Cobb and Vavak's solutions fall under Branke's first classification. In their studies, a time-averaged drop in fitness is used as an assumption that the population has converged and is struggling to explore new solutions. This approach, however, may suffer in real-world applications. The reason is that if it has become noticeable that the time-averaged fitness has dropped below a certain threshold, the system is already failing or underperforming in its task. Also implicitly associating fitness with diversity, instead of monitoring or maintaining it directly, may mask other issues. The manner in which these two solutions are applied, whereby the degree or rate of application is increased with a drop in fitness, suggests that these solutions are essentially temporary fixes or bandages on the more fundamental problem of maintaining diversity and exploring new areas of the search space.

Maintaining a level of diversity throughout the life cycle of a system certainly addresses these problems. Simoes and Costa, and Grefenstette continuously introduce new genetic material to the population by recycling extinct genes and introducing newly generated random material. Such mechanisms are necessary to prevent the problem described earlier, whereby crossover in a converged population does not yield new genetic material. Ghosh et al., and Mori pre-emptively solve the problem through selection and replacement strategies that attempt to prevent any individual from taking too strong a

hold of the population and dominating the genetic pool. Morrison too pre-emptively maintains diversity in his contribution, but more than this Morrison also ensures the population maintains a level of dispersion throughout the search space. This addition prevents a population, which may be genetically diverse, from clustering around one area of the search space and therefore provides the algorithm with the ability to maintain a dispersed coverage of a bounded search space. Table 3.4 summarises the classifications.

Table 3.4 Authors and the three approaches to diversity: increasing diversity, actively maintaining diversity, and pre-emptively maintaining diversity

Increase	**Maintain Active**	**Maintain Pre-emptive**
Cobb	Simoes & Costa	Ghosh
Vavak	Grefenstette	Mori
-	-	Morrison

This considered, an issue that largely escapes attention in the EC literature is whether in fact the drop in diversity is a result of too much evolution being conducted. Branke [33] attempted to measure change severity that could potentially have led to an evolution-throttling mechanism, though the developed metric proved to be inaccurate. Dempster and Jones [57] monitor for convergence in their population and stop evolution when the fitness of a number of the most fit individuals fails to improve by no more than 1%. This issue is potentially a legacy problem from static environments, where evolution is used as an optimiser and in a sense to overfit to a particular problem.

Ensuring a population is adequately diverse aids the algorithm in discovering new solutions in areas of the search space that before had provided poorly fit individuals. In continuing the development of Table 3.3, diversity is tailored to aid in the navigation of two types of changes, Markov and Complex Systems. For Markov type changes maintaining a dispersed and diverse population can ensure that some member will be in a relatively fit neighbourhood when a change occurs. With complex types of change the model of change is unknown and is also subject to change. Maintaining a diverse population in this case enables the system to discover relatively fit neighbourhoods as the optima move and also to evolve innovative solutions from areas of the solution space that had not previously been explored. Deterministic problems in this case are not specifically catered to. Yes, as in all problems, a level of diversity is necessary to adequately explore the solution space. However, in this case a converged solution is sought in uncovering underlying pattern or structure and the maintenance of a diverse population carries less weight.

Table 3.5 Diversity enables the algorithm to discover new solutions in Markov and Complex System problems

Type	Memory	Diversity
Markov	-	X
Complex System	X	X
Deterministic	X	-

3.3.4 Multiple Populations

Again addressing the issue of population convergence, researchers have adopted a separate strategy different to that of maintaining diversity in a population through the creation of multiple populations by partitioning the main population. The aim here being that separate multiple populations can exploit discrete areas of the solution landscape. This is based on Wright's (1932) Shifting Balance Theory (SBT). In this theory, Wright recognised that within a population premature genetic convergence can occur. However, if the population is split up into smaller subpopulations, genetic drift allows these discrete subpopulations to exploit new areas on the fitness landscape. A GA first developed along the lines of this theory is known as the Forking GA and was first developed by Tsutsui et al. [218] for static environments.

Multiple Populations: Background

Oppacher and Wineberg [167] developed the first SBT inspired GA for dynamic environments. Key to the implementation of such a system is the incorporation of a mechanism for meaningfully dividing the subpopulations. Oppacher and Wineberg achieve this through use of Hamming Distance. If a particular colony population overlaps the core population, extra evolutionary pressures are activated to push that colony away from the core population. Migration is then conducted from the colonies to the core population in a stochastic manner, based on a colony's fitness. In this case, the core population is responsible for the exploitation of best solutions, while the colonies explore the landscape for new promising optima.

Branke et al. [30] approach the problem from a slightly different angle through the Self-Organising Scouts model. Here, the larger base population searches the landscape for new undiscovered optima and assigns a scout subpopulation to areas which show promise. At each generation, an analysis of the population is conducted in order to determine whether there exists a group of individuals that are clustering around a phenotypic optimum. If this is the case, this group is separated from the base population and undergoes evolution only within the group to fully exploit the area.

Multinational GAs developed by Ursem [222] identify groups of individuals attempting to form nations. A nation is formed when a fitness valley exists between the most fit individual in an old nation and the most fit of a poten-

tially new nation, calculated by evaluating random points between the two individuals. A variable mutation rate is also adopted within nations where individuals on the periphery undergo a higher rate of mutation compared to the most fit individuals within the nation. While this method did provide evidence of being able to track multiple optima, it suffered from a lack of diversity on a global level.

Another method of splitting up the population is explored by Ronnewinkel and Martinez [184] based on a fitness envelope defined by Petrowski and Genet [171] where fitness envelopes are formed around centroids. This method has benefits over the previous paradigms as it does not require extra fitness evaluations to determine valleys and also does not require parameters for maximal distance around a centroid to encompass its colony.

Multiple Populations: Analysis

Through splitting up the population into subpopulations charged with exploring different areas of the fitness landscape, a GA can exploit multiple optima simultaneously. The number of potential optima to be explored, however, is limited by the minimum subpopulation size and the global population size. As demonstrated by Ursem, diversity can still become an issue with the population potentially converging to a range represented by converged subpopulations.

The function of multiple populations is to strike a balance between exploration and exploitation. In this regard its intention is generally applicable to search and as such is not catering to specific types of change in dynamic environments. Indeed, its benefits also extend to static environments.

Table 3.6 Using multiple populations aids search in general as it tries to strike a balance between exploration and exploitation

Type	Memory	Diversity	Multiple Populations
Markov	-	X	X
Complex System	X	X	X
Deterministic	X	-	X

3.3.5 Problem Decomposition

The act of problem decomposition is the breaking up of a problem into its more fundamental constituents. This relates to the building block hypothesis [89, 79], which states that GAs work well when short, low-order, highly-fit schemata or building blocks, recombine to form even more fit higher-order schemas. While analysis of this theory has largely been conducted for static environments [227, 69, 208], its importance is also carried through to dynamic

environments where the formation of robust generalised solutions goes hand in hand with problem decomposition.

Problem Decomposition: Background

Abass et al. [1] approach the problem of evolution in dynamic environments by assuming that the most efficient way of tracking an optimum in a changed environment is to extract structural knowledge about the problem from previously successful solutions. Using the information theoretic measure, the minimum description length (MDL) [180], a probabilistic decomposition model is established for the structure of the problem and manipulated as the environment changes. This model is then used to identify building blocks and to conduct crossover along the partitions of these building blocks. In their work Abass et al. maintain the decomposition model across changes in the environment, but restart their population of solutions at each change.

In Ronnewinkel et al. [183] a theoretical examination of a GA is undertaken, with mutation as the only genetic operator. Where regular changes occur in the environment, defined as changes that have a fixed duration and obey some deterministic rule that is the same for all change cycles, Ronnewinkel et al. observe the correspondence of their model to Eigen's *quasi-species* [62]. A quasi-species is a generalised self replicating entity that can be represented by a small number of building blocks. New genetic material is arrived at through mutation, which occurs during the copying process. The identification of the quasi-species model in a GA provides evidence of problem decomposition by identifying atomic problem building blocks that allow the algorithm to more efficiently adapt to changes within the problem's environment.

Problem Decomposition: Analysis

While the majority of research conducted on the building-block hypothesis and problem decomposition has focused on static environments, its use and understanding is of equal or greater importance for the construction of robust solutions for dynamic environments. Eigen's evidence of problem decomposition in the form of the quasi-species in a natural setting reinforces the importance.

The identification of a quasi-species and building blocks for problem decomposition also ties into one of the approaches discussed earlier: memory, or implicit memory more exactly. As a population evolves and begins to learn the structure of a problem, this information gets incorporated into the genetic structure of the individuals. Therefore, it operates as a type of memory and provides the population with a level of competency in its problem domain. This implicit memory, or population competency, then intuitively leads to the generation of solutions that will be more robust to changes.

For problem decomposition, the intention is to evolve a robust, competent population of solutions. This necessitates that the problem domain offer some information that may be analysed to provide an insight into how it is changing. In Table 3.7 problem decomposition is seen to aid with two problems. In Complex and Deterministic changes information may be extracted from the domain, providing insights to the nature of the changes or the structure of the solution. On the other hand, a system's response to a Markov type of change is essentially reactionary, as prior information or state does not aid in predicting the change.

Table 3.7 Problem decomposition requires the presence of information and dependencies in the domain

Type	Memory	Diversity	Multiple Populations	Problem Decomposition
Markov	-	X	X	-
Complex System	X	X	X	X
Deterministic	X	-	X	X

3.3.6 Evolvability

Where problem domains are dynamic, the ability of a population to evolve to a new area in the solution space becomes key. Navigating from a local optimum to a global optimum or following a moving optimum is more efficient when the representation or paradigm used makes this process easier. Providing a high level of evolvability may allow the migration to a new, better optimum without the population or individuals first experiencing a drop in fitness as they traverse the fitness landscape. In natural evolution, evolvability is described as the capacity for an adaptive response to a dynamic environment [83].

Evolvability: Background

Altenberg [3] approaches the issue of evolvability from a theoretical standpoint in a static environment and examines the *constructional fitness* of blocks of code. The constructional fitness is the probability that a block of code will increase an individuals fitness if it is added. A genetic operator then, based on this premise, will become focused on the dimensions of variability with a greater potential for increasing fitness. Altenberg recommends Upward-mobility selection (a type of steady state selection) and Soft Brood selection (a tournament selection is applied to the offspring before they are added to the population) as means of improving proliferation of constructionally fit blocks through the population. For recombination he advises the use of a type of genetic engineering where blocks of code are selected from a library

which has a rated catalog of blocks based on their historical performance in increasing an individual's fitness after insertion.

Risinger et al. [179] note that in dynamic environments more evolvable representations are better able to survive in the long-term. With this in mind, they set out to measure the evolvability of three different representations on a dynamic fitness function: a direct encoding, a modified direct encoding with parameters affecting how mutations should occur, and Genetic Regulatory Network (GRN) that uses a genotype-to-phenotype representation. The GRN presented highest level of evolvability due to its ability to conduct neutral mutations which enabled it to maintain a more diverse population and more easily switch to a new solution.

Shipman et al. [201] seek to harness the benefits of evolvability through the use of a many-to-one genotype-to-phenotype mapping, which is applied to the real-world dynamic problem of planning for the growth of a telecommunications network. They observe that such a representation allows for the potential of neutral drift that can migrate a population out of a local optimum. The GRN representation they adopt allows this, as the evolutionary search can continue in the genotypic space through neutral mutations while the underlying fit phenotype is maintained.

In Ebner et al. [60] the work of Shipman et al. is extended through drawing a link between phenotypic performance of an indirect encoding and the existence of neutral networks in the genotypic space. These neutral networks are identified by Engelhart and Newman [140] in the natural world where neutral mutations allow the maintenance of genotypic diversity with genes spread out along a "neutral" network while still mapping to a valid fit phenotype. This provides the ability to easily evolve to a new, fitter phenotype when circumstances change as the search space widely covered at the genotypic level. Ebner et al. examine the reachability of phenotypes, the innovation rate, connectivity between phenotypes and the extent of neutral networks where a direct encoding representation is also used as comparison. Across all metrics the indirect encoding provided far superior results due to it possessing the ability to easily reach other phenotypes with small shifts in the genotypic space.

Evolvability: Analysis

Providing an evolutionary algorithm with a strong level of evolvability gives it the potential to incrementally improve its population fitness without first experiencing a decrease as it escapes a local optimum. Where the environment is dynamic a high level of evolvability allows the population to more easily track a moving optimum. Indeed, where the environment is dynamic, a high level of evolvability is rewarded in the evolutionary process, as it has a greater chance of survival.

Altenberg focused on a direct encoding in a static environment and as such the selection and recombination recommendations made suggest a high rate of convergence. Under such schemes it may be difficult to shake out

of a neighbourhood where a target moves. However, in the representations examined above, those with an indirect mapping are implicitly coupled with another desirable attribute: a potentially higher level of diversity in the genotypic space. Because of the many-to-one mapping, individuals may spread out along a neutral network in the genotypic space providing the high levels of connectivity described by Ebner in the phenotypic space. Combined with this, such a representation also facilitates neutral mutations.

In adopting a many-to-one genotype-to-phenotype mapping a greater level of evolvability can be achieved which parallels features observed in nature. For dynamic environments the ability to easily reach another solution is imperative. Whether the type of change is Complex, Markov or Deterministic, providing good evolvability essentially lubricates the evolutionary process.

Table 3.8 Evolvability is key across all dynamic problems where the ability to adapt and improve fitness in a changing environment is necessary for survival

Type / Type	Memory	Diversity	Multiple Populations	Problem Decomposition	Evolvability
Markov	-	X	X	-	X
Complex System	X	X	X	X	X
Deterministic	X	-	X	X	X

3.4 Evaluation of Performance

3.4.1 Evaluation of Performance: Problem Description

In static-environment problems, comparing the performance of different algorithms is not typically difficult. Comparison of final fitnesses and the rate at which they are achieved are readily available and serve both quantitative and qualitative purposes. Where the environment is time variant, the final fitness only represents one snapshot into the life cycle of the dynamic process; it does not capture the performance of the system over time.

For static environments Feng et al. [66] formalise a number of metrics examining:

- Optimality
- Accuracy
- Sensitivity
- Convergence and
- Optimiser overhead.

Of these, Optimality and Accuracy may be used as measurements of a dynamic algorithm's quality through taking snapshots at regular intervals. However, these measurements, as defined by Feng et al., assume that the global

optimum is known and that the solution space is bounded. These two constraints may render the metrics unusable when applied to real-world dynamic problems.

3.4.2 Evaluation of Performance: Metrics for Dynamic Environments

De Jong [51] identifies two types of performance set in static environments: on-line performance and off-line performance. On-line performance is the performance or average fitness of the best solution during training on the in-sample data. Off-line performance is the fitness of the best solution on the out-of-sample data. Even though intended for static in-sample/out-of-sample tests, these types of performance bear relevance in dynamic environments where, during on-line training, the goal is to track the moving optima as closely as possible. Then when the system is live on new data, tracking the off-line performance becomes necessary.

Mori [132] derives a metric that produces the average ratio of best-evolved fitness to the global optima over time. Its equation is:

$$M = \frac{\sum_{t=1}^{T_{max}} \frac{f_{best}(t)}{f_{opt}(t)}}{T_{max}} \tag{3.2}$$

where T_{max} is the length of the search process or the number of time steps in the search, $f_{best}(t)$ is the best solution at time step t, and $f_{opt}(t)$ is the global optimum at time step t.

Trojanowski and Michalewicz [215] develop two metrics specifically for dynamic environments that measure accuracy and adaptability. The accuracy metric measures the difference between the current best solution before a change in the environment and the optimum value averaged over the entire run. It is given by the expression:

$$Accuracy = \frac{1}{K} \sum_{i=1}^{K} (err_i, \tau - 1) \tag{3.3}$$

While the adaptability metric measures the difference between the best individual at each generation and the optimum value averaged over the entire run, it is expressed as:

$$Adaptability = \frac{1}{K} \sum_{i=1}^{K} [\frac{1}{\tau} \sum_{j=0}^{\tau-1} (err_{i,j})] \tag{3.4}$$

where $err_{i,j}$ is the difference between the value of the current best individual of the j^{th} generation after the last change and the optimum value after the

i^{th} change. τ is the number of generations between two changes and K is the number of changes of the fitness landscape during the run.

Besides these metrics, other measurements of performance have been developed by charting differences between best performing individuals over time relative to the global optimum at the same time.

3.4.3 Evaluation of Performance: Analysis

A common thread through these metrics is the incorporation of the global optimum into the calculations. Unfortunately, however, in real-world dynamic problems it is often the case that a non-deterministic problem is being analysed. Under such circumstances, the comparability of the metrics across different problems is not possible. Indeed, the utility of comparing accuracy and adaptability across different problems is indeed questionable in the first place. For example, results achieved in comparing a symbolic regression system on an unbounded random walk, and a simple linear function will produce vastly different performances.

In examining the performance of algorithms on real-world dynamic problems, often the domain in question will possesses a wealth of relevant metrics. Indeed, such metrics produce a more relevant analysis of performance as the results can be compared against acceptable norms in the domain or in the performance of human competitors. In the domain analysed in this book, which is finance, a wide variety of metrics are used to analyse both the performance and behaviour of the subject system.

3.5 Benchmark Problems

Benchmark problems are used to provide a common baseline in understanding the behaviour of new algorithms or extensions to existing algorithms. The problems are generally straightforward to reproduce and allow researchers to gain a relative perspective of their work compared to others. According to Branke [29], benchmarks should be:

- Easy to describe
- Easy to analyse
- Tunable in their parameters and
- Strike a balance between complexity and simplicity.

With the aim of satisfying these criteria, Branke introduced the Moving Peaks benchmark. This benchmark is similar to Morrison and De Jong's [134]. The Moving Peaks benchmark is a multivariate problem where multiple peaks of varying height and width shift and change their dimensions after a number of generations. The goal here is to locate the highest peak.

Yang [235, 236, 237] produces a dynamic benchmark suite that varies in difficulty using a decomposable trap function. This problem is also multi-variate,

as the function decomposes to constituent building blocks and the difficulty of the problem can be changed on line.

Huang and Rocha [95] examine their system using an oscillating Royal Road benchmark along with trials on an Optimal Control Testbed, which crosses into the domain of engineering sciences.

Rand and Riolo [178] conduct benchmarking tests using Shaky Ladder Hyperplane-Defined functions that, like Yang's benchmark, are designed with building blocks in mind and are based on schemata.

de Franca et al. [70] develop a suite of dynamic problems based on the work of Angeline [6], where optima are displaced every n generations.

Many other researchers examine their algorithms on time-varying knapsack, oscillating, and scheduling problems.

3.5.1 Benchmark Problems: Analysis

Despite the best efforts and intentions of a number of authors, the number of *benchmark* problems in use, like dominance functions in Section 3.3.2, remains as varied as the number of researchers involved. This then has the negative consequence of placing the reader in the situation where benchmarks are being compared ahead of the actual algorithms.

Hidden by this problem is the question of what the researchers are actually aiming to achieve. If the long-term view is the application of algorithms to real-world problems, there exists a very real danger that algorithms are being optimised to solve benchmarks and not the relevant problem at hand. Added to this, a wide range of real-world problems from a multitude of domains exist and it is naive to assume that a particular benchmark can capture the various intricacies involved in each.

Benchmark problems certainly have their use as logical unit tests for extensions to algorithms in order to gain fundamental understanding of their behaviour. With this in mind, different benchmarks should simulate the various categories of change outlined in Section 3.2.

3.6 Chapter Review

The core aspects of EC in dynamic environments have been examined. The chapter began with an examination of dynamic environments and how they differ from static problems. This was followed by an attempt to classify and unify independent categorisations of the types of changes that occur while also identifying a number of properties of change.

Following from this an in-depth analysis was made of the various approaches adopted by researchers in advancing Evolutionary Algorithms to produce better performance in dynamic environments. The first approach analysed was the addition of explicit memory to algorithms. This generally

involves the inclusion of a cache or extra population to serve as a memory for previously effective solutions. The philosophy being that these solutions may prove to be useful again if the environment returns to a similar state, or their genetic material can provide a source to help direct the search neighbourhood. While results presented by authors do demonstrate improvements in benchmarks, the approach comes at a computational cost and has the undertones of being a brute-force algorithm. Indeed it begs the question, why not store all good solutions? It also rests on the assumption that previously learned optima may be useful, or returned to, in the future.

Implicit memory differs from explicit in that information is stored in degenerate or redundant genetic material. The favoured implementation in the literature is through the incorporation of diploid or multiploid genetic structures for the individuals. This enables the individual to store previously good information and swap quickly when a change occurs. This paradigm was seen to be limited in its success. It is applicable predominantly to problems that oscillate between a small number of states while still incurring a computational cost, and does not demonstrate any clear cohesive approach to the subject of dominance functions. It is noteworthy that both the explicit and implicit multiploid forms of memory come at a computational cost when one of the aims of EC in dynamic environments is to reduce computation cost by adapting a population, rather than restarting it from scratch, with each change. However, still under the umbrella of implicit memory, flexible, computationally-efficient methods are demonstrated through the use of a many-to-one genotype-to-phenotype mapping process, along with the use of degenerate genetic material in a linear-string representation. These methods were found to be more adaptable to multiple states over a standard EA, while the efficient representation meant that memory was maintained at little computational cost. Such a representation with a many-to-one mapping also addresses Eggermont's criticism [61] that the population, which is supposed to contain the history of evolution, does not keep enough information to allow the algorithm to react adequately to changes.

In examining the role of diversity, two paradigms were also observed. Examined first was the increasing of diversity triggered by a drop in performance. This approach while serving the purpose of increasing diversity, suffers in application, where the lag inherent in the triggering mechanism can mean a period of sub-par performance before recovery. The second approach, simply maintaining the level of diversity, is achieved in the literature through the use of a number of different strategies for both actively maintaining the level of diversity and pre-emptively maintaining it. None of these strategies, however, look on the issue of convergence within the population as a representation problem or one of overfitting. By adopting a representation with a many-to-one genotype-to-phenotype mapping, the search space is separated from the solution space, allowing a much greater level of freedom in the genotype space.

By partitioning their populations, researchers use multiple populations to probe separate areas of the solution space simultaneously. In doing so, a balance between exploitation and exploration is sought, though limitations are encountered in large search spaces where a population can only be divided up a certain number of times. Added to this, a loss of diversity is observed without maintaining migration between the populations.

Problem decomposition and the building block-hypothesis were examined next. To a large extent this theory addresses the good intentions that inspired the incorporation of memory into GAs for dynamic environments. By seeking to identify the structural or theoretical nature of the problem, an algorithm is able to establish an understanding or identify a structure that can aid it in tracking a moving optimum, or indeed in quickly generating solutions when the optimum returns to previously examined neighbourhoods. In effect, a level of competency is developed within the population.

Finally, improving the evolvability of individuals was examined. Two approaches were analysed. The first focused on the genetic operators by explicitly identifying constructionally fit blocks of code and biasing reproduction to favour blocks that demonstrated a good statistical fitness in this area. This approach, however, was rooted in static environments and sought to encourage a quick convergence. Other studies focused on the issue of evolvability from a representational problem where the genotype and phenotype spaces are separated allowing many-to-one mappings and the evolution of neutral networks. Neutral networks were seen to improve the accessability of phenotypes where small changes in the genome could enable the discovery of newly fit phenotypes.

Moving on from approaches to aid algorithms, two related sections on performance measurement and benchmark problems were covered. These sections are related because the same criticism can be applied to both. Namely, if the goal of research into EC in dynamic environments is to bring the art closer to applicability to real-world problems, then the pursuit of good results for benchmarks and general performance measures is potentially a distraction. A danger exists that the race to deliver improvements to algorithms is being driven by one-up-manship on benchmarks and metrics, at the expense of understanding the nuances of real-world problems. A correlation between improvement on a benchmark problem and improvement on the real-world problem must be established if results on a benchmark are to carry any weight. This still considered, benchmark problems by their nature do not capture all the dynamics of complex real-world problems and should be considered more as unit tests or controls. In reviewing the categorisations in Table 3.2 the benchmark problems developed generally attempt to emulate two types of change as seen in Table 3.9. By extension, the metrics developed to evaluate the algorithms also focus on these two types. In attempting to follow through on the goals of research for EC in dynamic environments, it may instead prove more useful to research and code to the problems rather than benchmarks.

Table 3.9 The types of change faced in benchmarks, real-world problems, and nature

Type	Benchmarks	Real-World	Nature
Type		**Real-World Problems**	
Markov	X	X	X
Complex System	-	X	X
Deterministic	X	X	X

This then leads to a final issue: of all the literature covered in this chapter, not one study was conducted on the application of GP to dynamic environments. This is potentially due to a trend of working with benchmark problems for dynamic environments that focus on fixed-length individuals in bounded search spaces. The purpose of these benchmarks is to allow the results of different algorithms to be broadly comparable. However, the majority of the benchmarks are relatively simple and may potentially be holding back progression into the richer representation of GP.

3.7 Conclusion

This chapter has outlined the current state of the art in the area of EC applied to dynamic environments. Emanating from this process, first of all, is a clear categorisation of dynamic problems that exist, along with the types of changes that occur. This is achieved through the unification and development of different authors' works, which aided in the literature survey by presenting a clear perspective on what issues the various authors were seeking to address with their contributions.

Of these contributions, the incorporation of explicit memory comes at an efficiency cost while at the same time only providing a better performance for one type of change. The use of multiploid implicit memory also falls under the same criticism, but newer representations with genotype-to-phenotype mappings present promise for the efficient storage of previously useful genetic material. The development of building blocks within individuals and a competent population also serve as means to retrieve previously useful information without introducing computational costs.

The maintenance of a high level of dispersion and diversity is of great importance in developing evolutionary algorithms for dynamic environments. The implementation of mechanisms to allow this runs counter to the legacy development of the field, which has matured focusing on static problems where a speedy convergence is desired. Nevertheless a number of approaches have been described here with further opportunity presented in this study through the utilisation of GE's decoupling of search and solution spaces. Such a decoupling also provides scope for exploring evolvability along with taking into account the added flexibility through the incorporation of a BNF grammar.

There has been too much focus on the development of algorithms to narrow benchmarks, which has led the EC community to focus their efforts on bounded Markovian and Deterministic-type problems.

The most striking outcome of this survey of the literature on EC in dynamic environments is the lack of research in the application of GP to dynamic problems. In this book we endeavour to close that gap. This gap is addressed in a number of the experimental chapters beginning with more simple types of change and progressing to complex types of change in Chapter 8. In the next chapter we discuss Grammatical Evolution and its potential for application in these non-stationary domains.