Viability evolutionary algorithms and applications to neuroscience and biology

THÈSE N^{O} 6360 (2015)

PRÉSENTÉE LE 23 JANVIER 2015

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

LABORATOIRE DE SYSTÈMES INTELLIGENTS

PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Andrea MAESANI

acceptée sur proposition du jury:

Prof. E. Telatar, président du jury
Prof. D. Floreano, directeur de thèse
Prof. N. Bredeche, rapporteur
Prof. M. Dal Peraro, rapporteur
Prof. J. Timmis, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2015

“ *What I cannot create, I do not understand.*
—Richard Feynman, *1988* ”

Acknowledgements

This thesis would not be possible without the support and help that I received from many people. I am truly grateful to my advisor, Prof. Dario Floreano, for having allowed me complete freedom in choosing my own research directions and for his trust and guidance throughout this thesis. I would also like to thank the members of my jury Prof. Emre Telatar, Prof. Nicolas Bredeche, Prof. Jon Timmis, and Prof. Matteo Dal Peraro for the time dedicated reading this thesis and for their valuable comments on the manuscript.

A special thank you goes to Dr. Pavan Ramdya for the incredibly exciting and productive times of our research collaboration. I am deeply indebted to Pavan for the many things that I have come to learn thanks to his huge knowledge of (neuro)science, and for having kept me constantly motivated with his passion for science. A big thanks goes also to Prof. Roger D. Hersch for having invited me for a summer research project at EPFL, back in 2009. Probably, I would not have started my PhD studies without his original invitation. I admire the rigour and patience with which Roger tackles research in his lab, and I enjoyed a lot my time there building setups together.

Other wonderful people contributed to the realization of this thesis. Dr. Jürg Germann is one of them. I've shared the office for more than three years with this incredible Sardinian-Swiss-german man, part-time rock-star/wind-surfer/ski-teacher, and full-time PhD student (or viceversa at the beginning :-P). We supported each other through all the good and bad moments that a doctorate involves. Thanks a lot for teaching me about materials and soft polymers, I had a lot of fun learning it. Thanks also for the daily dose of good mood that you brought into the office. Dr. Pradeep Fernando and Dr. Steffen Wischmann are two others that I would like to thank for the good advices and guidance that I received from them during the first part of my doctoral research. I enjoyed spending my time discussing with you about the most disparate topics. Also, a big thank goes to Dr. Giovanni Iacca, with whom I collaborated towards the end of my PhD. Thanks to him I learned a lot more about evolutionary computation, due to his vast knowledge of the field. Also thanks a lot to Dr. Matteo DeGiacomi, Giorgio Tamo and Dr. Kyle Gustafson for all the interesting things I came to know during our collaborations.

There is no PhD student without a lab. I have to thank my labmates both for providing feedback on my work and for creating such a liveable atmosphere in the lab. It has been a great

time, between lab work, cooking battles and lab retreats. I won't write all your names as you are too many[1], but I want just to mention a few of you. Géraud thanks for being such a funny guy and teaching me a lot of electronics stuff. I enjoyed spending my time with you. One day I will also beat you at tennis. Josh (let's write entirely - Dr. Joshua Evan Auerbach - otherwise I am sure he will get pissed if I don't), Krishna and Deniz helped me a lot proofreading my poor English. With Josh I had tons of interesting discussions and it is really a pity I do not have enough time in the lab to start some crazy projects on machine learning together. I would have loved that. Pawel and Ilya gave me a lot of advices during the first part of my PhD. Thanks also to Meysam, my new office mate, for being so stressed about his thesis: it decreased my stress about handing in this thesis on time (just kidding Meysam!). Moreover Meysam, hurry, because we have to work on that side project of ours at some point. Finally, thanks Michelle for all the help with administrative matters during the PhD and the laughs about all strange things happening in the lab.

It is now the time to acknowledge the amazing Collocation Zoologique (my flatmates): Goffre (Marco), Valentina, Matassi (Mattias), Roby and until recently Santillozzo (Sebastian). Thanks guys for all the laughs we had during these years. I felt part of a (big) family being with you. Living there really helped me through all the difficult and stressful moments[2]. A special thank you goes to Valentina for allowing me to mount some absurd setup at home, including the bitcoin farm and the tomato hydroponic plantation in the house.

These years would have been very sad without the amazing Italians here in Lausanne: Frenk (the first one I met in Lausanne and the man that can eat the most amount of food in a single time) and Daniela (where is Ciocca?), Pool (finally you got a car with head restraints), Spillo (you should seriously stop smoking in my ski suit) and Martina (there is something that moves!), Tungo (Madoooooo - thanks for all the interesting discussions!) and Daniela, il Maresciallo (Aho, l'hai pulita la mela?) and Silvia, Giulio (Andiamo a Gorla?) and Roby, Pana, Elena, Millans (whenever you want to learn to swim just ask) and Silvia, Ferdinando (now known as well as Perdinando, please Messi do not spoil me any TV series once you read this) and Carla, the two Stefano (Mintchev e Varricchio), Enza and Lorenzo (please stop simulating fouls at soccer), Alice and Gabriele, la Gallina, and least but not last Tortello, Grosso, Betta and Brizzi. Thanks for all the dinners, nights, football, swimming, hikes and trips that we spent together. It would have been incredibly boring without you guys. Thanks also to Roberto for all the coffees and interesting discussions about mountains and life during these years at EPFL. A number of other people made my years at EPFL amazing, but is impossible to list all of them here, I will only mention Jonas, Koko, and the lab mates of Marco & Vale which I came to know during this time.

[1] You can find them here http://lis.epfl.ch/people and in the alumni section :-)

[2] In fact, sometimes living there increased the stress. I profit of this space to NOT thank Marco for having woken me up at least 100 times at 4am, and for inviting 20 people at home consistently before I had some deadline :-)

Among the crazy italians, Andrea Biasiucci is one of my friends that deserves a full paragraph. Andrea is feeding me daily with neuroscience stories (to be honest also with stories of another kind[3]) and has an inextinguishable drive for trying to help people affected by neurological disorders. I'm happy to share with him our startup project and to learn continuously from him about neurotechnology. As I am already writing about our project, thanks also to all the people that have collaborated with us, including Géraud, Marco, Furio and Stefano.

A well deserved thanks goes also to all my lifetime friends for still being in touch after all these years. Furio, Capo, and Colu, I think I bothered you enough talking about my research. Also, special mention to all the friends that come to visit me over the years, Padu and Jessica, Mango, Antonio, Elena and Claudio (Claudio, please bring better shoes next time we go hiking). We had great times together.

Finally, I conclude with the most important ones. First, I want to thank my family for having always supported me in these years. Grazie mamma for the good food at our Sundays lunch whenever I was in Italy, thanks dad for keeping me updated with all the news regarding Lausanne, and in particular thanks to my sister for tolerating all my jokes. Special mention to my grandparents Luigi & Mariuccia for pushing me to study, without them I would not have taken this path. Last but not least, Roberta, for sure my most important discovery. Thanks Roby for supporting and encouraging me over all these years without ever[4] complaining for the many weekends spent at work and for always accompanying me, no matter what foolish idea I'm pursuing.

Lausanne, 9 September 2014 Andrea Maesani

[3] They can be found on repubblica.it/biasiucci
[4] Editor's note: almost never

Abstract

Evolutionary algorithms are heuristic methods widely used to solve optimization problems. Generally, they operate on a population of individuals, representing candidate solutions to these problems, that improve gradually over several generations. Evolutionary algorithms select individuals with better quality (*fitness*), and reproduce them introducing random mutations. Better individuals are reproduced at a higher rate than those with worse fitness. This process of selection and reproduction, based on the ordering given by the fitness, models competition between individuals and leads to survival-of-the-fittest. Evolutionary algorithms repeatedly select and reproduce individuals until a satisfactory solution to the problem is found.

Although widely adopted, this traditional paradigm for evolutionary algorithms suffers from few simplistic assumptions made by the pioneers of the field. First, describing the quality of individuals using a single fitness value can be very challenging when several objectives and constraints have to be aggregated in the same fitness function. Second, as noted by several evolutionary biologists, in nature survival-of-the-fittest does not derive uniquely from competition, but it is better seen as the outcome of a process of competition going on in parallel to a process of elimination of non-fit individuals. Furthermore, in evolutionary algorithms, fitness is assigned prior to the reproduction of individuals. Conversely, in nature fitness is generally measured *a-posteriori*, after an individual has reproduced, and is the result of the aforementioned process of elimination and competition, not the cause of competition.

To address the mismatch between our understanding of natural evolution and the current assumptions made in evolutionary algorithms, Mattiussi and Floreano proposed a novel paradigm called *Viability Evolution* that does not require to aggregate in the same fitness function objectives and constraints. This paradigm puts emphasis on the elimination of (non-viable) individuals that do not meet a set of changing viability criteria, defined by the problem objectives and constraints. Adapting the viability criteria during evolution leads to a modification of the traits of the individuals in the population, and gradually drives the population towards desired solutions. Despite having been proposed on a conceptual level, the idea of Viability Evolution lacked experimental validation. This thesis complements the original idea of Viability Evolution proposing and testing *viability evolutionary algorithms*. Here, we investigated several viability-based algorithms with the aim of showing potential advantages that this new paradigm could offer to the field of evolutionary computation.

Abstract

We started our investigation by testing whether or not an algorithm based on Viability Evolution could produce higher diversity levels in the evolving population compared to traditional competition-based evolutionary methods, which quickly lose diversity. Experimental results showed that our viability-based method maintains higher diversity in the evolving population and generates more unique solutions on the tested set of problems. Then, we tested if the additional information available to a viability-based algorithm, such as the number of viable or non-viable individuals for each viability criteria, may be exploited during the search for achieving better performance in optimization problems. To this purpose, we combined viability principles into a state-of-the-art evolutionary method for unimodal constrained optimization. The additional information available during the evolutionary process allows to self-adapt the algorithm's parameters and achieve increased performance. Moreover, we extended this latter method deriving a viability-based algorithm to solve multimodal constrained optimization problems. Experimental results showed that this method, called mViE, outperforms current state-of-the-art algorithms on a well-known set of constrained optimization benchmarks. Successively, we embedded mViE into a novel framework for predicting structures of protein assemblies at atomic resolution. We showed how this framework performs equally or better than a previous version of the method, with the advantage that the constraints present in this problem do not have to be combined into the fitness function, as finding the weights for this combination is in itself an optimization problem.

Finally, we used evolutionary algorithms to optimize neural networks in the context of a neuroscience investigation. The optimization of neural networks is a very arduous problem with sparse and hard-to-find satisfactory solutions, spread over a vast design space. On this difficult problem, we compared our mViE method with a standard evolutionary computation method. After collecting large-scale, high-resolution measurements of *Drosophila* walking of several isogenic lines of flies, we searched for neural circuit models that could match observed patterns of spontaneous and odor-induced walking in *Drosophila*. By combining dynamical systems analysis and simulations, we showed how a noise-mediated memory of odor exposure accounts for the observed responses. Our results suggested how neural systems may use neural noise to shape behaviors, producing robust group behavioral responses and single-individual unpredictability at the same time.

In conclusion, this thesis contributes to evolutionary computation by investigating the Viability Evolution paradigm and proposing efficient viability-based evolutionary algorithms for constrained optimization. Furthermore, this thesis also contributes to biology, by providing a new viability-based framework for predicting protein assemblies at atomistic resolution and by elucidating a mechanism through which neural noise may shape animal behaviors.

Keywords: artificial evolution, evolutionary computation, viability evolution, diversity, constrained optimization, artificial neural networks, neural noise, animal behavior, spontaneous behavior, macromolecular assembly.

Sommario

Gli algoritmi evolutivi sono largamente impiegati per risolvere problemi di ottimizzazione per i quali altri approcci non sono applicabili, in cui bisogna trovare soluzioni che massimizzino o minimizzino uno o più obbiettivi e/o soddisfino dei vincoli. Questi algoritmi operano su una popolazione di individui (o soluzioni a un dato problema), migliorandone gradualmente la qualità durante il processo evolutivo simulato. Gli algoritmi evolutivi selezionano, infatti, gli individui con migliore qualità (o *fitness*), definita attraverso la cosiddetta funzione di fitness, e li riproducono introducendo mutazioni; gli individui migliori sono riprodotti maggiormente rispetto agli individui con peggiore fitness. Questo processo di selezione e riproduzione basato sull'ordinamento introdotto dalla funzione di fitness modella la competizione tra individui e cerca di simulare la selezione naturale (*survival-of-the-fittest*). Diversi biologi evolutivi hanno tuttavia osservato come in natura la selezione non derivi unicamente dalla competizione tra gli individui, bensì da un processo di competizione operante in parallelo ad uno di eliminazione di individui non abbastanza "fit". Inoltre, negli algoritmi evolutivi, il valore di fitness di ogni individuo è assegnato prima della riproduzione. Invece, in natura, il valore di fitness è normalmente misurato a posteriori, dopo che un individuo si è riprodotto, ed è il risultato dei già citati processi di eliminazione e competizione, non la causa della competizione tra di essi.

Per risolvere questa mancata corrispondenza tra il paradigma attualmente utilizzato in evoluzione artificiale e la nostra conoscenza dell'evoluzione naturale, Mattiussi e Floreano hanno proposto un nuovo paradigma chiamato Viability Evolution (*viability* è tradotto in seguito come fattibilità). Questo nuovo paradigma enfatizza l'eliminazione di individui che non soddisfano criteri di fattibilità definiti sugli obbiettivi o i vincoli di un problema. L'adattamento di questi criteri durante il processo evolutivo si riflette in una modifica dei tratti degli individui nella popolazione, e permette ad esempio di condurre gradualmente la popolazione verso le soluzioni al problema considerato. Nonostante il paradigma di Viability Evolution sia stato proposto a un livello concettuale, l'idea non è mai stata sottoposta a prova sperimentale. Lo scopo di questa tesi è verificare se una diversa modellazione degli algoritmi evolutivi attraverso questo nuovo paradigma possa apportare o meno dei vantaggi, e se sì quali. Inizialmente, abbiamo verificato se un algoritmo basato su Viability Evolution possa produrre livelli accresciuti di diversità tra gli individui di una popolazione sotto evoluzione, a differenza di algoritmi tradizionali basati su competizione che perdono diversità velocemente. I risultati sperimentali hanno mostrato che il metodo proposto mantiene una diversità maggiore nella popolazione e genera più soluzioni uniche rispetto a un metodo basato su competizione in una serie di pro-

blemi considerati. In secondo luogo, abbiamo ipotizzato che un algoritmo basato su principi di Viability Evolution possa avere accesso a più informazione durante il processo evolutivo, ad esempio il rapporto tra gli individui che soddisfano o no ciascun criterio di fattibilità, e può quindi sfruttare queste informazioni aggiuntive per ottenere migliori prestazioni nella risoluzione di un problema di ottimizzazione. Abbiamo quindi applicato principi di Viability Evolution su un metodo stato dell'arte per ottimizzazione unimodale con vincoli. Le informazioni aggiuntive disponibili permettono di auto-adattare i parametri del metodo durante l'evoluzione e accrescere le sue prestazioni. Inoltre, abbiamo proposto una versione di un algoritmo basato su principi di Viability Evolution per risolvere problemi di ottimizzazione multimodali con vincoli. I risultati sperimentali hanno dimostrato che quest'ultimo metodo, chiamato mViE, è competitivo con lo stato dell'arte e supera molti metodi sui problemi di test considerati in questa tesi. Successivamente, abbiamo applicato mViE all'interno di un nuovo metodo per la predizione di strutture atomiche di assemblaggi di proteine. Abbiamo mostrato come questo metodo offra prestazioni uguali o migliori a sistemi analoghi, con il vantaggio che, grazie al paradigma di Viability Evolution, i vincoli di questo problema non debbono più essere pesati nella stessa funzione di fitness, trovare i quali è in se un problema di ottimizzazione.

Infine, abbiamo utilizzato mViE per ottimizzare reti neurali nel contesto di una ricerca in neuroscienza sul ruolo del rumore neurale nei comportamenti animali. Ottimizzare reti neurali è in generale molto difficile, dato che le buone soluzioni sono sparse nello spazio di ricerca e difficili da trovare. Su questo difficile problema, abbiamo dimostrato che il nostro metodo è in grado di ottenere prestazioni pressochè equivalenti rispetto a metodi evolutivi tradizionali. Le reti neurali ottenute grazie ai metodi evolutivi impiegati ci hanno permesso di studiare il ruolo del rumore neurale sui comportamenti animali. Il rumore è una caratteristica ubiqua dei sistemi neurali, ma se e come influenzi il comportamento è ancora largamente sconosciuto. Dopo aver raccolto misure su larga scala ad alta risoluzione temporale sulla camminata di diverse linee isogeniche di mosche *Drosophila* (98 linee e più di 20.000 mosche analizzate), abbiamo cercato reti neurali che riproducessero i motivi osservati nella camminata spontanea o indotta da stimolazione olfattiva. Combinando l'analisi di sistemi dinamici e simulazioni, abbiamo dimostrato come una memoria dell'esposizione all'odore mediata dal rumore possa spiegare i motivi osservati nelle *Drosophila*. Abbiamo mostrato come i sistemi neurali possano utilizzare rumore per regolare i comportamenti e garantire allo tempo stesso sia riproducibilità nelle risposte di gruppo che non predicibilità a livello del singolo individuo.

In conclusione, questa tesi contribuisce sia al campo della computazione evolutiva, con un investigazione del paradigma di Viability Evolution, sia alla biologia, mostrando un nuovo meccanismo con cui il rumore neurale può regolare i comportamenti animali e fornendo un nuovo metodo per predire la struttura di assemblaggi proteici.

Parole chiave: evoluzione artificiale, computazione evolutiva, viability evolution, diversità, ottimizzazione vincolata, reti neurali artificiali, rumore neurale, comportamento animale, comportamento spontaneo, assemblaggio macromolecolare.

Contents

List of Figures

List of Tables

1 Introduction

Our current understanding of natural evolution has provided remarkable insights to computer scientists for designing powerful biologically-inspired methods. It is now widely accepted that a population will undergo evolutionary change when: i) varied phenotypes are present in the population, ii) different phenotypes have different rates of survival and reproductive success, and iii) phenotype traits are heritable (Lewontin, 1970). Based on these fundamental principles, the field of Evolutionary Computation (EC) is devoted to the study of algorithms inspired by evolutionary processes. Thousands of practitioners apply EC methods to tackle problems ranging from classical engineering optimization, like the design of power plants (Koch et al., 2007) or buildings (Kämpf and Robinson, 2010), to the most complex applications in pharmacology (Neri et al., 2007), disease diagnostics (Smith et al., 2007; Smith and Timmis, 2008) and prediction of molecular structures (Degiacomi and Dal Peraro, 2013). Moreover, EC simulations are used to test scientific questions in evolutionary biology, for which it is not possible to perform real experiments due to the large time-scales of natural evolution. For example, EC allowed to observe the emergence of strategies for communication between robots (Mitri et al., 2009; Floreano and Keller, 2010; Wischmann et al., 2012), the evolution of altruism (Clune et al., 2011; Montanier and Bredeche, 2013) and the emergence of complexity during an evolutionary process (Lenski et al., 2003). EC methods have also become extraordinary tools for the investigation of neural systems (Ruppin, 2002) and the automatic discovery of analytical relations emerging from experimental data (Schmidt and Lipson, 2009).

In this introductory chapter, a short history of the field is provided along with a description of the main paradigm on which modern Evolutionary Computation is based. Then, a few limitations of current evolutionary methods are highlighted and an alternative abstraction for EC, called Viability Evolution, is introduced. The validation of Viability Evolution represents the main subject of this thesis. In particular, this thesis is concerned with showing some possible advantages that adopting the Viability Evolution paradigm can offer to the field of Evolutionary Computation. Finally, a fascinating question from neuroscience is presented and an attempt is made to tackle it using EC. The chapter ends with an outline of the organization of this thesis.

1.1 A brief historical view of evolutionary computation

The roots of Evolutionary Computation can be traced back to the mid 20th century. At that time, two of the founders of modern computer science, John von Neumann and Alan Turing, separately developed the idea of implementing principles inspired by evolution into computers. The theoretical framework for self-replicating machines that could mutate and transfer mutations to the offspring was first introduced in a series of lectures by von Neumann, published posthumously in (Von Neumann, 1966). In parallel, Turing speculated on a machine learning method that could exploit random mutations and its analogy to mechanisms behind natural evolution (Turing, 1950). Nevertheless, the first real evolutionary experiment performed on a computer is due to an Italian-Norwegian mathematician, Niels Barricelli, working in the group of Von Neumann. Barricelli experimented with simulated organisms (Barricelli, 1954) defined by discrete sets of rules[1], and observed for the first time the reproduction of what he called "symbiorganisms", and the emergence of cooperation and parasitism (Barricelli, 1962; Barricelli, 1963). Several other researchers laid the foundations of the field experimenting with disparate methods for simulating evolution. For example, Fraser and Bremermann performed computational experiments for studying biological evolution, Box developed a method to optimize an industrial process, named "evolutionary operation", and Friedman developed the first example of a robot that could automatically design and build electric circuits to control its own behavior. These and other seminal research articles are collected in (Fogel, 1998a; Fogel, 1998b).

After the work of these pioneers of EC, the field differentiated among three main avenues of research. Evolutionary Programming was introduced by Fogel (Fogel, 1962) in an attempt to evolve automata capable of predicting future events using knowledge of observed events. From the original ideas of Rechenberg and Schwefel (Schwefel, 1965; Rechenberg, 1965) for solving engineering optimization problems with real-valued parameters, the paradigm of Evolution Strategies was born. Finally, Holland's interest in systems that could adapt to uncertain and changing environments lead to Genetic Algorithms (Holland, 1975; De Jong, 1975). The following years were characterized by investigations of the performance of these algorithms on several abstract and real-world problems and by the cross-breeding of ideas among these methods. In recent times, several flavors of powerful evolutionary algorithms for stochastic optimization have been introduced. Among these, Artificial Immune Systems (Farmer et al., 1986), Ant Colony Optimization (Dorigo et al., 1991), Particle Swarm Optimization (Eberhart and Kennedy, 1995), Differential Evolution (Storn and Price, 1997), and more recently Covariance Matrix Adaptation Evolution Strategy (Hansen et al., 2003) are some of the most used by practitioners worldwide.

[1] Barricelli's simulated organisms probably represent the precursors of what are now known as 'cellular automata'.

1.2 The canonical paradigm of evolutionary computation

Although important differences exists among the different classes of EC methods, an Evolutionary Algorithm (EA) is generally characterized by the fact that it operates on a population of individuals. These individuals can represent, for example, the candidate solutions to a specific optimization problem. The genotype of an individual is encoded into a properly designed data structure that can be randomly mutated or recombined thanks to *ad-hoc* operators. To generate offspring, individuals are reproduced proportionally to a measure of their performance generally referred to as a *fitness function*, or a *cost function* when used in an optimization context. In artificial evolution, as only a limited number of individuals is maintained in a fixed-size population, competition between individuals arises and leads to *survival-of-the-fittest*.

Since its inception, EAs have been developed around this paradigm, illustrated in Figure 1.1. The principle of survival-of-the-fittest has been implemented by sorting individuals using the values provided by a fitness function and by selecting the most "fit" of them. For about three decades, evolutionary methods flourished using this simple abstraction. Multi-objective problems and constrained optimization problems were tackled by mixing the different objectives in the same fitness function and by penalizing the fitness by the constraint violations. A shift in the paradigm used for dealing with multi-objective problems happened around the beginning of the '90s. Many researchers started to propose algorithms that independently modelled each objective and that did not require the aggregation of objectives into a unique fitness function. The turning point occurred with Goldberg's suggestion to directly incorporate the concept of Pareto Optimality[2] and domination of solutions[3] in evolutionary methods (Goldberg, 1989). Following his idea, a new class of Multi-Objective Evolutionary Algorithms (MOEA), using special sorting routines that explicitly adopted Pareto concepts[4], have been developed for multi-objective optimization (see for example (Coello Coello, 2006) for a comprehensive review).

This fundamental change in the way of tackling multi-objective problems has allowed to overcome inherent problems of mixing conflicting objectives of different scales and units in the same fitness function. However, the same problem persists in constrained optimization, where the solutions to an optimization problem must also satisfy a number of requirements defined through constraint functions. Although a myriad of techniques have been proposed to handle constraints[5], evolutionary constrained optimization seems lacking a similar unifying paradigm, and the literature still abounds with methods mixing constraints in the same fitness function.

[2] Introduced by the economist Vilfred Pareto teaching at the University of Lausanne around 1896.

[3] A solution of a problem with multiple objectives Pareto-dominates another solution if the first is not inferior to the second in all objectives and there is at least one objective where it is better. In this context, the aim of an evolutionary algorithm is to discover the optimal non-dominated set of solutions, or optimal Pareto Front.

[4] We refer to non-dominated sorting algorithms that rank solution in non-dominated Pareto fronts.

[5] Carlos A. Coello Coello has collected over the years more than 1200 references of constraint-handling techniques used in evolutionary meta-heuristics, available at http://www.cs.cinvestav.mx/~constraint/

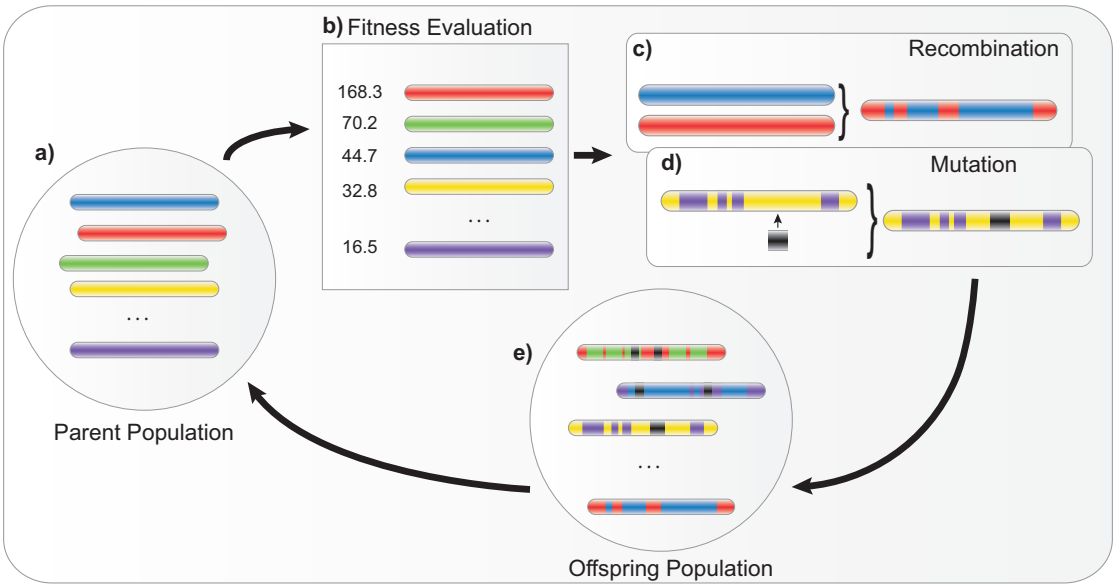


Figure 1.1: Artificial Evolution schematic. a) The different individuals in a population (colored rectangles) store information about the solution of a problem or encode a description of a virtual organism. b) The individuals are evaluated according to the fitness function and ranked according to this value. The fitness value is directly used to select individuals for reproduction. During reproduction two mechanisms can introduce variation in the offspring individuals; c) recombination mixes the information contained in the genotypes of two individuals and d) random mutations can affect the genotypes of the individuals. e) After undergoing recombinations and mutations, an offspring population is generated. The repeated application of these operations leads to selective reproduction of the fittest individuals and overall to an increase in the performance, defined in terms of a fitness function, of the members of the population.

Furthermore, although remarkable examples of "creative solutions" discovered by current EAs exists, as in the famous case of an evolved radio antenna for a NASA mission that outperformed human-designed ones with surprising design choices (Hornby et al., 2011), we are still unable to simulate open-ended evolution scenarios that could result in the incredible diversity generated by natural evolution processes. As discussed in (Mattiussi and Floreano, 2003), two consequences could have derived from embracing the currently widely adopted paradigm of evolutionary computation. First, due to the use of a single fitness function, this paradigm may have introduced problems from the start in solving constrained and multi-objective problems. Second, the current paradigm may still constrain the development of novel and more powerful evolutionary methods. Thus, the question is: what alternative paradigm one may adopt to overcome the discussed shortcomings? As seen previously, changes in underlying paradigms, for example in the case of multi-objective problems, allowed considerable advances in evolutionary computation.

Evolutionary algorithms abstract the principle of survival-of-the-fittest by ranking individuals

according to the value returned by the fitness function[6]. Thus, current EAs fundamentally model only competition among individuals. On the contrary, survival-of-the-fittest in nature does not result uniquely from such competition. Mayr notes that natural evolution can be better seen in terms of a process of eliminations (Mayr, 2002, p. 130). A mere process of selection of the fittest in each generation would be highly restrained, whereas eliminations of less fit permits the survival of a large number of individuals. As remarked by (den Boer, 1999), it is in fact more appropriate to think of natural evolution as, adopting his terminology, *non-survival of the non-fit*. According to den Boer, it is the norm that individuals which are not the "fittest" survive and reproduce, and only in rare cases a total (or partial) ordering of the individuals determines their reproductive success. Only after many generations the prevailing properties of surviving individuals result in a set of properties suggesting an overall survival-of-the-fittest.

Non-survival of the non-fit allows the persistence of temporarily non-favourable genes and their recombination with other individuals. The opposite is modelled in current EAs. Individuals compete in each generation according to a pre-established fitness function, whereas eliminations are totally absent from current EC paradigms. This may have important implications on the outcome of the evolutionary process (Mayr, 2002, p. 130-131). For example, the excessive emphasis on competition in current evolutionary algorithms may easily lead to loss of diversity in the evolving population, and consequently to the convergence of the population to a local optimum (Eiben and Smith, 2003, p. 29).[7] Furthermore, viewing an artificial evolution process solely under the fitness function paradigm can constrain choices made during the development of new methods and limit the amount of available information during the search process, as we hope will become clearer in the following of this thesis.

1.3 The Viability Evolution paradigm

Mattiussi and Floreano proposed an alternative abstraction of artificial evolution named Viability Evolution (ViE) (Mattiussi and Floreano, 2003) to overcome the inherent mismatch between our understanding of natural evolution and the current paradigm of EC. In the following, we briefly describe the ViE paradigm using their original notation.

The main idea behind ViE is to separately model competition and elimination events.[8] Both these events contribute to the reproductive success of the individuals in a population and eventually determine survival-of-the-fittest. Competition events are modelled through a *competition function*[9] that defines a partial or total ordering between the individuals of a

[6] Also multi-objective evolutionary algorithms, even though they do not require aggregating objectives in the same fitness, rank solutions according to a partial ordering.

[7] In Evolutionary Computation, this phenomenon is generally called "premature convergence"".

[8] In the original report competition events are referred to as selection events. However, we deem more appropriate to call these events "competition" ones. Selection has the broad meaning of "choosing from a group", and in this light elimination as well is an act of "selection".

[9] Originally called *selection function*.

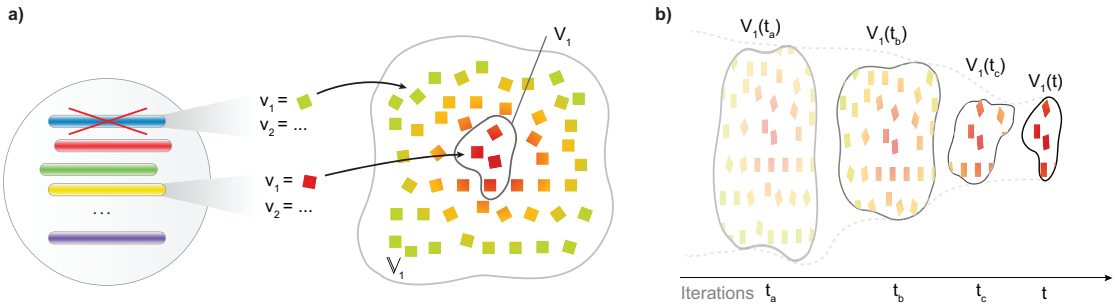


Figure 1.2: Viability Evolution schematic. a) Every individual in the population is viable if and only if all its viability criteria are satisfied, i.e. they belong to the viability set. a) Viability space $\mathbb{V}_1$ and set V_1 for the first viability criteria (thus the subscript $_1$) of the individuals of the population (colored rectangles). The current values (small squares) for this criteria that are viable are only those enclosed within the viability set V_1. The individual colored in blue is non-viable and will be eliminated from the population. b) The viability set can change during the evolutionary process. The viability viability tube represents the trajectory of the viability set in the viability time-space $\mathbb{V} \times T$. The panel depicts the viability set V_1 at different time steps of evolution $t_a \leq t_b \leq t_c \leq t$.

population. Assuming that $\mathbb{I}$ is the set of possible phenotypes, we can define the competition function $\sigma : \mathbb{I} \to \mathbb{S}$, where $\mathbb{S}$ determines a partial or total ordering of the phenotypes, for example $\mathbb{S} = \mathbb{R}$. The competition function is essentially what has been called until now a "fitness function" in evolutionary computation. Elimination events, instead, are modelled assuming that each individual has to satisfy a number of *viability criteria* to survive. The values of each of these viability criteria must be maintained within admissible ranges, specified by a *viability set*. The viability set does not necessarily have to remain constant along the evolutionary process but can be adapted by an experimenter or by an evolutionary algorithm itself, causing non-survival of non-fit individuals, or allowing favourable conditions for individuals that would not have been deemed viable before.

Viability sets can be modelled by defining a *viability function* $v : \mathbb{I} \to \mathbb{V}$ where $\mathbb{V}$ is the *viability space* of all allowed viability criteria values. According to this definition, and given a viability set $V \in \mathbb{V}$, we can say that an individual $i \in \mathbb{I}$ is viable with respect to the current viability set V if and only if it satisfies all the viability criteria, or equivalently $v(i) \in V$ (Fig. 1.2a). If the viability set changes at each simulated time step t, the set comprising all the viability sets that are valid at different time steps is referred to as the *viability tube* $\theta = \{V(t)\}_{t \in T}$, defined within the *viability space-time* $\mathbb{V} \times T$ (Fig. 1.2b). An individual surviving across multiple iterations will cover a trajectory within the viability tube and remain viable as long as its trajectory remains within the tube.

Viability criteria can be, for example, defined on objectives or constraints of an optimization problem. To provide a concrete example to the reader, let's consider the case of a real-valued constrained optimization problem where each solution $i \in \mathbb{I}$ has to satisfy m constraints of the type $g_j(i) \leq 0, j \in 1, \ldots, m$ to be considered feasible. These inequalities can be easily

described under the ViE formalism by defining $\mathbb{V} = \mathbb{V}_1 \times \ldots \times \mathbb{V}_m, \mathbb{V}_j \in \mathbb{R}$. The viability set $V = V_1 \times \ldots \times V_m$ can be relaxed at the beginning of the evolutionary process to equal the whole viability space $V = \mathbb{V}$ and can be subsequently made more stringent to reach the target viability set $V_T = V_{T1} \times \ldots \times V_{Tm}, V_{Tj} \in (-\infty, 0]$, $j = 1, \ldots, m$, that corresponds to the g_j inequalities specified above. In the particular case where a component V_j of the viability set can be expressed in terms of a lower and upper bound, we refer to the bounds of this interval as *viability boundaries*.

Now that it is clear how competition and elimination events can be modelled, we can introduce the main structure of an algorithm based on the Viability Evolution paradigm, shown in Algorithm 1. In its simpler form, a Viability Evolution algorithm reproduces surviving individuals using a selection and reproduction operator r which uses information provided by the competition function σ. The function σ can be used to determine the level of competition among individuals and in the simplest case when one wants to implement a scenario without competition, it can simply return the same value for every individual. After reproduction, non viable individuals are eliminated by the *viability elimination operator* e^v. As the population size is not constant, random eliminations performed by a *contingency elimination operator* e^c can reduce the number of individuals to avoid the unlimited growth of the population. Finally, the viability set is updated towards a target viability set by the *viability set update operator*.

Algorithm 1 Pseudo-code of generic structure of an algorithm based on the Viability Evolution paradigm. Competition and eliminations are modelled by providing appropriate viability function v, competition function σ, and the r, e^v and e^c operators for modifying the population. The operator u is used to modify the viability sets.

1: $t \leftarrow 0$
2: $P(0) \leftarrow$ SAMPLEINITIALPOPULATION()
3: $V(0) \leftarrow$ ASSIGNINITIALVIABILITYSET()
4: $P(0) \leftarrow e^v\left(P'(0), v(P'(0)), V(0)\right)$ ▷ Eliminate for viability
5: **while** ¬ TERMINATIONCONDITION() **do**
6: $\quad P'(t) \leftarrow r\left(P(t), \sigma(P(t))\right)$ ▷ Competition and reproduce
7: $\quad P''(t) \leftarrow e^v\left(P'(t) \cup P(t), v(P'(t) \cup P(t)), V(t)\right)$ ▷ Eliminate for viability
8: $\quad P(t+1) \leftarrow e^c\left(P''(t)\right)$ ▷ Eliminate for contingency
9: $\quad V(t+1) \leftarrow u\left(V(t)\right)$ ▷ Update viability set
10: $\quad t \leftarrow t+1$
11: **end while**

The details of the specific operators r, e^v, e^c and u are not specified, leaving the possibility to implement a broad variety of algorithms under the Viability Evolution paradigm by providing different implementations of these operators. For example, by disabling elimination operators one returns to the original artificial evolution abstraction. On the other hand, by disabling competition one is left with an algorithm that only employs eliminations. In this second case, modifying the viability sets is the only way to drive (indirectly) the population towards desired areas of the search space. Finally, the use of both operators allows one to model competition and eliminations in the same algorithmic framework.

1.4 Contributions of this thesis to Viability Evolution

Although Mattiussi and Floreano laid the conceptual ideas for this novel paradigm of Evolutionary Computation in their seminal report (Mattiussi and Floreano, 2003), the Viability Evolution abstraction remained untested for several years. This thesis provides a first evaluation of these ideas. Here, our main aim is to study what advantages the adoption of this paradigm can offer to the field of Evolutionary Computation. Thus, we propose several viability-based evolutionary algorithms, or simply *viability evolutionary algorithms*, that allow us to investigate different aspects of the Viability Evolution paradigm.

First, we are particularly interested in understanding the evolutionary dynamics of a process based on eliminations rather than pure competition. Capitalizing on the views of Mayr (Mayr, 2002) and den Boer (den Boer, 1999) presented previously, we hypothesise that a simulated evolutionary process based on eliminations may result in a higher level of genetic diversity in the population throughout evolution. This can potentially lead to innovative solutions discovered at the end of a simulated evolutionary process. Several mechanisms for maintaining higher diversity are available in the literature, for example fitness sharing, crowding and clearing, as reviewed in (Sareni and Krahenbuhl, 1998). Fitness sharing and clearing approaches modify the fitness of individuals depending on their proximity to other individuals. In crowding techniques the offspring replaces the most similar individuals that are worst in terms of fitness. Other approaches include multi-population EAs (Tsutsui et al., 1997; Ursem, 1999; Siarry et al., 2002; Lung, 2004) and methods that maintain explicit diversity information in each individual (Collard and Escazut, 1995; Kominami and Hamagami, 2007). However, all the approaches mentioned above still rely on the traditional paradigm of Evolutionary Computation. Novelty search explicitly promotes diversity by using a diversity measure as an objective during the search (Lehman and Stanley, 2008; Mouret, 2011). Although in this case the underlying abstraction is different, i.e. there is an explicit bias (modelled as an objective) towards increasing diversity, this seems to be very far from simulating a natural evolutionary process. In brief, the question is: can an algorithm designed according to Viability Evolution principles maintain higher diversity "for free" during the search?

Second, we already saw that an algorithm based on viability principles models distinctly the survival criteria of the individuals. In multi-objective or constrained optimization scenarios these viability criteria can be naturally defined on the objectives or constraints of the problem at hand, without any need of aggregating them in a fitness function. Although it is now natural in multi-objective optimization, after the introduction of Pareto optimality concepts, to model each objective separately, it is less intuitive for constrained optimization, where still a myriad of methods aggregate constraint violations into the same fitness function. Under the viability abstraction, a user provides a unified description of objectives and constraints as viability criteria. Therefore, algorithms modelled on the viability paradigm have access to a new source of information not available before: the number of individuals that satisfy or not (are viable or not) each viability criteria. This information, which is not available in approaches that mix constraints and objectives in the same fitness, can now be used for adapting an algorithm's

parameters or dynamically modifying the viability sets to drive the evolutionary search. For example, when exploring difficult regions of the search space a higher number of unviable individuals has to be expected and an algorithm having access to such information can adapt itself to handle the situation. To conclude, a different modelling of evolution provides access to more information during the search process, which in turn allows for designing more efficient algorithms.

In this thesis we want to highlight these aspect of the Viability Evolution paradigm by focusing on constrained optimization scenarios, in which it is still not obvious that constraints should be modelled separately. Can this additional information be used for producing more effective algorithms for constrained optimization? Both this and the diversity issue will be addressed in the first part of this thesis, leading to the development of a very effective method for constrained optimization, named *mViE*[10].

1.5 Applications to neuroscience and biology

Evolutionary Computation methods are unpaired tools for generating creative solutions of real-world problems. For this reason they have been used on multiple occasions for tackling biological questions and have provided novel insights to explain biological phenomenon, from evolutionary biology to neuroscience. In the second part of the thesis, we use evolutionary computation methods to approach a scientific question in neuroscience, regarding the role of neural noise in animal behavior.

Neural noise is a ubiquitous and poorly understood feature arising at different levels of organization of the brain (Faisal et al., 2008). Although some of its molecular basis have been understood, its role in higher behaviors and cognitive function is still unknown, with the exception of few cases[11]. Is neural noise only a disturbance present in neural systems or does it have some positive role contributing to behaviors? Neural noise may leave a direct imprint on spontaneous behaviors (Martin et al., 1999; Flavell et al., 2013), occurring in the absence of overt external stimulation, where its influence may be greater. However, does neural noise also affect sensory-driven behaviors and if yes, how?

To tackle these questions we can exploit the power of evolutionary algorithms for automatically discovering neural circuit models capable of explaining large *Drosophila melanogaster* walking behaviors datasets in the absence, i.e. spontaneous behaviors, and presence of sensory stimulation. After having obtained these models, we can dissect their dynamics to gain insights into how real circuits may achieve the observed behavioral features [12]. In our investigation, we describe these circuit models in terms of Continous-Time Recurrent Neural Networks

[10] We do not enter here in the details, the acronym stands for 'memetic Viability Evolution'.

[11] Stochastic resonance (McDonnell and Abbott, 2009) is a well known phenomenon in sensory neurobiology where noise helps the detection of weak signals in sensory systems.

[12] Of course this does not imply that real neural circuits actually work in the same way our simulated circuits work. It is however possible to gain novel insights that can drive and inform further research *in vivo*, or at least derive new hypothesis on how certain neural computations could be performed

(CTRNN) (Beer, 1995). Evolutionary Computation is a naturally apt tool for optimizing neural network parameters (Beer and Gallagher, 1992), given the complexity of these search spaces[13]. The unconstrained optimization problem considered here, i.e. fitting neural circuit models, while being essential for trying to answer the aforementioned neuroscientific questions, coincidentally represents a good real-world problem on which to test *mVie*. This is particularly interesting as it belongs to a problem class for which it was not designed[14], and allows us to compare its performance with a state-of-the-art EC algorithm.

Furthermore, we apply mViE to a second real-world application, this time in biology, related to the prediction of atomic structures of protein assemblies. The knowledge of the structure of an assembly at atomistic resolution is required for studying its function (Ward et al., 2013). A state-of-the-art framework for structural prediction of symmetric protein assemblies has been presented in (Degiacomi and Dal Peraro, 2013). This framework models the search for candidate predictions as an optimization process. The cost function is defined by a weighted combination of an energy function to be minimized and geometrical constraints on the assembly, whose violations have to be minimized as well. However, the definition of weights for the energy and constraints is in itself a time-consuming optimization problem. mViE, by modelling separately constraints as viability criteria can remove the need for combining these different constraints and energy into a single fitness function. Thus, we apply the mViE algorithm, combined with further improvements, on this prediction problem and derive a more powerful prediction framework[15].

1.6 Organization of the thesis

The chapters of this thesis are self-contained to allow an easier access to to the content more suiting the reader's own interest. All chapters are based on material submitted or published in scientific journals and in many aspects follow the same structure of a scientific article.

In the first part of the thesis, we report the main results and contribution regarding the validation of Viability Evolution principles in Evolutionary Computation. In **Chapter 2**, we model a simple Viability Evolution algorithm based on a canonical genetic algorithm. In this chapter, we are interested in particular in investigating how the diversity of evolving population may differ in the original genetic algorithm and the one based on Viability Evolution principles. We show that it is possible to drive the search towards interesting regions of the search space by adapting the viability boundaries and using eliminations only, eventually discovering more diverse solution than the ones found by a standard competition-driven algorithm.

[13] Search spaces, or "fitness landscapes" in EC jargon, for this problems can be very difficult to explore due to the high number of "bad" networks that produce trivial responses.

[14] We remember that mVie is designed for constrained optimization scenarios, whereas in this case the problem is unconstrained.

[15] We recently applied mViE to the prediction of protein assembly structures, therefore no separate chapter exist for describing our results. An appendix contains the main experimental results and description of the method used, but no detailed description is available.

In **Chapter 3**, we orient our research towards more practical aspects of Evolutionary Computation, in particular to tackle constrained optimization problems. Our aim is to show that the adoption of the Viability Evolution paradigm provides additional information during the search, which can be readily used for improving an algorithm's performance. Specifically, we enrich a state-of-the-art algorithm, Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES), with viability boundaries. We show that viability boundaries allow for collecting additional information about the structure of the search space. This information can then be used to more effectively adapt the parameters of CMA-ES. This leads to a more efficient algorithm for unimodal constrained problem solving.

This method is extended in **Chapter 4** to solve a broader class of constrained optimization problems. We show that by using a population composed of distinct search distributions that use viability boundaries we can solve also multimodal constrained problems more efficiently. We compare our final proposed method (mViE) with current state-of-the-art constrained optimization methods, showing its competitiveness. Finally, we recently tested mViE on a biological problem, the prediction of molecular assemblies. The results of this investigation are briefly presented in **Appendix A**.

In the second part of the thesis, composed of **Chapter 5**, EAs are a fundamental ingredient of an exciting multidisciplinary research project where we try to elucidate the role of noise in the neural systems of *Drosophila* flies. We combine artificial evolution with dynamical systems analysis and genome-wide association studies to investigate how neural noise may be used by neural systems to sculpt behaviors. The optimization of neural network parameters that need to be tackled here represents a good real-world scenario for testing the Viability Evolution algorithm presented in the previous chapter (mViE). Our findings show that noise can be employed to construct complex and robust behavioral responses and they also demonstrate how EAs can be used to tackle neuroscientific problems.

2 Artificial evolution by viability rather than competition

In the traditional Evolutionary Computation paradigm, selective reproduction of the fittest, inspired by competition-based selection in nature, leads to loss of diversity within the evolving population. This can cause the premature convergence of an evolutionary algorithm, hindering the discovery of many different solutions. In this chapter, we investigate a preliminary version of a Viability Evolution algorithm that puts emphasis on the elimination of individuals not meeting a set of changing viability criteria, which are defined on the problem objectives and constraints. Experimental results show that this simple version of Viability Evolution maintains higher diversity in the evolving population and generates more unique solutions when compared to classical competition-based evolutionary algorithms.

The contents of this chapter are adapted from:

Maesani A, Fernando PR, Floreano D (2014). Artificial Evolution by Viability Rather Than Competition. PLoS ONE 9(1): e86831. doi:10.1371/journal.pone.0086831

DISCLOSURE: The experiments discussed in section 2.4.2 have been performed by a co-author of this study, Dr. Pradeep Ruben Fernando. We report them here for completeness.

2.1 Introduction

Evolutionary algorithms are heuristic optimization methods inspired by natural evolution (Goldberg, 1989; Fogel, 1994; Fogel, 1995; Bäck, 1996). They operate by selecting, reproducing, and mutating the genotypes of individuals with higher performance in a population where each individual is a candidate solution to the problem. A fitness function is used to score individuals according to how well they perform on the problem objectives, and a selection operator allocates higher number of copies with random mutations to individuals with higher fitness. This process of fitness-based selection models natural competition between organisms of a population for contributing offspring to the next generation. The generational cycle of fitness assessment, selective reproduction of individuals with higher fitness, and random mutations is repeated until a satisfactory solution to the problem is found. The simplicity, effectiveness, and wide applicability of evolutionary algorithms have contributed to their adoption in a very large number of problem domains, from computer science to engineering, all the way to pharmacology (Foster, 2001; Eiben and Schoenauer, 2002). Moreover, evolutionary algorithms are widely used to investigate biological questions by conducting in-silico experiments (Lenski et al., 2003; Clune et al., 2008; Wischmann et al., 2012; Bongard, 2011).

A difficulty of evolutionary algorithms that hinders the discovery of several unique solutions stems from the gradual loss of diversity caused by the repeated application of competition-based reproduction of the fittest individuals, which can lead to premature convergence of the evolving population to a sub-optimal solution (Eiben and Smith, 2003; Mattiussi et al., 2004). Furthermore, in multi-modal problems, where there are multiple global or local optima, possibly distributed over the solution space, evolutionary algorithms tend to converge to only one set of solutions (*i.e.* one global or local optimum) as population diversity decreases. Although several techniques have been proposed to delay or reduce premature convergence, for example see (Park and Ryu, 2010; Adra and Fleming, 2011; Ginley et al., 2011) for a recent review of existing methods, loss of population diversity is intrinsic to the majority of evolutionary algorithms and is influenced by the selection method employed.

In this chapter, we test the hypothesis that Viability Evolution (Mattiussi and Floreano, 2003), by modelling separately competition and eliminations, can maintain higher levels of diversity during evolution than traditional competition-based evolutionary methods. First, as no practical algorithm has been proposed in (Mattiussi and Floreano, 2003), we have to devise a simple viability evolution algorithm, that captures the fundamental properties of the viability paradigm. To make things simple, we do not model competition and only use eliminations with changing viability boundaries to drive the evolutionary search. Furthermore, to compare this viability algorithm with traditional competition-based algorithms we directly incorporate

viability principles into a simple genetic algorithm. The resulting method is named *ViE* in the rest of this chapter. We show that ViE maintains a higher level of diversity, leading to the discovery of a larger number of alternative solutions than found by traditional competition-based evolutionary algorithms.

2.2 A simple Viability Evolution algorithm

The Viability Evolution algorithm presented in this chapter (ViE) consists of defining viability boundaries, creating an initial population, and repetitively applying reproduction, elimination, and boundary updates until the boundaries meet the desired values (Figure 2.1). Viability boundaries are expressed as inequalities on the problem objectives and define the characteristics of the desired solutions. The algorithm follows as close as possible the structure of a steady-state genetic algorithm (SSGA), as we deemed important for our investigation to base our method on a very simple genetic algorithm for facilitating the analysis of the method's performance. SSGA is easy to study due to the generation of a single individual per iteration.

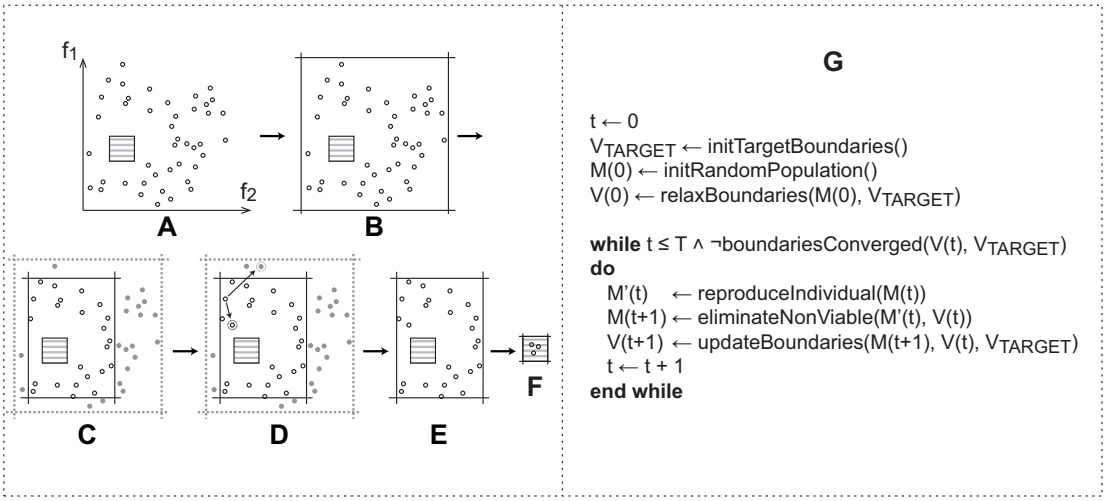


Figure 2.1: The Viability Evolution (ViE) algorithm. The population under evolution is shown in a two-dimensional objective space, defined by the f_1 and f_2 objective functions in this example. (A) Individuals of the initial population $M(0)$ (black circles) are randomly generated. The region enclosed by the target viability boundaries (gray stripes) is extremely unlikely to contain any of the randomly generated individuals in the initial population. (B) The initial viability boundaries $V(0)$ are set by the algorithm in terms of inequalities on the objectives to encompass all individuals in the initial population. (C) Viability boundaries are modified to approach the target boundaries V_{TARGET} ; as a result, a fraction of the population becomes non-viable (gray shaded circles) and is marked for elimination. The way in which the boundaries are modified depends on the specific viability boundary update procedure implemented by the user. See Figure 2.2 for further details on the update mechanism used in this chapter. (D) All viable individuals are allowed to reproduce by making one mutated copy at each iteration of the algorithm. Mutated copies that fall within the viability boundaries are allowed to stay along with the parent. Mutated copies that fall outside the viability boundaries are marked for elimination. (E) Non viable individuals are eliminated from the population. (F) The process described in (C-E) is repeated for many iterations until the viability boundaries reach the target values or the maximum number of evaluations t is exhausted. (G) The algorithmic description of Viability Evolution.

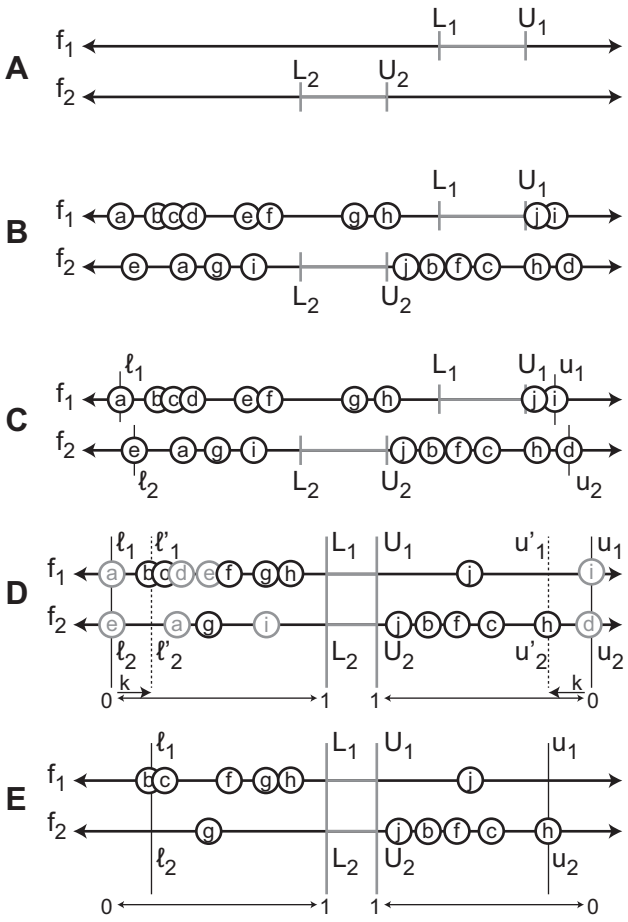


Figure 2.2: Boundary update mechanism in the Viability Evolution (ViE) algorithm. (A) Let us assume, without loss of generality, that the problem to be solved is defined by two objectives, f_1 and f_2. The target regions of the given problem are defined by the *target* viability boundaries - $[L_i, U_i]$ for each objective function f_i respectively. Thus, the goal is to find solutions which have values between L_i and U_i for each objective function f_i respectively. (B) Individuals of the initial population are randomly generated. Each individual is represented using a circle on the axis of each objective function. The position of a circle on the axis of an objective function f_i indicates the value of the corresponding individual for that particular objective. In this example, each individual is represented using 2 circles - one on each axis of the two objective functions. (C) The *initial* viability boundaries are set for each objective f_i by identifying the extreme values $[\ell_i, u_i]$ on either side of the corresponding target viability boundaries $[L_i, U_i]$. The initial viability boundaries thus encompass all individuals in the initial population. (D) The viability boundaries are then tightened such that at least a minimum fraction k of individuals become non-viable. To illustrate this clearly, the intervals - $[\ell_i, L_i]$, and $[u_i, U_i]$ are both rescaled to [0, 1] here. The new values for the viability boundaries $[\ell'_i, u'_i]$ (shown as dotted lines) for each objective function f_i are computed such that at least a minimum fraction k of individuals in the population become non-viable (shown as light gray circles). (E) Non viable individuals are eliminated from the population. The population continues to evolve with the new viability boundaries until the next boundary update.

To explain the workings of the Viability Evolution algorithm, let us consider the example of finding the electronic design of a low-pass filter that meets desired values for the gain-bandwidth product (GBW), the pass band flatness (PBF) and the stop band attenuation (SBA). These three parameters represent the viability conditions for the survival of the circuits. For each viability condition c a lower bound L_c and an upper bound U_c are defined (Fig 2.3). A circuit x is deemed viable only if all its viability boundaries are satisfied, that is $L_{GBW} \leq GBW(x) \leq U_{GBW}$, $L_{PBF} \leq PBF(x) \leq U_{PBF}$, $L_{SBA} \leq SBA(x) \leq U_{SBA}$. Because the initial population is randomly generated, it is extremely unlikely that any individual can satisfy all the viability conditions. Therefore, the lower and upper bounds of the viability conditions are initially set to encompass all individuals, and are gradually modified during evolution to approximate the desired values.

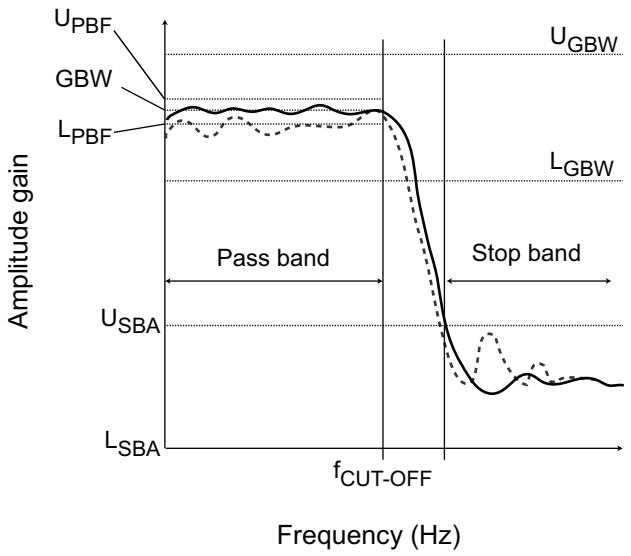


Figure 2.3: Example of viability boundaries definition for a filter design problem. A candidate filter design being optimized with Viability Evolution must satisfy certain requirements defined by the user as viability boundaries. Here, the filter gain-bandwidth product (GBW, computed at the cutoff frequency $f_{CUT-OFF}$) must satisfy the viability boundary $L_{GBW} \leq GBW(x) < U_{GBW}$. The stop-band attenuation (SBA) of the filter is also constrained by the viability boundary $L_{SBA} \leq SBA(x) < U_{SBA}$. Finally, a filter must also satisfy a requirement on the pass-band flatness (PBF), i.e. the deviation of amplitude gain from the gain at cut-off frequency, such that $L_{PBF} \leq PBF(x) < U_{PBF}$. The response for two different filters is depicted in figure. The first filter (solid line) is viable as it satisfies all viability boundaries, while the second filter (dashed line response) is non-viable, as it violates the viability boundaries expressed on pass-band flatness.

At each subsequent iteration of the algorithm, each individual can reproduce by adding a mutated copy to the population (the parent remains in the population). In order to give each unique individual in the population equal chance of being reproduced, we have to account for the possibility of clones, resulting for example from individuals that remain viable for a long time and produce lots of copies. To achieve this, the algorithm keeps track of the descendants of the initial population by assigning a different family identifier to every individual in the initial population (note that only mutations are used in reproduction). The reproduction probability of each individual takes into account the size of its family. This is done by first selecting a family of individuals inversely proportionally to its size from the current population and then randomly selecting an individual within that family. Once an individual has generated an offspring, its family identifier is assigned to the offspring and the family size is increased by one unit.

After the reproduction phase, individuals that fall outside the current viability boundaries are eliminated and the size of the families of these individuals is reduced accordingly. The two events that may lead to elimination of an individual are mutations and modifications of the viability boundaries. All the viability boundaries are modified simultaneously (Figure 2.2) so that at least a fraction of individuals (defined by the user) is discarded from the population. After each boundary update, boundaries are not modified until the population generates a number of viable individuals equal to at least the number of eliminated individuals. As soon as this condition is satisfied, the viability boundaries are updated again and this process is repeated until they reach the target values.

Once the boundaries are converged to the desired values, the algorithm returns the final population of solutions to the user. Note that all these solutions satisfy the user-defined criteria of success. Therefore, the user may choose any one of them or use additional criteria after inspection of the genotypes of the solutions.

2.3 Experimental Setup

We compared the Viability Evolution algorithm with two traditional, competition-based, evolutionary algorithms: a genetic algorithm with steady-state replacement (Whitley, 1989), which will be denoted as SSGA in the rest of the chapter, and the Elitist Non-dominated Sorting Genetic Algorithm, or NSGA-II (Deb et al., 2002a). As mentioned above, SSGA is very similar to the ViE algorithm. Both SSGA and ViE follow the same cycle of parent selection, offspring generation, and selection of individuals for the next generation. Both use the same mutation operators to produce exactly one offspring per generation or iteration. SSGA and ViE differ in the mechanisms used to select the parent individuals for reproduction and the surviving individuals for the next generation. While SSGA uses the fitness-based rank of individuals for both operations, ViE allows all viable individuals to survive and reproduce. ViE was also compared against NSGA-II in search scenarios generated from standard multi-objective benchmark functions. NSGA-II is a widely used evolutionary algorithms for multi-objective

Table 2.1: Standard benchmark functions used to generate the single-objective fitness landscapes. The a_{ij}, b_{ij} and c_i coefficients defined in the Fletcher-Powell and Langerman functions are the same used in (Eiben and Bäck, 1997). The Hump function was randomly generated using the multimodal test generator presented in (Rönkkönen et al., 2008). In the table we report the D-dimensional problem formulation (if available) or a 2-dimensional formulation. Furthermore, we denote if the functions employed are multi-modal (M) and/or separable (S), and their original reference (R).

Function	Formulation	Domain	M	S	R
Sphere	$f_1(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$	$x_i \in [-5.12, 5.12]$		✓	(Eiben and Bäck, 1997)
Double Sum	$f_2(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j\right)^2$	$x_i \in [-65.536, 65.536]$			(Eiben and Bäck, 1997)
Rastrigin	$f_3(\mathbf{x}) = 10D + \sum_{i=1}^{D} (x_i^2 - 10 \cdot cos(2\pi x_i))$	$x_i \in [-5.12, 5.12]$	✓	✓	(Eiben and Bäck, 1997)
Ackley	$f_4(\mathbf{x}) = 20 + e - 20e^{\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right)} - e^{\left(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i)\right)}$	$x_i \in [-20, 30]$	✓		(Eiben and Bäck, 1997)
Griewangk	$f_5(\mathbf{x}) = 1 + \sum_{i=1}^{D} \frac{x_i^2}{400D} - \prod_{i=1}^{D} cos\left(\frac{x_i}{\sqrt{i}}\right)$	$x_i \in [-600, 600]$	✓		(Eiben and Bäck, 1997)
Fletcher-Powell	$f_6(\mathbf{x}) = \sum_{i=1}^{D} (A_i - B_i)^2$ $A_i = \sum_{j=1}^{D} (a_{ij} \cdot sin(\alpha_j) + b_{ij} \cdot cos(\alpha_j))$ $B_i = \sum_{j=1}^{D} (a_{ij} \cdot sin(x_j) + b_{ij} \cdot cos(x_j))$	$x_i \in [-\pi, \pi]$	✓		(Eiben and Bäck, 1997)
Langerman	$f_7(\mathbf{x}) = -\sum_{i=1}^{D} c_i e^{\left(-\frac{1}{\pi}\sum_{j=1}^{D}(x_j - a_{ij})^2\right)} \cdot cos\left(\pi \sum_{j=1}^{D}(x_j - a_{ij})^2\right)$	$x_i \in [0, 10]$	✓		(Eiben and Bäck, 1997)
Shubert	$f_8(\mathbf{x}) = \sum_{i=1}^{5} i \cdot cos((i+1)x_1 + i) \cdot \sum_{i=1}^{5} i \cdot cos\left((i+1)y_2 + i\right)$	$x_i \in [-10, 10]$	✓		(Li et al., 2002)
Vincent	$f_9(\mathbf{x}) = -\frac{1}{D}\sum_{i=1}^{D} sin\left(10 \cdot log(x_i)\right)$	$x_i \in [0.25, 10]$	✓		(Shir and Bäck, 2006)
Hump	$f_{10}(\mathbf{x}) = h_k \left[1 - \left(\frac{d(\mathbf{x},k)}{r_k}\right)^{\alpha_k}\right]$ if $d(\mathbf{x},k) \leq r_k$ otherwise $f_{10}(\mathbf{x}) = 0$ $d(\mathbf{x},k)$ is the Euclidean distance to the k-th peak where h_k, α_k and r_k are height, shape and radius of k-th peak	$x_i \in [0, 1]$	✓		(Singh and Deb, 2006)

optimization and uses sophisticated techniques to rank individuals and explicitly promote the maintenance of high diversity in the evolving population. All the three algorithms do not use crossover to simplify the analysis of the results.

We first compared ViE and SSGA on ten search landscapes [1] generated using standard single-objective benchmark functions (Eiben and Bäck, 1997; Li et al., 2002; Singh and Deb, 2006; Shir and Bäck, 2006), as we wanted to start with the simplest scenario as possible with a single viability criteria. Figure 2.4 shows the search landscapes in two dimensions for these functions, whereas their mathematical formulations can be found in Table 2.1.

In these search landscapes, the goal for the compared algorithms is to find the highest number of solutions with at least a score that satisfies the user requirements. This goal is not explicitly modelled in the SSGA or ViE. SSGA proceeds by optimizing the single objective, and ViE models the search using a single viability boundary, that is tightened until it reaches the target objective score. Here, we are interested to understand if different diversity levels emerges from the different evolutionary dynamics of the two algorithms. Thus, we compared genetic diversity and number of unique solutions discovered by the two algorithms. The number of unique solutions was measured as the number of unique individuals found within target areas defined on these search landscapes.

These target areas were specified by thresholding the original objective function of each selected standard benchmark problems (Figure 2.5A). All solutions with an objective value higher

[1] As in the viability abstraction a "fitness function" is not present, we prefer to call what normally is known as a *fitness landscape* as search landscape

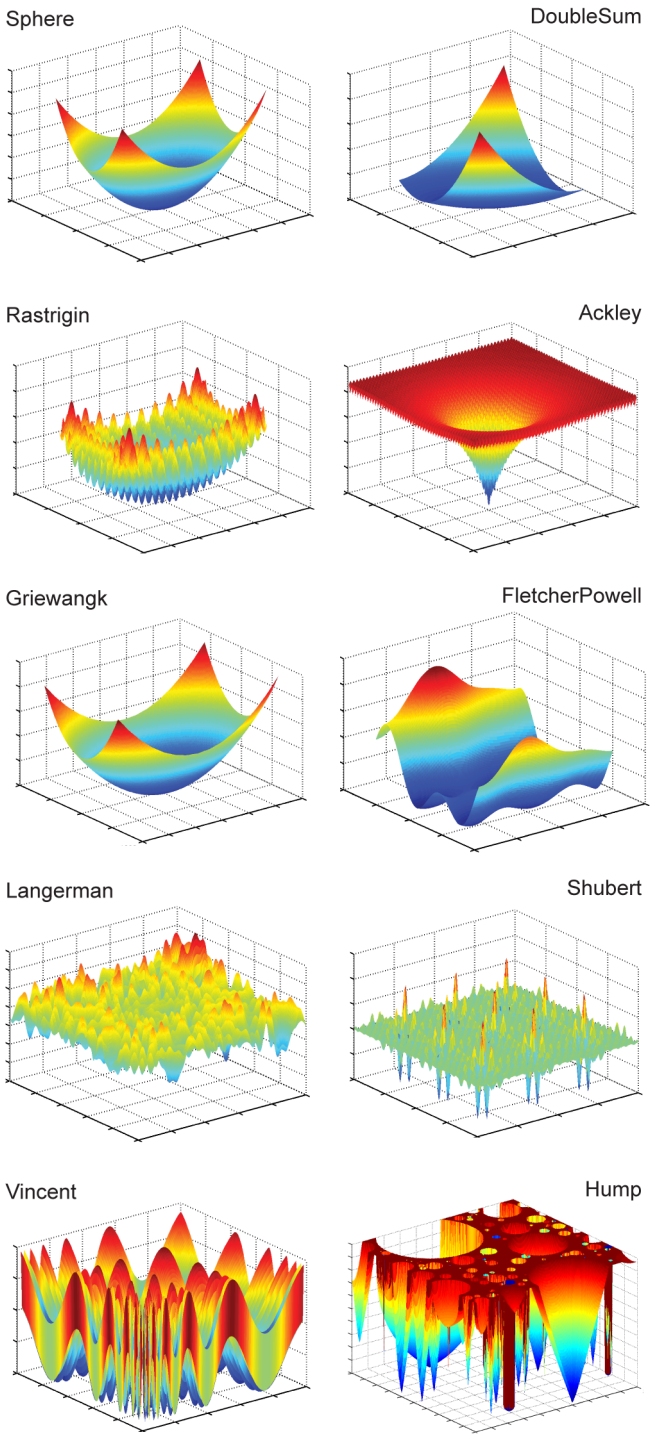


Figure 2.4: Fitness landscapes for single-objective problems. The single-objective functions include uni-modal, multi-modal and non-separable functions (Table 2.1). We defined fitness-capping thresholds on the landscapes to obtain a number of disconnected areas containing solutions at the same fitness level (Figure 2.5). The Griewangk landscape, globally similar to Sphere, contains a large number of local minima that are indistinguishable in this figure.

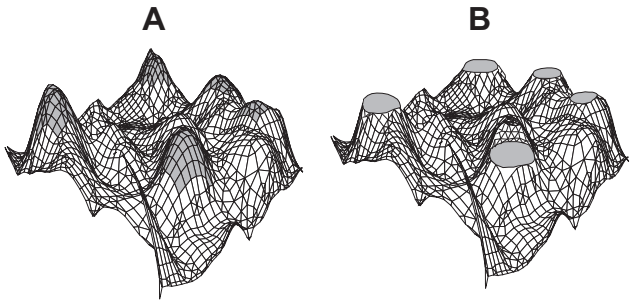


Figure 2.5: Generation of search landscapes for testing diversity and number of unique solutions discovered. (A) We used standard single-objective benchmark functions to define different search landscapes. A threshold on the objective function of a standard benchmark function identifies one or more (possibly disjoint) target areas, depicted as gray regions. (B) In order to prevent competition-based algorithms from reducing diversity after reaching the target areas, and thus allowing a fair comparison with ViE, a search landscape is reshaped such that the same objective value is assigned to any solution lying above threshold so that they all have the same probability of being selected for reproduction.

than the threshold were considered lying in a target area. The threshold value was defined so that the number of discoverable unique solutions was equal to the constant population size in SSGA in order to ensure that ViE did not take advantage of its variable population size by simply increasing the number of viable solutions. By limiting our experiments to two and three dimensional problems over finite solution spaces (bitstring encoding), we can enumerate the entire solution space and hence, precisely count the number of solutions in the target regions, which allows a precise comparison of the three evolutionary algorithms.

Furthermore, to make the comparison fair, we modified the search landscapes by setting for all solutions contained within the target areas an objective value equal to the threshold, i.e. we flattened the search landscape in target areas (Figure 2.5B). Therefore, each solution within the target areas had the same chance of being selected for reproduction, and this prevented SSGA from further reducing diversity by selective reproduction of above-threshold solutions. The resulting search landscapes are named “single-objective” in the following, as they were derived from standard single-objective benchmark functions.

In some cases, the resulting target areas were disjoint and far away from each other (see Table 2.2 for the characterization of the different single-objective functions and the threshold values used), which represented some of the most interesting problems for the comparison of the two algorithms because their search landscapes contain very different target solutions, i.e., lying in the different target areas. For such problems, we compared the ability of the algorithms to thoroughly explore the solution space and find as many disjoint target areas as possible, while also considering the number of evaluations taken by each algorithm to find the target areas. The ten, single-objective search landscapes included two that were generated from functions with no local optima and having a single target area (Sphere and DoubleSum),

Table 2.2: Characteristics of the fitness landscapes generated for the different single-objective experiments. In this table, we report the benchmark function used to generate the landscape, the number of disconnected target areas (T) and the threshold applied on the original function to discriminate the target areas (A). Additionally, we classify these problems into three main categories: uni-modal with single target areas (a), multi-modal with single (b) or multiple (c) target areas, and indicate in the table which group each problem belongs to. The sum of the number of unique solutions over all the target areas of each problem is 100, except for Ackley (97).

Benchmark	**Function**	**Group**	**T**	**A**
Sphere	f_1	a	1	0.00020322
Double Sum	f_2	a	1	0.032016
Rastrigin	f_3	b	1	0.040295
Ackley	f_4	b	1	0.29747
Langerman	f_7	b	1	−3.0622
Fletcher-Powell	f_6	c	4	0.13806
Griewangk	f_5	c	13	0.14654
Shubert	f_8	c	18	−186.637
Vincent	f_9	c	36	−0.99998
Hump	f_{10}	c	36	−0.99951

three generated from multi-modal functions and having a single target area (Rastrigin, Ackley, Langerman), and five generated from multi-modal functions and having disjoint target areas (FletcherPowell, Griewangk, Shubert, Vincent, Hump).

We also generated more complex search landscapes using standard multi-objective problems, referred in the following as "multi-objective". ViE, SSGA and NSGA-II were compared on three search landscapes, obtained using a standard problem generator called DTLZ (Deb et al., 2002b) (described in Tables 2.3 and 2.4). The multi-objective problems were composed of three objectives, that have been modelled as three separated viability criteria in ViE. Moreover, the algorithms were compared on an electronic circuit design problem (Figure 2.6). The generation of these multi-objective search landscapes followed the method for flattening the landscapes in target areas, described above in Figure 2.5.

Both SSGA and NSGA-II algorithms were terminated when the objective values of all the individuals in the population reached the best achievable objective, corresponding to the one used for generating the target areas. For ViE, this corresponds to terminating the algorithm when the viability boundaries reach the target boundary values.

Each evolutionary algorithm was assessed N times ($N = 50$) on each benchmark problem. For each repetition i of an algorithm, the random number generator used by the probabilistic functions (i.e., generation of the initial population, reproduction, and mutation) was initialized using seeds r_i, $r_i \in R = \{r_1, r_2, \ldots, r_N\}$, where R was a set of N random numbers generated by

Table 2.3: The multi-objective DTLZ problem definitions. The DTLZ problems, as originally introduced in (Deb et al., 2002b), have been specifically designed for multi-objective EA and allow to control the difficulty of converging to the Pareto-optimal front. Specifically, these three problems pose different difficulties to the optimization algorithms. The DTLZ1 test problem requires the optimizer to find solutions on linearly distributed Pareto fronts, while the DTLZ2 and DTLZ4 test problems contain solutions distributed on spherical Pareto fronts. The DTLZ4 test problem has an additional problem difficulty as each front in the solution space contains an uneven distribution of solutions. Using ViE on multi-objective problems is simple because the experimenter does not have to combine the different objectives into a single fitness function, but can directly define the target set in terms of constraints on the different objectives (see Table S4 for the definition of the target viability sets).

Problem	**Formulation**
DTLZ1	$min\ f_1(x) = \frac{1}{2}x_1x_2...x_{n-1}(1+g(x_n))$
	...
	$min\ f_{n-1}(x) = \frac{1}{2}x_1(1-x_2)(1+g(x_n))$
	$min\ f_n(x) = \frac{1}{2}(1-x_1)(1+g(x_n))$
	$g(x) = 1+(x-0.5)^2 - cos(20\pi(x-0.5))$
DTLZ2	$min\ f_1(x) = (1+g(x_n))cos(\frac{\pi}{2}x_1)...cos(\frac{\pi}{2}x_{n-1})$
	$min\ f_2(x) = (1+g(x_n))cos(\frac{\pi}{2}x_1)...sin(\frac{\pi}{2}x_{n-1})$
	...
	$min\ f_n(x) = (1+g(x_n))sin(\frac{\pi}{2}x_1)$
	$g(x) = (x-0.5)^2$
DTLZ4	$min\ f_1(x) = (1+g(x_n))cos(\frac{\pi}{2}x_1^\alpha)...cos(\frac{\pi}{2}x_{n-1}^\alpha)$
	$min\ f_2(x) = (1+g(x_n))cos(\frac{\pi}{2}x_1^\alpha)...sin(\frac{\pi}{2}x_{n-1}^\alpha)$
	...
	$min\ f_n(x) = (1+g(x_n))sin(\frac{\pi}{2}x_1^\alpha)$
	$g(x) = (x-0.5)^2$
	$\alpha = 100$

software available at http://www.random.org/integer-sets/.

The initial population size M was set to 100 for the single-objective benchmarks and to 300 for the electronic circuit design problem and the 3-objective benchmark problems, unless otherwise stated in the Experimental Results section. For each repetition, we allowed each algorithm to evaluate at most T individuals ($T = 10000$), if the termination criteria were not reached earlier.

The genotype of the individuals was a binary string encoding 2 parameters for single-objective problems and for the electronic circuit problem, and 3 parameters for the multi-objective problems. Each parameter was encoded by 12 bits for single-objective problems, 10 bits for the electronic circuit problem, and 8 bits for multi-objective problems. Mutation consisted of flipping each bit of the genotype with probability $\frac{1}{l}$ where l was the genotype length. In SSGA, selective reproduction was performed by means of tournament selection (size $k = 2$, which

Table 2.4: The target viability boundaries for the multi-objective benchmark problems. The target boundaries for the DTLZ and the filter design problems are described by constraints on the problem objectives. This table shows the target viability boundaries A and the number of target solutions M for each problem.

Benchmark	**A**	**M**
DTLZ1	$f_i(x) \in [0, 0.30937]$	296
DTLZ2	$f_i(x) \in [0, 0.58361]$	300
DTLZ4	$f_i(x) \in [0, 0.67827]$	300
Filter design	$c_1 \in [9 \cdot 10^6, 10 \cdot 10^6]$	298
	$c_2 \in [0, 0.22]$	
	$c_3 \in [0, 0.2988]$	

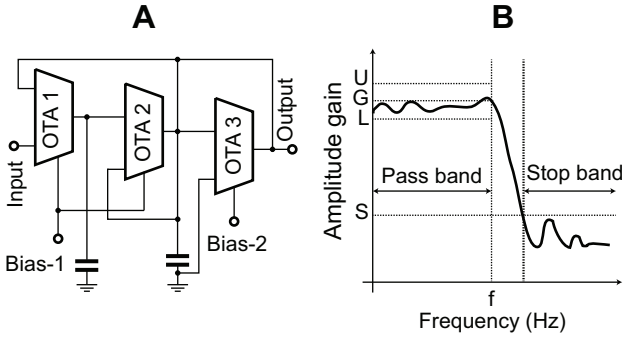


Figure 2.6: The filter design problem. (A) A low-pass filter was evolved using the circuit topology derived from (Geiger and Sanchez-Sinencio, 1985) (depicted in figure). This circuit topology allows the filter functionality to be modified using two bias current inputs (Bias-1 and Bias-2). The filter functionality is specified using constraints on three frequency response characteristics, namely gain-bandwidth product, pass band flatness and stop band attenuation. Hence, a solution to this problem is a pair of bias current values and the goal of an evolutionary algorithm is to find values for these two bias currents, assuming the fixed topology filter circuit, such that the specified low pass filter functionality is obtained. The three constraints on the frequency response characteristics of the filter are set such that there are approximately 300 (296, due to the quantization resolution introduced by the fixed bitstring encoding on possible values) bias current pair values that satisfy all three constraints. The performance of each candidate solution is obtained from simulations of the filter circuit using the SPICE circuit simulator. The SPICE models for the operational trans-conductance amplifiers (OTAs) used to build the filter circuit are available from http://www.ti.com/product/LM13700. (B) A typical frequency response of a low pass filter. The desired cutoff frequency f and output amplitude G are shown. The maximum deviation from G is defined by specifying a lower bound L and upper bound U. Finally, S represents the desired value for the maximum amplitude of any stop band ripple.

corresponds to the lowest possible selection pressure). NSGA-II also employs tournament selection (size $k = 2$) with the crowded comparison operator as proposed in (Deb et al., 2002a). Crossover was disabled in all the evolutionary algorithms. SSGA, and ViE generated 1 offspring per iteration while NSGA-II uses its default generational offspring generation and replacement policies. In the Viability Evolution algorithm, the fraction of killed individuals at every constraint update was set to 5% of the population size. The computer code, and all the software needed to reproduce the results presented in this chapter can be found at http://lis.epfl.ch/VIE.

We used the NSGA-II multi-objective optimizer with constraints for the multi-objective experiments (available at http://www.iitk.ac.in/kangal/codes.shtml). The constraints were set to the target viability boundaries values. This ensures that the NSGA-II algorithm will attempt to maintain high diversity as well to reduce constraint violations, and correctly assign maximum preference to the solutions within the target area of the search space.

2.4 Results

2.4.1 ViE maintains higher diversity in "single-objective" search landscapes

ViE is able to maintain higher genetic diversity, than SSGA on all single-objective search landscapes (Figure 2.7). Genetic diversity was computed as Hamming distance between all individuals in the population as in (Wineberg and Oppacher, 2003). Genetic diversity is significantly higher ($P < 0.001$, Wilcoxon rank sum test) over the entire evolutionary time, except for the initial iterations of the algorithms where diversity is comparable in both algorithms due to the random initialization of their populations.

Higher genetic diversity in ViE results in a significantly higher number of unique target solutions in the population at the final iteration than in SSGA in all benchmark problems except Ackley where ViE display similar performance to SSGA (Wilcoxon rank sum test, Rastrigin and Shubert: $P < 0.05$; Vincent: $P < 0.01$; Ackley: $P > 0.05$, all remaining benchmark functions: $P < 0.001$), as shown in Figure 2.8A. This holds also for different values of mutation rates (see Figure 2.16 for genetic diversity, and Figure 2.17 for number of unique solutions).

However, the better results achieved in terms of number of solutions discovered come at the cost of a longer evolutionary process for ViE in terms of number of evaluations before completion (1.42 ± 0.22 SD times longer than SSGA; see Figure 2.9 for non-aggregated results).

The higher diversity maintained throughout evolution enables ViE to be more effective on multi-modal problems with respect to SSGA by escaping regions of the fitness landscape with local optima and eventually discovering regions with global optima. ViE always outperforms SSGA in terms of successful repetitions of the algorithm (Figure 2.10), defined as those repetitions where the algorithm discovers at least one target solution. SSGA prematurely converges and is not able to discover any target solution in many repetitions even though a low selection

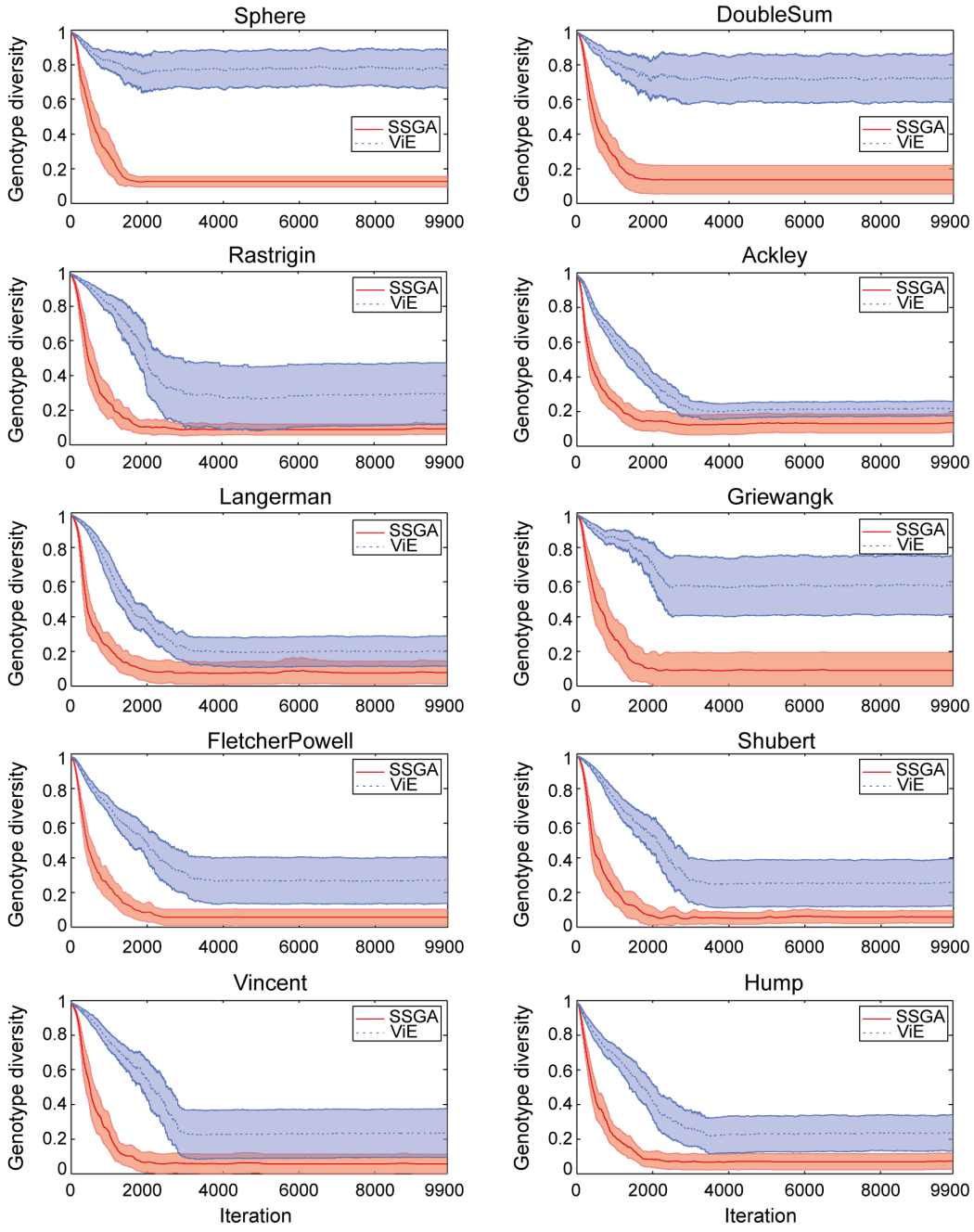


Figure 2.7: Genetic diversity maintained by SSGA and ViE. Average population genetic diversity (and confidence intervals) maintained during evolution for the 50 repetitions of each experiment.

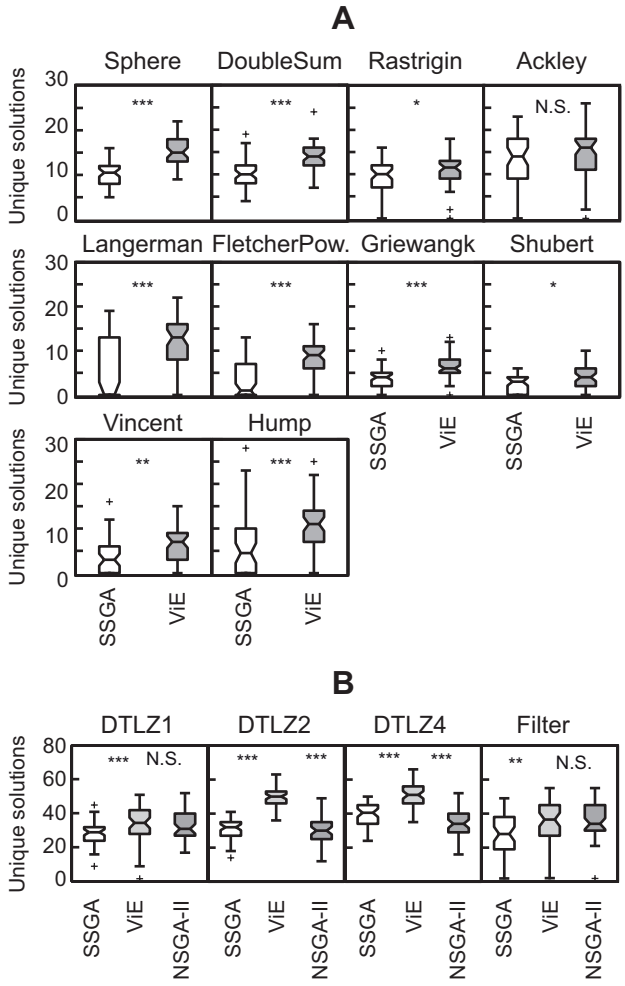


Figure 2.8: Number of unique target solutions discovered by SSGA and ViE. A) single-objective and B) multi-objective search landscapes. Each boxplot shows results for 50 repetitions of the algorithms on each function ($^{*}P < 0.05$, $^{**}P < 0.01$, $^{***}P < 0.001$, otherwise $P > 0.05$, Wilcoxon rank-sum test; N.S. not significant).

pressure was employed in these experiments (tournament selection with tournament size 2).

The ability of ViE and SSGA to discover solutions in disconnected target areas within a single repetition of the algorithm was investigated on search landscapes that feature a high number of disjoint target areas (> 10, Griewangk, Shubert, Vincent, Hump). Furthermore, both algorithms were tested with different initial population sizes ($M = 100, 250, 500, 750, 1000$ individuals) in order to assess if the algorithms could benefit from a larger, and potentially more diverse, initial population. ViE discovered more disconnected target areas than SSGA on all the search landscapes and for all initial population sizes ($P < 0.001$, Wilcoxon rank sum test; Hump: $P < 0.01$; Figure 2.11).

Additionally, the efficiency of the two algorithms were compared as the average number of

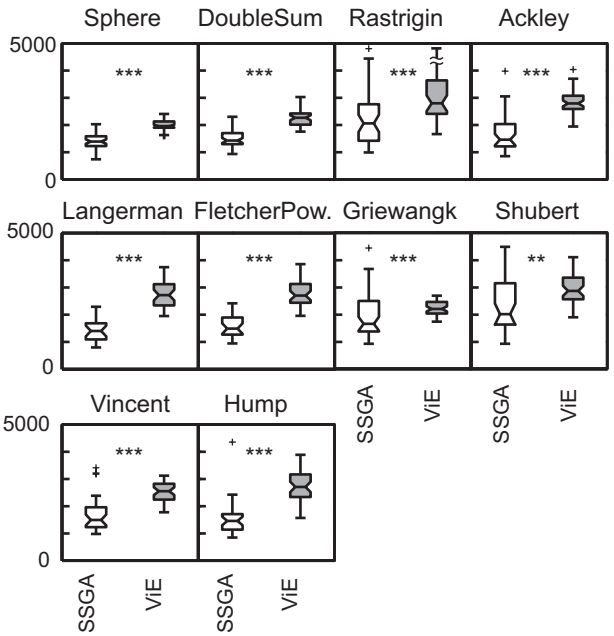


Figure 2.9: Number of iterations before completion of the evolutionary process for SSGA and ViE. Each box plot presents the results for 50 repetitions of the experiments on a different single-objective benchmark problem, as indicated by the titles above the boxes ($^{*}P < 0.05$,$^{**}P < 0.01$,$^{***}P < 0.001$, otherwise $P > 0.05$, Wilcoxon rank-sum test; N.S. not significant). The maximum value of the ViE Rastrigin boxplot (not representable otherwise) is 5478.

evaluations (individuals) necessary to find a single target area. ViE displayed significantly higher efficiency for all the tested initial population sizes (Wilcoxon rank-sum test, $P < 0.001$; Figure 2.12).

2.4.2 ViE compares favourably in "multi-objective" search landscapes against a multi-objective method with explicit diversity preservation

We compared ViE against the multi-objective optimization algorithm NSGA-II, which includes specific operators to maintain diversity in the evolving population. For sake of coherence with the results reported above, we also compared ViE and NSGA-II with SSGA endowed with a popular multi-objective technique, called weighted-sum approach (Deb, 2001), for combining multiple objective values into a single value. The three algorithms were assessed by counting the number of unique solutions that met the specified target performance for three search landscapes derived from standard multi-objective benchmark problems and for an electronic circuit design. ViE performed better than SSGA on all the multi-objective search landscapes (Figure 2.8B; Wilcoxon rank sum test, $P < 0.001$ for DTLZ benchmarks, and $P < 0.01$ for the circuit evolution experiment). ViE performed better than NSGA-II on all the search landscapes derived from multi-objective problems (Wilcoxon rank sum test, $P < 0.001$, except for DTLZ1 where $P > 0.05$) and performed as well as NSGA-II on the electronic circuit design (Wilcoxon

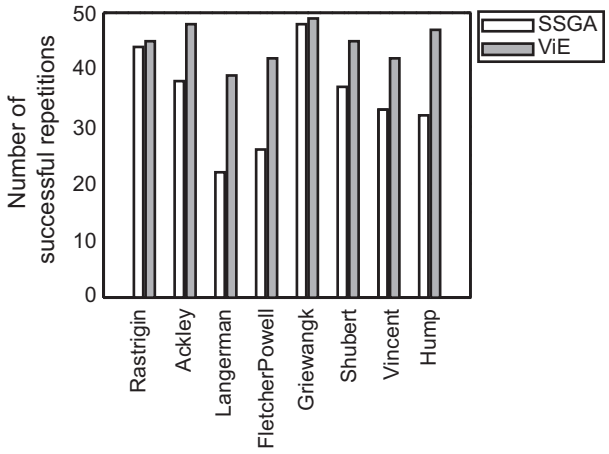


Figure 2.10: Number of successful repetitions for SSGA and ViE. Results for SSGA and ViE on single-objective, multi-modal problems out of a total of 50 repetitions.

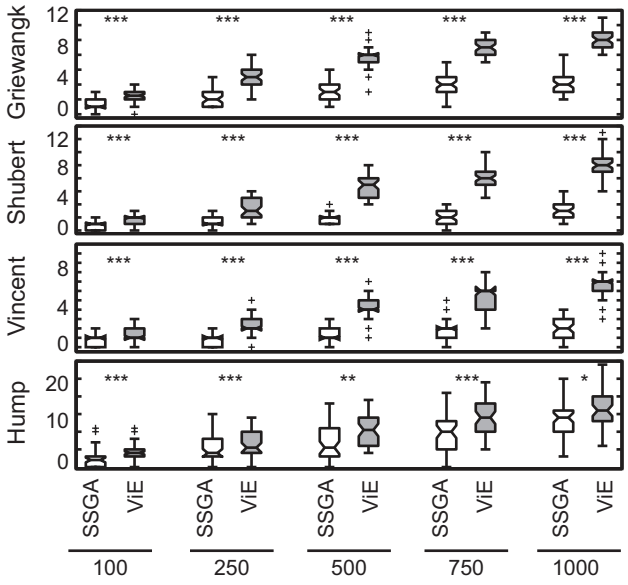


Figure 2.11: Number of disconnected target areas discovered by SSGA and Viability Evolution. Each box plot presents the results for different initial population sizes over 50 repetitions of the experiments ($^{*}P < 0.05$, $^{**}P < 0.01$, $^{***}P < 0.001$, otherwise $P > 0.05$, Wilcoxon rank-sum test; N.S. not significant). Viability Evolution can discover significantly more number of target areas for every initial population size ($P < 0.001$, except Hump where $P < 0.01$ for population size 500 and $P < 0.05$ for population size 1000, Wilcoxon rank sum test) than SSGA.

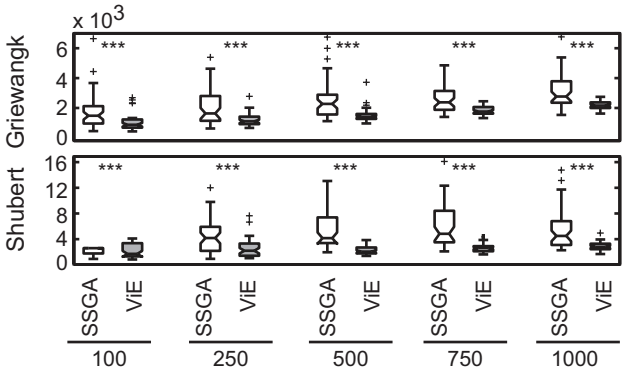


Figure 2.12: Efficiency of SSGA and Viability Evolution. Efficiency is measured as number of evaluations used per target area discovered over 50 repetitions of the experiment ($^{*}P < 0.05, ^{**}P < 0.01, ^{***}P < 0.001$, otherwise $P > 0.05$, Wilcoxon rank-sum test; N.S. not significant). A repetition of the evolutionary experiment lasts a higher number of evaluations in Viability Evolution. However, Viability Evolution is able to discover more target areas per repetition than SSGA. Its efficiency is significantly better than SSGA ($P < 0.001$, Wilcoxon rank sum test). To enhance readability of the box plots, we removed two outlier data points: Griewangk SSGA (500), Value 8676 and Griewangk SSGA (750), Value 12984. The computation of efficiency was performed only on Griewangk and Shubert, since the target areas in these benchmarks are regularly distributed in the search space and therefore have the same probability of being discovered.

rank sum test, $P > 0.05$).

2.4.3 Comparisons against methods that explicitly encourage diversity

Fitness sharing

One may argue that SSGA was not designed for the specific problem domain considered here, i.e., maximize the number of unique solutions discovered at completion of the evolutionary process, and that diversity preservation techniques might help SSGA achieve a higher number of unique final solutions. Thus, we compared Viability Evolution against SSGA endowed with a well-known diversity preservation technique, namely fitness sharing (Goldberg and Richardson, 1987), referred to as SSGA-FS.

We set the niche-radius parameter σ as suggested in (Deb and Goldberg, 1989). The niche-radius is computed using

$$\sigma = \frac{\sqrt{\sum_{k=1}^{p} (x_{k,max} - x_{k,min})}}{2\sqrt[p]{q}}$$

where p is the number of parameters, $x_{k,min}$ e $x_{k,max}$ are the decision space boundaries of

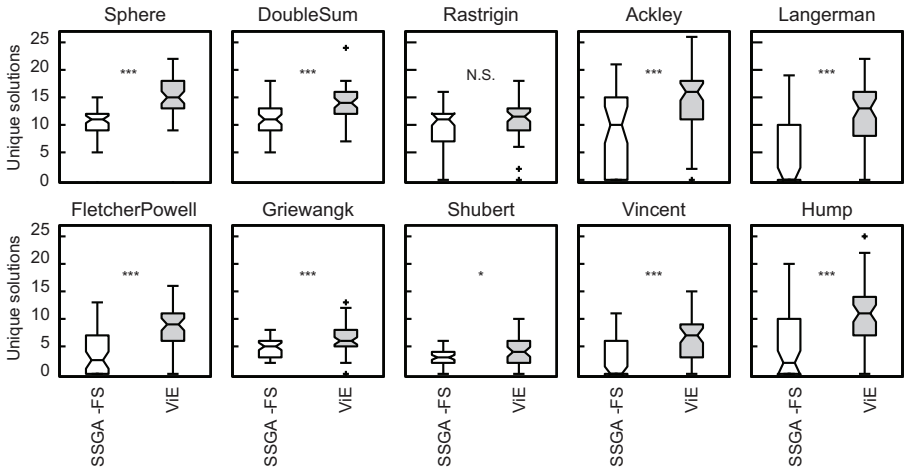


Figure 2.13: Number of unique target solutions discovered by SSGA-FS and ViE on single-objective search landscapes. Each plot shows results for 50 repetitions of the experiments on each function ($^{*}P < 0.05, ^{**}P < 0.01, ^{***}P < 0.001$, otherwise $P > 0.05$, Wilcoxon rank-sum test; N.S. not significant). As SSGA was originally designed to discover optimal solutions and not to maximize the number of unique solutions discovered at the final generation, we equipped it with a traditional diversity preservation mechanism, fitness sharing (Goldberg and Richardson, 1987), obtaining a modified version of SSGA named SSGA-FS. ViE can discover more unique solution than SSGA-FS in all benchmarks ($P < 0.001$; Shubert: $P < 0.01$, Wilcoxon rank sum test) except for Rastrigin, where results are not significantly different.

each parameter and q is the number of peaks (in our case disconnected target areas) in the fitness landscape. Niche-radius values for each benchmark problems are reported in Table 2.5.

Viability Evolution can discover more unique solution than SSGA with fitness sharing on all single-objective search landscapes ($P < 0.001$; Shubert: $P < 0.05$, Wilcoxon rank sum test, Figure 2.13) except for Rastrigin, where results are not significantly different.

Multi-objective methods with explicit diversity measure as objective

Also, we tested viability Evolution against another technique which adds an explicit objective to foster diversity. The multi-objective method NSGA-II was modified to optimize two objectives: minimize the distance to the target areas, and maximize the diversity of the current population. This resulting method is named NSGA-II-D. This second objective was computed for each individual as the average Hamming distance between the individual and the other individuals in the population. Viability Evolution can discover a higher number of unique target solutions than NSGA-II with a diversity objective on all the single-objective search landscapes ($P < 0.001$, Wilcoxon rank sum test, Figure 2.14).

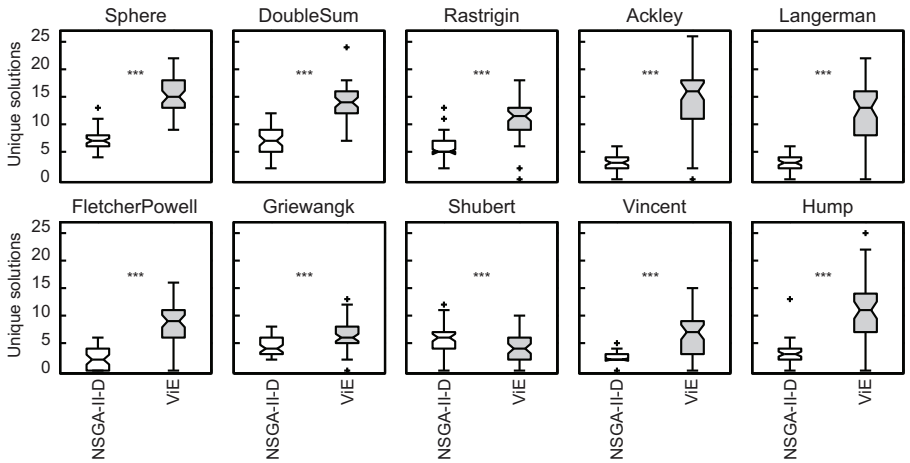


Figure 2.14: Number of unique target solutions discovered by NSGA-II with a diversity objective (NSGA-II-D) and ViE on single-objective search landscapes. Each plot shows results for 50 repetitions of the experiments on each function ($^{*}P < 0.05$,$^{**}P < 0.01$,$^{***}P < 0.001$, otherwise $P > 0.05$, Wilcoxon rank-sum test; N.S. not significant). ViE can discover a higher number of unique target solutions than NSGA-II-D on all the benchmark problems ($P < 0.001$, Wilcoxon rank sum test).

2.4.4 Contribution of each component of ViE in the discovery of unique solutions

The family mechanism employed by ViE to prevent the dominance of clonal individuals may contribute to diversity preservation. To disambiguate the contribution given by the family mechanism we performed additional control experiments where we compared the number of unique target solutions discovered by SSGA, ViE, SSGA equipped with the family mechanism (SSGA-F) and Viability Evolution without the family mechanism (ViE-noF) on single-objective (Figure 2.15A) and multi-objective search landscapes (Figure 2.15B).

Both SSGA-F and ViE equipped with the family mechanism obtain equal or better performance than their versions without it (ViE-noF and SSGA). However, ViE can discover more unique target solutions than SSGA-F in four benchmark problems (Langerman and Fletcher-Powell: $P < 0.05$; Hump and DTLZ2: $P < 0.01$, Wilcoxon rank sum test, Figure 2.15), and display performance similar to SSGA-F in the other benchmark problems. Also, ViE without family mechanism can discover more unique target solutions than SSGA on four benchmarks (FletcherPowell and DTLZ2: $P < 0.001$; Griewangk: $P < 0.01$; Rastrigin: $P < 0.05$, Wilcoxon rank sum test, Figure 2.15), and displays performance similar to SSGA in the other benchmark problems.

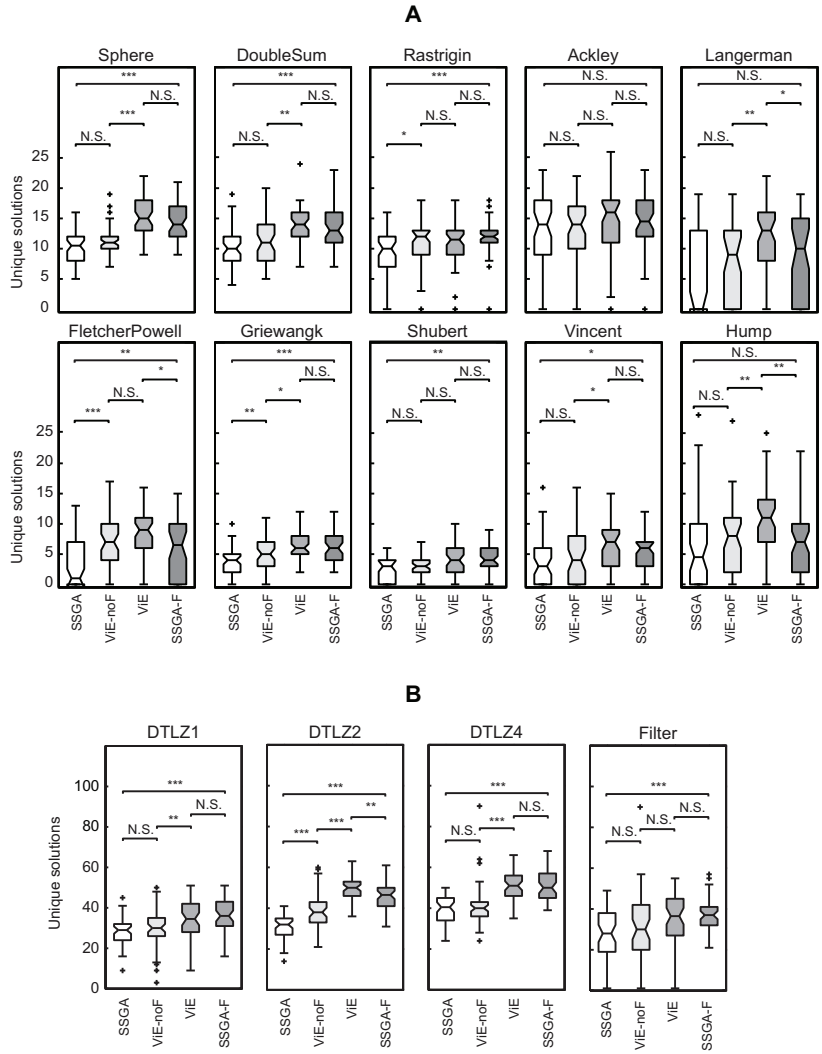


Figure 2.15: Number of unique target solutions discovered by SSGA, ViE, SSGA equipped with the family mechanism (SSGA-F) and Viability Evolution without the family mechanism (ViE-noF) on single- and multi-objective search landscapes. Each plot shows results for 50 repetitions of the experiments on each function ($^{*}P < 0.05, ^{**}P < 0.01, ^{***}P < 0.001$, otherwise $P > 0.05$, Wilcoxon rank-sum test; N.S. not significant). A) Single-objective search landscapes results. ViE can discover more unique target solutions than SSGA-F in three benchmark problems (Langerman and FletcherPowell: $P < 0.05$; Hump: $P < 0.01$, Wilcoxon rank sum test), displaying similar performance in the other benchmark problems. ViE-noF can discover more unique target solutions than SSGA on three benchmarks (FletcherPowell: $P < 0.001$; Griewangk: $P < 0.01$; Rastrigin: $P < 0.05$, Wilcoxon rank sum test), displaying similar performance in the other benchmark problems. B) Multi-objective search landscapes results. The contribution of the family mechanism always increases the performance of both SSGA and ViE respect to their versions without family mechanism ($P < 0.001$, except when comparing ViE and ViE-noF in DTLZ1: $P < 0.01$, Wilcoxon rank sum test). Moreover, in the DTLZ2 problem, Viability Evolution can obtain better performance than SSGA both when comparing SSGA against ViE-noF ($P < 0.001$, Wilcoxon rank sum test) and SSGA-F against ViE ($P < 0.01$, Wilcoxon rank sum test).

2.5 Discussion

In nature reproductive success depends on several factors that influence the probability of survival and reproduction of individuals. Two primary factors, as pointed out by Darwin (Darwin, 1859, p. 116), are the competition among individuals for scarce resources (selection of the fittest) and the ability of individuals to withstand current environmental conditions (elimination of the non-viable). Traditional evolutionary algorithms are inspired by competition-based reproductive success by ranking individuals according to their fitness and selecting only the best for reproduction.

Whenever elimination is considered in Evolutionary Computation (Atmar, 1994; Baum et al., 2001; Marín and Solé, 1999), individuals are eliminated according to their fitness score, thus falling into the competitive scenario of reproductive success. Viability Evolution, instead, models reproductive success as the ability of individuals to withstand current environmental conditions and eliminates individuals that are not viable due to the effect of random mutations or changing environmental conditions (viability boundaries). The use of boundaries had been previously advocated to constrain evolutionary search in specific regions of the search space (Storn, 1999), but boundary update was based on competition among individuals rather than elimination. Viability boundaries can be seen as a set of binary fitness functions with adaptive thresholds (Lässig and Hoffmann, 2009), and in this perspective, here we provide a self adaptive procedure for threshold selection. It had also been suggested (Juric, 1994) that giving equal chance of reproduction to individuals satisfying a minimal fitness level could result in higher variability of the evolved solutions, but no practical algorithm was proposed. A threshold defining the survival of individuals was used in (Lehman and Stanley, 2010), but the threshold was always fixed to a constant value. This method was later extended (Gomes et al., 2012), by progressively modifying the threshold. However, in both cases, the search was driven mainly by an objective promoting novelty of the solutions and the threshold was defined on a single objective. ViE does not use measures of novelty and drives the search by modifying viability boundaries on all problem objectives or constraints.

Viability Evolution can be used both for problem solving by defining target viability boundaries and for open-ended evolution by identifying viability boundaries that model the interactions between the evolving individuals and their environment (as in digital evolutionary ecosystems such as Tierra (Ray, 1991) and Avida (Adami, 2006; Ofria and Wilke, 2004)). Novel environmental conditions could be easily introduced by adding or deleting viability boundaries at any time during the process of artificial evolution.

Even though the elimination step of the Viability Evolution algorithm resembles at first sight existing survivor selection methods employed in genetic algorithms (Culling Method (Baum et al., 2001), Truncation Selection (Mühlenbein and Schlierkamp-Voosen, 1993), Extinctive Selection (Bäck, 1996)) the resulting evolutionary dynamics of ViE are different (see Figure 2.18 for a practical example comparing SSGA with Truncation Selection to ViE) and are due to the interplay of eliminations, varying size populations, changing viability boundaries and the

family mechanism. Although the family mechanism effectively contributes in some cases to the better performance in ViE, as shown in section 2.4.4, it is unavoidable in this simple version of ViE with fixed population size where "more fit" solutions discovered from the beginning of the evolutionary process can bias the search towards them by having more "evolutionary time" to generate similar solutions than other less fit individuals. More advanced versions of a viability algorithm that could exploit dynamic populations and "aging" mechanisms of solutions may get rid of this algorithmic procedure and keep a truly unbiased reproduction.

To illustrate the advantages of the novel operational principle, we compared ViE to a canonical competition-based Evolutionary Algorithm, namely SSGA, without any state-of-the-art explicit diversity preservation techniques such as niching, maintenance of sub-populations, etc. However, one may argue that SSGA was not been designed for the specific problems domain considered here, i.e., maximize the number of unique solutions discovered at completion of the evolutionary process. Therefore, we also compared Viability Evolution with two genetic algorithms with diversity preservation techniques: SSGA endowed with fitness sharing (Goldberg and Richardson, 1987) and NSGA-II with an explicit objective for diversity. In both cases however, ViE could discover on almost all problems a higher number of unique and more diverse solutions.

Moreover, when using diversity preservation methods, one should consider that instrumenting an evolutionary method with such techniques usually requires the definition of additional parameters (for example a niching radius (Deb and Goldberg, 1989), or a niche capacity (Petrowski, 1996)), which are difficult to identify because the search landscape is unknown, or depends on measures of diversity in genotypic or phenotypic space (Deb and Goldberg, 1989; Petrowski, 1996; Lehman and Stanley, 2008; Toffolo and Benini, 2003; Lehman and Stanley, 2011), or requires keeping an archive of "diverse" solutions (Lehman and Stanley, 2008; Lehman and Stanley, 2011). Viability Evolution does not require the definition of additional niching parameters, diversity measures or the maintenance of an additional archive of solutions. Nonetheless, these explicit diversity preservation techniques are also applicable to ViE, and could possibly increase its performance too.

Although the dimensions of the problems in this study were kept small to make clear conclusions about the effectiveness of the EAs, the SSGA already fails to find target solutions in many runs (see Figure 2.10). Even though the scalability of the proposed approach to problems of higher dimensionality remains to be investigated, it must be considered that we presented here one of the possible procedures to update the boundaries (indeed a very simple one, to ease the comparison with respect to existing algorithms). The boundary update procedure presented here modifies all the boundaries together. This however is not a necessity as some boundaries may be harder to satisfy than others and may benefit from a differential update speed of each boundary.

For example, each boundary update could be made proportional to the ratio of viable/unviable individuals for the corresponding objectives. In the future, more sophisticated procedures

might be introduced, taking into account multiple factors to define which and by how much a viability boundary should be tightened (or relaxed), possibly enhancing the performance of ViE to address large-scale optimization problems.

2.6 Conclusion

Beside the better results in terms of number of unique solutions discovered by ViE on multi-modal and multi-objective problems (with the exception of the Ackley function and the electronic circuit design where ViE and NSGA-II reported the same performance), in Viability Evolution it is not necessary to aggregate multiple objectives or constraints into a single fitness function. Considering the well-known difficulty of designing fitness functions for multi-objective problems, this is a significant advantage even when ViE performs as well as other traditional evolutionary algorithms that require the formulation of an aggregated fitness function. When compared to multi-objective methods that also do not aggregate fitness, ViE offers a different approach that does not use partial ordering between individuals, as non-dominated sorting does in current multi-objective methods. Incidentally, the definition of viability boundaries in ViE is similar to the engineering practice of designing artefacts that meet desired operating ranges, such as temperature, voltage, frequency output, etc., which can be found in the specification list of any electronic or mechanical product on the market.

Although one the main purpose of this chapter is to show that artificial evolution can be performed with the sole use of viability based eliminations, ViE is compatible with the competition-based approaches and could be extended to encompass forms of competition-driven reproduction by introducing higher reproduction rates of viable individuals whose fitness could be computed while keeping unchanged all other aspects of the algorithm. A suitable combination of viability-based elimination and competition-based reproduction would allow a user to preferentially select for individuals with specific features within a diverse population of viable individuals and would provide a comprehensive evolutionary framework that models both competition and viability in natural evolution.

The investigation performed in this chapter suggests one possible advantage deriving from adopting Viability Evolution principles: increased genetic diversity in the evolving population. However, the algorithm presented here is far from being applicable to real-world problems, as it is based on a simple genetic algorithm. In the next chapter, we try to show a second advantage that the adoption of the viability paradigm can bring: additional information is made available during the search process by monitoring the generation of viable and non-viable individuals. This additional information will be used in the next chapters to derive efficient viability-based evolutionary algorithms able to compete with state-of-the-art EC methods.

2.7 Supporting Information

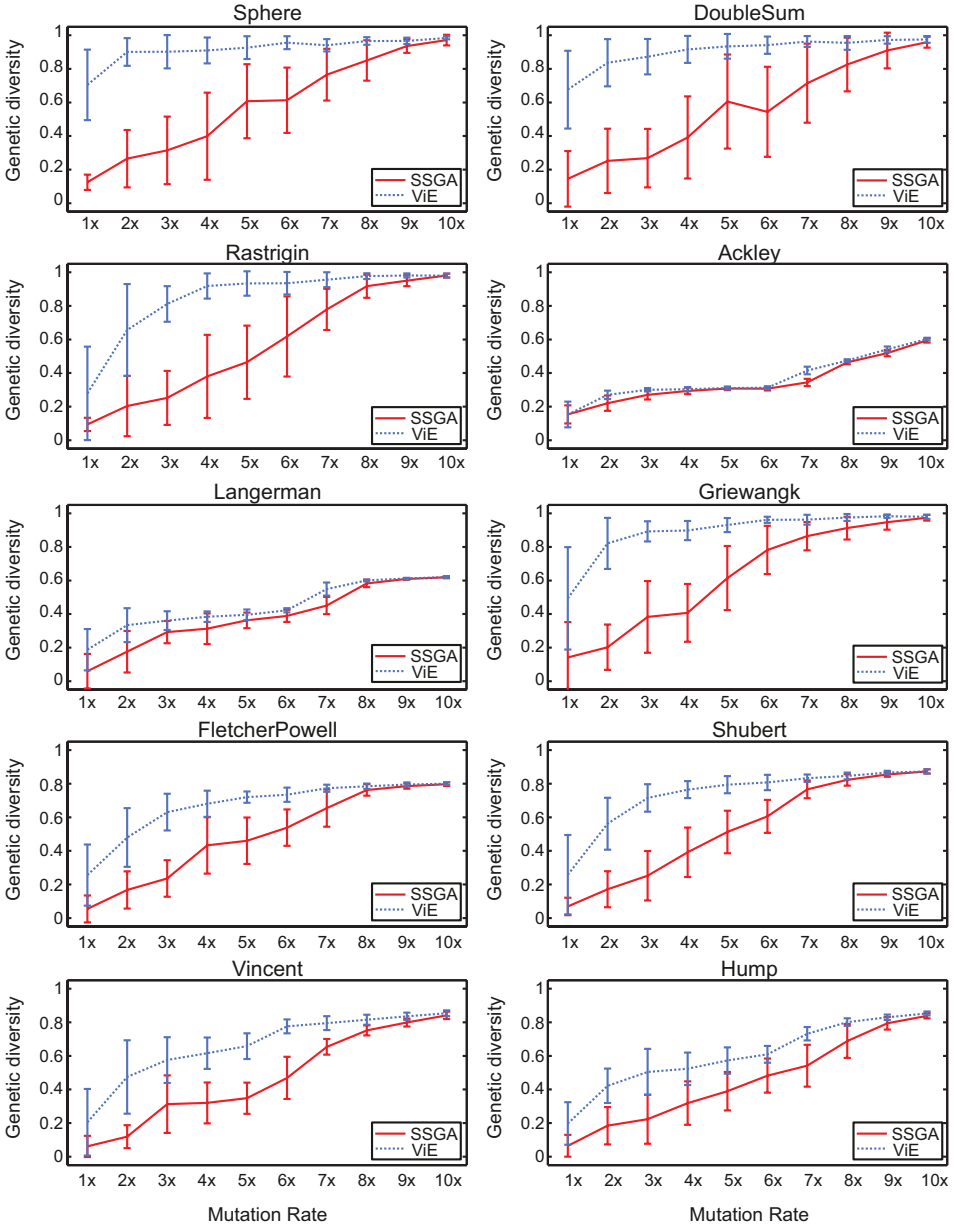


Figure 2.16: Genetic diversity of unique target solutions discovered by SSGA and ViE on single-objective search landscapes, varying the mutation rate up to 10 times its original value. Mutation, in the original configuration (1x), consisted of flipping each bit of the genotype with probability $\frac{1}{l}$ where l is the genome length. Each plot shows results for 25 repetitions of the experiments on each function. In general, genetic diversity increases with mutation rates. However, high genetic diversity obtained using high mutation rate does not always result into a higher number of discovered target solutions (Figure 2.17).

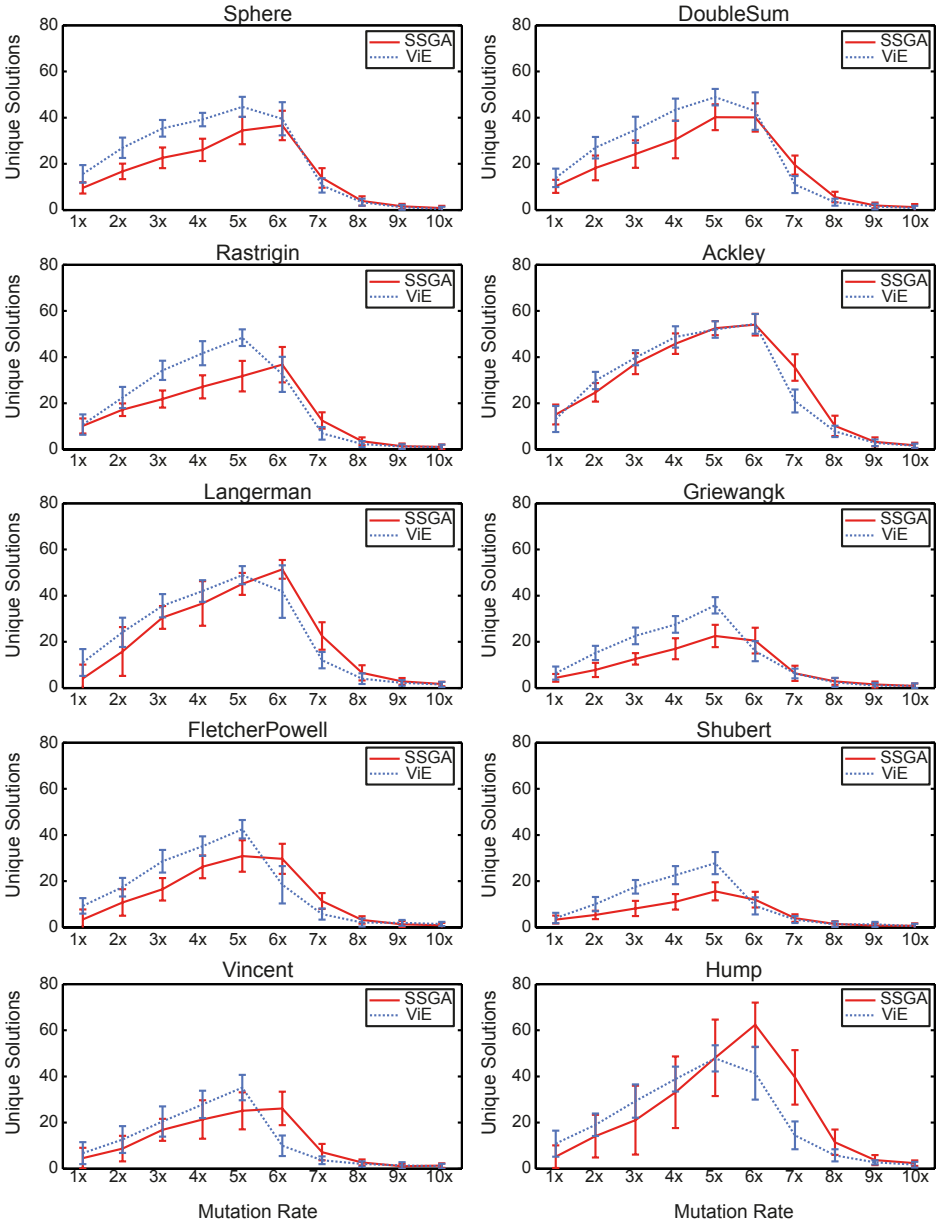


Figure 2.17: Number of unique target solutions discovered by SSGA and ViE on single-objective search landscapes, varying the mutation rate up to 10 times its original value. Each plot shows results for 25 repetitions of the experiments on each function. Mutation, in the original configuration (1x), consisted of flipping each bit of the genotype with probability $\frac{1}{l}$ where l is the genotype length. Each plot shows results for 50 repetitions of the experiments on each function.

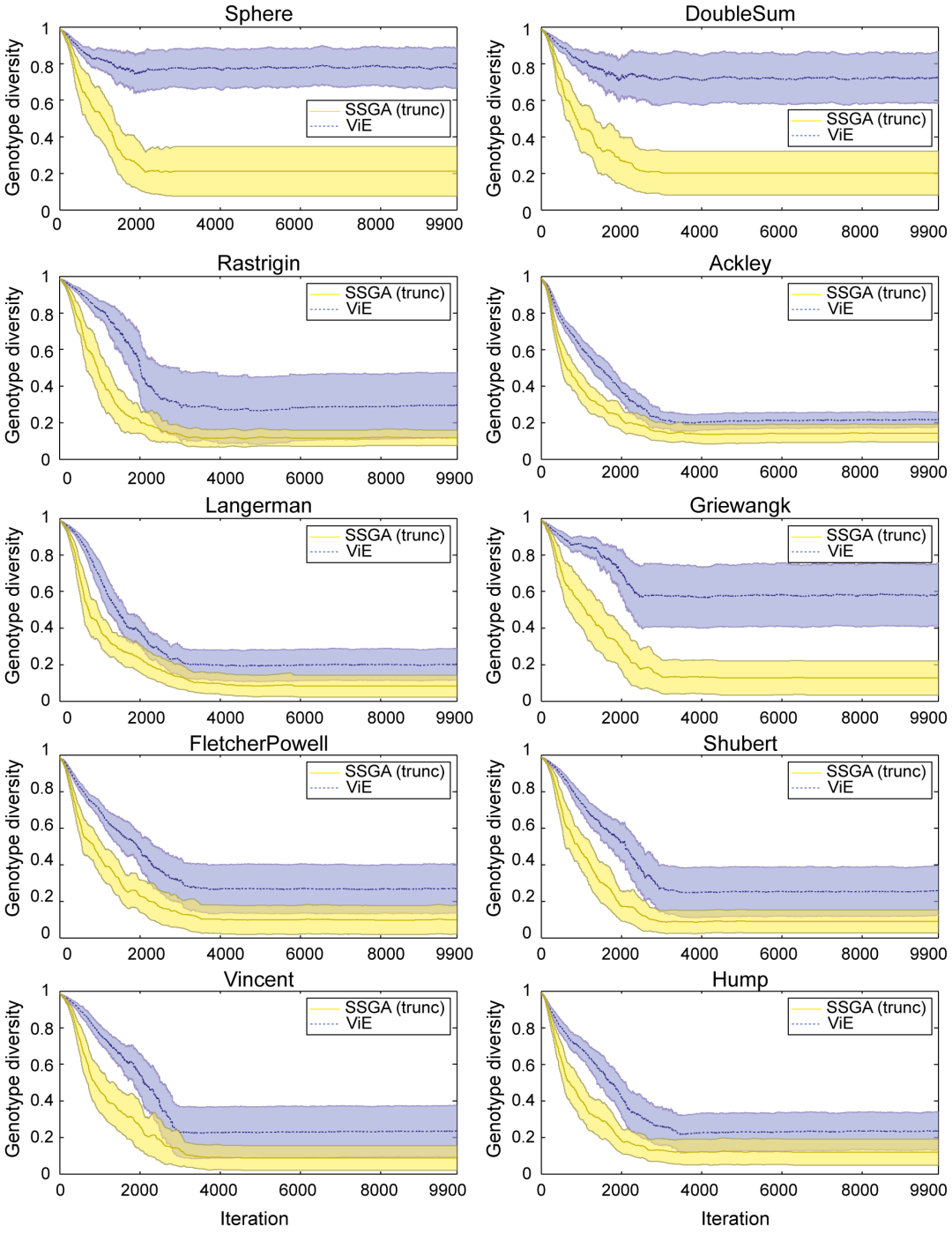


Figure 2.18: Average population genetic diversity (and confidence intervals) maintained by SSGA (with truncation selection) and Viability Evolution over 50 repetitions of the experiments. Even though at first sight the update method we used in ViE to tighten the viability boundaries may seem similar to SSGA with truncation selection (with an unusual high level of selection of 95% of the population), the evolutionary dynamics of these two algorithms are remarkably different).

Table 2.5: Niche-radius values for SSGA with fitness sharing in single-objective benchmarks. The values are derived from the formula suggested in (Deb and Goldberg, 1989).

Benchmark	**Niching radius**
Sphere	7.240773
Double Sum	92.681900
Rastrigin	7.240773
Ackley	35.355339
Langerman	7.071068
Fletcher-Powell	2.221440
Griewangk	235.339362
Shubert	3.333333
Vincent	1.149049
Hump	0.117851

3 Information from viability boundaries to build efficient adaptive algorithms

In the previous chapter we introduced a simple version of a Viability Evolution algorithm and tested its feasibility as an evolutionary method. We wanted to highlight one of the advantages that can derive from adopting the Viability Evolution paradigm: the increased diversity in the population during evolution. Here, we suggest a second advantage that the viability paradigm could provide: viability boundaries can also be used as an additional source of information for evolutionary algorithms. Furthermore, in the previous chapter our purpose was not to provide a competitive version of Viability Evolution with the state-of-the-art. Instead, in this chapter, we focus our attention on a more applied field of Evolutionary Computation, constrained optimization, with the explicit purpose of deriving more efficient evolutionary methods. We embed Viability Evolution principles on a state-of-the-art algorithm for constrained optimization and we show how additional information gathered from viability boundaries can be used for self-adapting the algorithm's parameters. The resulting method displays competitive performance when tested on unimodal problems from a well-known constrained optimization benchmark set.

The contents of this chapter are adapted from:

Maesani A, Floreano D (2014). Viability Principles for Constrained Optimization Using a (1+1)-CMA-ES. Proc. of the 13th Int. Conf. on Parallel Problem Solving from Nature, Lubjana, 2014

3.1 Introduction

Evolutionary computation methods are often used to solve real-valued black-box optimization problems, a large number of which require satisfying constraints. Without loss of generality, solving a real-valued constrained optimization problem in $\mathbb{R}^n$ means minimizing the objective function $f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^n$, subject to inequalities[1] defined on m constraints function $g_i(\boldsymbol{x}) \leq 0, i = 1,\ldots,m$.

Several approaches have been proposed to solve constrained problems using evolutionary algorithms (Mezura-Montes and Coello Coello, 2011), ranging from rejecting solutions that violate constraints (infeasible solutions) to more sophisticated strategies that modify the ranking of individuals by penalizing the fitness using a function of constraint violations (penalty functions). Other popular approaches include stochastic ranking of solutions (Runarsson and Yao, 2000), ϵ-constrained optimization (Takahama and Sakai, 2010), feasibility rules to rank solutions (Deb, 2000), and transformation of constraints into objectives. Although these methods are necessary to handle infeasible solutions and constraints, an efficient optimizer is essential to progress during the search.

Currently, many state-of-the-art algorithms for unconstrained optimization are based on Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen et al., 2003). In CMA-ES, a covariance matrix describing correlations between decision variables is learned and adapted during the search to maximize the likelihood of generating successful solutions. Although CMA-ES is a powerful optimizer in unconstrained settings (Hansen et al., 2010), it may suffer from premature convergence in presence of constraints, a common problem in strategies with adaptive step-size control (Kramer and Schwefel, 2006). Furthermore, methods for constrained optimization based on CMA-ES often require providing a feasible solution as a starting point.

A different modelling of objectives and constraints in CMA-ES may offer novel possibilities for handling constraints and allow the initialization of the algorithm from infeasible solutions. Viability Evolution (Mattiussi and Floreano, 2003; Maesani et al., 2014) is an abstraction of artificial evolution that models an optimization process using viability boundaries, which are modified over time to drive the search towards desirable regions of a search space, as shown in Figure 3.1. Under this abstraction, mutations can produce viable solutions, which survive, or non-viable solutions, which are eliminated from the population. Viability boundaries are generally defined as admissible ranges of problem objectives and constraints. At the beginning of the search the boundaries are relaxed to encompass all randomly generated initial solutions and then gradually tightened. Once viability boundaries reach the desired target boundaries they are not tightened further, and the evolutionary process is considered complete.

In this chapter, we borrow concepts from Viability Evolution, and combine them with active covariance updates for CMA-ES (Arnold and Hansen, 2012), to derive a novel algorithm for

[1] Equality constraints can always be rewritten as inequalities by using a tolerance value on the equality.

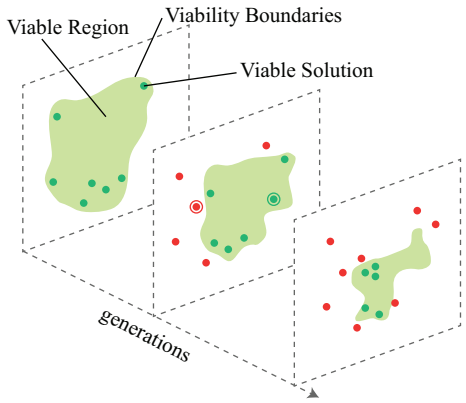


Figure 3.1: Viability boundaries initially encompass all randomly generated solutions. We represent the viable region as a projection on a two-dimensional plane of the viability boundaries (shaded area). During the search, the boundaries are made more stringent. Viable solutions are retained in the population (dots in the shaded area), whereas solutions that do not satisfy viability boundaries are eliminated. Mutations can generate solutions (circled dots) that fall outside or inside the viable region.

constrained optimization. Here, we restrict ourselves to testing our method only in the case where it is started from a feasible solution, as done in (Arnold and Hansen, 2012) which reports current state-of-the-art performance on a set of eight unimodal functions.

The chapter is structured as follows. In Section 3.2 we discuss the state-of-the-art on constraint handling in evolution strategies and we elucidate the workings of a (1+1)-CMA-ES with constraint handling proposed in (Arnold and Hansen, 2012). In Section 3.4 we discuss Viability Evolution principles and the proposed approach for constrained optimization. Experimental setup and results of the proposed approach are presented in Section 3.5. Finally, we conclude with a brief discussion of the proposed approach in Section 3.6, and we propose future continuations of the work.

3.2 Related Work

Classical approaches to handle constraints in evolution strategies consist of simply discarding and resampling infeasible solutions (Schwefel, 1993) or using penalty functions. Penalty functions usually depend on the amount of constraints violation or number of violated constraints (Hoffmeister and Sprave, 1996), and in some cases also on the fitness of selected feasible solutions (Oyman et al., 1999). The penalty functions can also be adaptive: for example the relative weight of each constraint in the penalty can be modified according to the number of iterations where infeasible solutions are discovered (Collange et al., 2010), or according to the ratio between feasible and infeasible individuals (Kramer et al., 2013).

Other methods do not use penalty functions. An approach performs selection based on three feasibility rules (Mezura-Montes and Coello Coello, 2005): feasible individuals are

compared on objectives, infeasible ones are compared on total constraint violations, and feasible individuals are always ranked before infeasible ones. Similarly, a recently proposed method modifies the ranking of individuals based on three independent rankings: by objective function, by constraint violation amount, and by number of violated constraints depending on if the solution is feasible or infeasible (Kusakci and Can, 2013b). Other approaches reduce the probability of generating infeasible solutions when in the proximity of the constraint, by moving the mean of the population (Kramer et al., 2005) or by explicitly controlling the step size using a lower bound (Kramer and Schwefel, 2006).

Another way in which constraints can be handled is learning surrogate models for linear constraints. One of these methods has been shown to be a promising approach to reduce the number of constraint function evaluations by predicting if solutions are feasible or infeasible, adapting directly the covariance matrix using the learned information, and repairing solutions that turn out to be infeasible (Kramer et al., 2009). The work has been recently extended to non-linear constraints, learning models using support vector machines (Gieseke and Kramer, 2013). Another recently proposed variant of CMA-ES (Beyer and Finck, 2012) makes use of repair mechanisms, but the algorithm is very specific to the problem being solved (financial portfolio optimization).

3.3 (1+1)-CMA-ES with active covariance matrix adaptation

Among the various methods proposed for handling constraints in CMA-ES, Arnold and Hansen (Arnold and Hansen, 2012) recently proposed a modification of a (1+1)-CMA-ES that has displayed great performance improvements with respect to other methods on unimodal constrained problems when started from a feasible solution. The method maintains a (low-pass filtered) vector representing the direction of violations of steps with respect to each constraint. These vectors are used to update the covariance matrix such that the variance in the direction of violation is reduced. A (1+1)-CMA-ES combines (1+1) selection (Beyer and Schwefel, 2002) with covariance matrix adaptation (Hansen et al., 2003). Given a parent solution $\boldsymbol{x} \in \mathbb{R}^n$, an offspring solution $\boldsymbol{y}$ is sampled according to $\boldsymbol{y} \leftarrow \boldsymbol{x} + \sigma \boldsymbol{A}\boldsymbol{z}$ where $\boldsymbol{A}$ is the Choleski decomposition of the covariance matrix $\boldsymbol{C} = \boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{z} \sim \mathcal{N}(0, \mathbf{I})$ is sampled from a normal distribution. The global step size $\sigma \in \mathbb{R}_+$ is changed according to a modified 1/5 rule proposed in (Igel et al., 2006). The probability $P_{succ} \in [0,1]$ of generating successful solutions and σ are updated at each iteration

$$P_{succ} \leftarrow (1-c_p)P_{succ} + c_p \mathbb{1}_{f(\boldsymbol{y}) \leq f(\boldsymbol{x})} \tag{3.1}$$

$$\sigma \leftarrow \sigma e^{\left(\frac{1}{d}\left(P_{succ} - \frac{P_{target}}{1-P_{target}}(1-P_{succ})\right)\right)} \tag{3.2}$$

where $\mathbb{1}_{f(\boldsymbol{y})\leq f(\boldsymbol{x})}$ is 1 if the condition is true or 0 otherwise, the learning rate $c_p \in (0,1]$ determines the fading of P_{succ} and the damping factor d controls the step size variation. P_{target} determines the probability threshold that decreases or increases σ. The covariance matrix is adapted using the original rank-one update rule of CMA-ES, $\boldsymbol{C}^{(g+1)} = \alpha\boldsymbol{C}^{(g)} + \beta\boldsymbol{v}^{(g)}\boldsymbol{v}^{(g)^T}$, which increases the variance in the direction of the provided vector $\boldsymbol{v}$ from one iteration g to the following one. Using a vector of fading successful steps $\boldsymbol{s}$, called the evolution path, in place of vector $\boldsymbol{v}$, allows the strategy to increase the likelihood of sampling new solutions in the direction of already successful steps. In fact, there is no need to maintain the covariance matrix $\boldsymbol{C}$, as updates can be performed directly on the Choleski factor $\boldsymbol{A}$ as proved in (Igel et al., 2006) according to

$$\boldsymbol{A} \leftarrow \sqrt{\alpha}\boldsymbol{A} + \frac{\sqrt{\alpha}}{\|\boldsymbol{w}\|^2}\left(\sqrt{1+\frac{\beta}{\alpha}\|\boldsymbol{w}\|^2}-1\right)\boldsymbol{s}\boldsymbol{w}^T \tag{3.3}$$

where $\boldsymbol{w} = \boldsymbol{A}^{-1}\boldsymbol{s}$ and $\beta = c_{cov}^{+} \in \mathbb{R}^n$. In practice the evolution path $\boldsymbol{s}$ and α are updated depending on P_{succ}. If the probability of success is small ($P_{succ} < P_{thresh}$) then the covariance matrix is updated considering the current step $\boldsymbol{A}\boldsymbol{z}$, such that $\boldsymbol{s} \leftarrow (1-c)\boldsymbol{s} + \sqrt{c(2-c)}\boldsymbol{A}\boldsymbol{z}$ and $\alpha = 1-c_{cov}^{+}$. Otherwise ($P_{succ} \geq P_{thresh}$), the update does not consider the current step in order to avoid the variance increasing too much in the direction of already successful mutations. In this case the covariance matrix is always updated using Equation 3.3 but the evolution path is set to $\boldsymbol{s} \leftarrow (1-c)\boldsymbol{s}$ and $\alpha = 1 - c_{cov}^{+} + c_{cov}^{+}c(2-c)$.

An alternative “active" covariance matrix update that also considers particularly unsuccessful steps worse than the fifth ancestor of the current solution was proposed in (Arnold and Hansen, 2010). In this case, the covariance matrix is updated using the current unsuccessful step $\boldsymbol{A}\boldsymbol{z}$ and the following rule that decreases the variance in the direction of that step[2]

$$\boldsymbol{A} \leftarrow \sqrt{\alpha}\boldsymbol{A} + \frac{\sqrt{\alpha}}{\|\boldsymbol{z}\|^2}\left(\sqrt{1-\frac{\beta}{\alpha}\|\boldsymbol{z}\|^2}-1\right)\boldsymbol{A}\boldsymbol{z}\boldsymbol{z}^T \tag{3.4}$$

where $\alpha = \sqrt{1+c_{cov}^{-}}$ and $\beta = c_{cov}^{-}$.

Interestingly, the same rule can be used to decrease variance in the direction of constraint violations. Similarly to what is done with the evolution path, Arnold and Hansen (Arnold and Hansen, 2012) proposed to use fading vectors of steps that violate constraints in combination with active covariance updates. Specifically, for each constraint i that is violated by step $\boldsymbol{A}\boldsymbol{z}$, the vector $\boldsymbol{v}_i \leftarrow (1-c_c)\boldsymbol{v}_i + c_c\boldsymbol{A}\boldsymbol{z}$ is updated. Whenever even a single constraint is violated, the covariance matrix is updated according to

$$\boldsymbol{A} \leftarrow \boldsymbol{A} - \frac{B}{\sum_{i=1}^{m}\mathbb{1}_{g_i(\boldsymbol{y})>0}}\sum_{i=1}^{m}\mathbb{1}_{g_i(\boldsymbol{y})>0}\frac{\boldsymbol{v}_i\boldsymbol{w}_i^T}{\boldsymbol{w}_i\boldsymbol{w}_i^T} \tag{3.5}$$

[2]Note that the sign in the parenthesis is inverted. Furthermore, if $\|z\|^2 \geq \frac{1+c_{cov}^{-}}{2c_{cov}^{-}}$ then $c_{cov}^{-} = \frac{1}{2\|z\|^2-1}$.

where $\boldsymbol{w_i} = \boldsymbol{A}^{-1}\boldsymbol{v_i}$. The parameters used in the algorithm are set to the following (Arnold and Hansen, 2010). We will refer to this method in the following as (1+1)-acCMA-ES (active constrained CMA-ES).

Table 3.1: Parameter setting for the (1+1)-ViE. Parameters are set as in the original Arnold's paper (Arnold and Hansen, 2010)

$c = \frac{2}{n+2}$	$c_c = \frac{1}{n+2}$	$c_p = \frac{1}{12}$
$d = 1 + \frac{n}{2}$	$B = \frac{0.1}{n+2}$	$P_{target} = \frac{2}{11}$
$c^+_{cov} = \frac{2}{n^2+6}$	$c^-_{cov} = \frac{0.4}{n^{1.6}+1}$	$\sigma = 0.1$

3.4 Introducing viability principles in CMA-ES

Modelling an evolutionary algorithm using the Viability Evolution abstraction offers novel possibilities. For example, in the case of constrained optimization viability boundaries can be defined to relax problem constraints at the beginning of the search, and be made more stringent over time to lead solutions into the feasible regions. The key idea proposed here is to use changing viability boundaries that define admissible regions of the search space (viable regions) in combination with the active covariance matrix updates proposed by Arnold and Hansen (Arnold and Hansen, 2012). Active covariance updates are used to decrease the variance in the direction of boundary violations. As the boundaries defined on constraint functions values can be relaxed, the algorithm is compatible with infeasible starting solutions. On the other hand, whenever a viable solution is generated, the standard covariance matrix update rule of (1+1)-CMA-ES is employed to increase the variance in the direction that generated the viable solution. Because different boundaries may affect the global probability of generating viable solutions P_{succ}, we maintain a vector of probability of success $\boldsymbol{p}_{succ}$, that tracks which boundary is more likely to cause the generation of non viable solutions. As depicted in Figure 3.2A, when the covariance matrix is well adapted to a boundary, the probability of generating a new viable solution is greater or equal to 50%. Otherwise, when the probability of success is lower than 50% for at least one boundary, as shown in Figure 3.2B, the covariance matrix should be modified and the global step size reduced. To achieve this, we reduce the global P_{succ} probability. Conversely, the overall P_{succ} probability and all elements of the $\boldsymbol{p}_{succ}$ vector are increased whenever a viable solution is generated. Note that in the method presented in (Arnold and Hansen, 2012) not adapting P_{succ} on failure may lead to the use of outdated information for step-size adaptation.

The pseudo-code of our method, referred to as (1+1)-ViE, is presented in Algorithm 2. The user

Algorithm 2 (1+1)-VIE-CMA-ES pseudo-code. Problem objectives and constraints are modelled using the viability boundaries abstraction. Parameters $d, c, c_c, c_p, B, P_{target}, c^+_{cov}$ and c^-_{cov} are defined as in (Arnold and Hansen, 2010).

Require: $\sigma \in \mathbb{R}_+$ initial global step size

$\alpha \leftarrow 1 - c^+_{cov}$, $\beta \leftarrow c^+_{ccov}$, $s \leftarrow 0$
$\boldsymbol{A} \leftarrow \mathbf{I}$
for $i = 1 \dots m+1$ **do**
    $\boldsymbol{v}_i \leftarrow [0, \dots, 0]_{n\times 1}$ ▷ The last $\boldsymbol{v}_i$ and b_i correspond to the objective
end for
$\boldsymbol{b} \leftarrow [max(0, g_1(\boldsymbol{x})), \dots, max(0, g_m(\boldsymbol{x})), \infty]$
$\boldsymbol{p}_{succ} \leftarrow [\frac{1}{2}, \dots, \frac{1}{2}]$
$\boldsymbol{x} \leftarrow$ randomly generate solution
while ¬ termination condition **do**
    $\boldsymbol{z} \sim \mathcal{N}(0, \mathbf{I})$
    $\boldsymbol{y} \leftarrow \boldsymbol{x} + \sigma \boldsymbol{A}\boldsymbol{z}$
    $\boldsymbol{V} \leftarrow [\mathbb{1}_{g_1(\boldsymbol{y})>b_1}, \dots, \mathbb{1}_{g_m(\boldsymbol{y})>b_m}, \mathbb{1}_{f(\boldsymbol{y})>b_{m+1}}]$ ▷ Boundary violations
    if $\exists\, i : \boldsymbol{V}_i = 1$ **then**
        for all $i : \boldsymbol{V}_i = 1$ **do**
            $\boldsymbol{v}_i \leftarrow (1 - c_c)\boldsymbol{v}_i + c_c \boldsymbol{A}\boldsymbol{z}$
            $\boldsymbol{w}_i \leftarrow \boldsymbol{A}^{-1}\boldsymbol{v}_i$
        end for
        $\boldsymbol{A} \leftarrow \boldsymbol{A} - B\sum_{i=1}^{m} \mathbb{1}_{g_i(\boldsymbol{y})>0} \frac{\boldsymbol{v}_i \boldsymbol{w}_i^T}{\boldsymbol{w}_i \boldsymbol{w}_i^T}$ ▷ Decrease variance
        $\boldsymbol{p}_{succ} \leftarrow (1 - c_p)\boldsymbol{p}_{succ} + c_p[\mathbb{1}_{\boldsymbol{V_1}=0}, \dots, \mathbb{1}_{\boldsymbol{V_{m+1}}=0}]$ ▷ Update success probability
        if $\exists\, i : \boldsymbol{p}_{succ_i} < \frac{1}{2}$ **then**
            $P_{succ} \leftarrow (1 - c_p)P_{succ}$ ▷ Decrease global P_{succ}
        end if
    else
        $P_{succ} \leftarrow (1 - c_p)P_{succ} + c_p$ ▷ Increase success probabilities
        $\boldsymbol{p}_{succ} \leftarrow (1 - c_p)\boldsymbol{p}_{succ} + c_p$
        $\sigma \leftarrow \sigma exp\left(\frac{1}{d}\left(P_{succ} - \frac{P_{target}}{1-P_{target}}(1 - P_{succ})\right)\right)$
        $\boldsymbol{s} \leftarrow (1 - c)\boldsymbol{s} + \sqrt{c(2-c)}\boldsymbol{A}\boldsymbol{z}$
        $\boldsymbol{w} \leftarrow \boldsymbol{A}^{-1}\boldsymbol{s}$
        $\boldsymbol{A} \leftarrow \sqrt{\alpha}\boldsymbol{A} + \frac{\sqrt{\alpha}}{\|\boldsymbol{w}\|^2}\left(\sqrt{1 + \frac{\beta}{\alpha}\|\boldsymbol{w}\|^2} - 1\right)\boldsymbol{s}\boldsymbol{w}^T$
        $\boldsymbol{b}_{1..m} \leftarrow \Big[max\Big(0, min\Big(b_1, g_1(\boldsymbol{y}) + \frac{b_1 - g_1(\boldsymbol{y})}{2}\Big)\Big), \dots,$
            $max\Big(0, min\Big(b_m, g_m(\boldsymbol{y}) + \frac{b_m - g_m(\boldsymbol{y})}{2}\Big)\Big)\Big]$
        if $V_{i:1,\dots,m} = 0$ **then** ▷ Update boundary on objective when feasible
            $\boldsymbol{b}_{m+1} \leftarrow f(\boldsymbol{y}) + \frac{f(\boldsymbol{x}) - f(\boldsymbol{y})}{2}$
        end if
        $\boldsymbol{x} \leftarrow \boldsymbol{y}$
    end if
end while

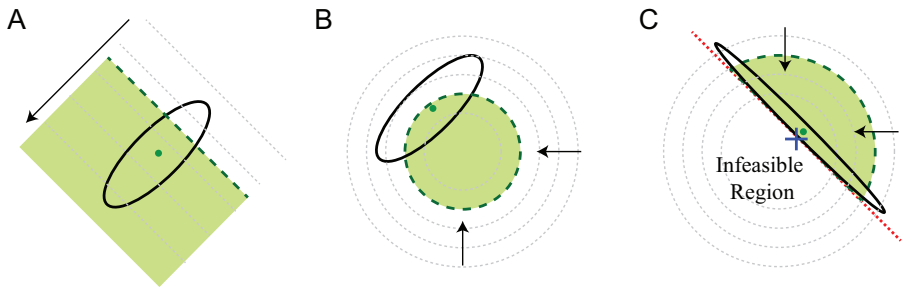


Figure 3.2: Possible scenarios encountered during a search. A) The covariance matrix (ellipsoid in solid line) is well adapted with respect to a boundary (dashed line). The probability of generating a successful solution in the viability region (shaded area) is greater than 50%. Isocline of the objective function are shown as thin dotted lines and the gradient direction is shown by the arrow. The mean of the search distribution is represented as a dot. B) The covariance matrix should be adapted. Probability of generating successful solutions is lower than 50%. C) The method encounters difficulties when the direction to reach the optimum (shown as a cross) is the same that generates infeasible solutions that violate the constraint (thick dotted line).

must only provide an initial step size σ. The algorithm sets the initial viability boundaries b as either the target boundary (0 for the constraints) or a relaxed value if an infeasible solution is provided. The initial boundary for the objective is set to ∞. At each iteration, boundary violations V are checked. The active covariance matrix update for feasible solutions (Equation 3.4) and the stall of updates of the original method in presence of high probability of success are not used. A single update rule is applied whenever a viable solution (that does not violates the boundaries b) is generated. When this happens, the mean of the population is updated to the new viable solution and the boundaries are tightened.

3.5 Results

The proposed method was tested on all the eight benchmark functions used in (Arnold and Hansen, 2012). These benchmark functions include problems from two to ten dimensions with up to eight non-linear constraints. The experimental setup is identical to the one reported in (Arnold and Hansen, 2012), including the same number of repetitions, equivalent generation of initial solutions, the same termination condition, and the same parameter settings for the (1+1)-CMA-ES (reported in Table 3.1). For each benchmark function we counted the total number of objective function and constraints function evaluations. We tested the method starting it 99 times from different initial solutions, uniformly sampled from the solution space until a feasible solution is found. Iterations needed to obtain the starting feasible solution are not counted in the results, as in (Arnold and Hansen, 2012).

Results are reported in Table 3.2. The method is competitive on seven out of eight problem. Our method has medians lower than what were reported by Arnold and Hansen (Arnold and Hansen, 2012) for constraint function calls by a factor of 0.15, 0.33, 0.11, 0.56, 0.49, 0.57 on g06,

	g06		g07		g09		g10	
	(1+1)-ViE	acCMA	(1+1)-ViE	acCMA	(1+1)-ViE	acCMA	(1+1)-ViE	acCMA
Function Evaluations								
10th	282	**272**	**1578**	1939	**1305**	1430	**1387**	2794
50th	333	**308**	**1794**	2211	**1452**	1674	**1697**	3976
90th	385	**364**	**2049**	2703	**1595**	2074	**2554**	5369
Constraint Evaluations								
10th	**797**	827	**7184**	10435	**3474**	3626	**7360**	15621
50th	**900**	1060	**7545**	11283	**3660**	4106	**8295**	18781
90th	**986**	1223	**8032**	12704	**3913**	5075	**11322**	23088

	TR2		2.40		2.41		HB	
	(1+1)-ViE	acCMA	(1+1)-ViE	acCMA	(1+1)-ViE	acCMA	(1+1)-ViE	acCMA
Function Evaluations								
10th	465	**376**	**863**	1326	**820**	1483	638	**623**
50th	520	**443**	**1023**	1990	**954**	2271	**734**	768
90th	561	**510**	**1209**	3326	**1100**	3581	**841**	1150
Constraint Evaluations								
10th	751	**616**	**3166**	4551	**3183**	5235	2659	**2338**
50th	812	**708**	**3570**	6994	**3449**	8108	**2893**	2912
90th	884	**839**	**3899**	11114	**3801**	12056	**3185**	3970

Table 3.2: Experimental results of the (1+1)-ViE and comparison against the (1+1)-acCMA-ES proposed in (Arnold and Hansen, 2012)

g07, g09, g10, 2.40, 2.41 respectively and almost identical performance on HB [3]. In the linear constrained sphere function problem TR2, our method exceeds values reported by Arnold and Hansen (Arnold and Hansen, 2012) by a factor 0.15. In one problem, g06, our method, while being better on the overall number of constraint evaluations, performs slightly worse on number of objective function evaluations.

In our view, one of the reasons of decreased performance in the TR2 problem probably lies in the specific orientation of the constraint. From experimental investigation, we observed that the mean of the search distribution tends to align to the normal direction to the optimum (a situation similar to the one depicted in Figure 3.2C), which in this case is also the same direction that is most likely to violate the constraint. Probably, in cases like this one when the direction of constraint violation is very close to the direction of viable solutions generation, the covariance matrix update should be stalled, or the variance should be decreased along the other axis.

[3] It must be noted that although the reported function evaluations are consistently lower than the method of Arnold (compare 10th, 90th percentiles and medians), the extent of the analysis performed here does not allow to draw conclusions on the statistical significance of the results, as no hypothesis testing was performed.

3.6 Conclusions

In this chapter, we proposed (1+1)-ViE, a method that combines viability boundaries and active covariance matrix updates in a (1+1)-CMA-ES. Our algorithm showed competitive performance with respect to state-of-the-art methods on all the benchmark problems except on the constrained sphere function problem TR2. Further investigations are needed to solve the lower performance experienced on TR2. Here, we tested the method only when starting from feasible initial solutions, but our algorithm is also compatible with infeasible starting solutions. In the next chapter, we extend this method to deal with multi-modal constrained problems and we test it also when started from infeasible starting solutions.

4 Constrained multimodal optimization using viability evolution principles

In the previous chapter we introduced a viability-based algorithm that displayed promising performance on a set of unimodal problems. In this chapter, we devise a novel memetic computing approach, based on previous chapter's algorithm, for solving constrained optimization problems with inequality constraints. Our framework embeds several design choices from the Viability Evolution paradigm. Viability boundaries are adapted during the search to drive an evolving population of local search units towards desired regions of search space. The local units can be recombined by means of global Differential Evolution operators. An adaptive scheduler toggles between the exploitative and explorative regimes, selecting at each iteration wether to advance one of the local optimizers, apply the evolutionary operators, or do both. The proposed algorithm, named *mViE*, was tested on a diverse set of benchmark functions and engineering problems. mViE outperforms several state-of-the-art methods in terms of quality of solutions and computational resources needed.

The contents of this chapter are adapted from:

Maesani A, Iacca G, Floreano D (2014). Adopting Viability Evolution Principles in a Memetic Framework for Constrained Optimization. Submitted to IEEE Transactions on Evolutionary Computation, in review.

4.1 Introduction

Several real-world optimization problems are characterized by the presence of one or more inequality constraints that limit the feasible region of the search space. When defined in a continuous domain, such problems can be formulated as:

$$\min f(\boldsymbol{x}),\ \text{s.t.}: \begin{cases} l_i \leq x_i \leq u_i, & i = 1,2,\ldots,n \\ g_j(\boldsymbol{x}) \leq 0, & j = 1,2,\ldots,m \end{cases} \tag{4.1}$$

where $f(\boldsymbol{x})$ is the objective (fitness) function to be optimized and $\boldsymbol{x} \in \mathbb{R}^n$ is a vector of design variables $[x_1, x_2, \ldots, x_n]$.

The search space is delimited by box constraints that define the admissible range $[l_i, u_i], l_i \in \mathbb{R}, u_i \in \mathbb{R}$ of each variable x_i. Moreover, the feasible space is determined by inequality constraints $g_j(\boldsymbol{x})$[1]. Examples of constrained optimization problems can be found in many fields where physical, geometrical or resource requirements may limit the feasibility of the solutions.

Due to the vast range of applications, constrained optimization has attracted over the past few decades the interest of a large part of the Computational Intelligence research community[2]. Pioneering studies can be traced back to research on Genetic Algorithms (GA) (Michalewicz et al., 1996; Rasheed, 1998; Coello Coello, 2000; Coello Coello and Mezura-Montes, 2002) and Evolution Strategies (ES) (Mezura-Montes and Coello Coello, 2005). More recently, Particle Swarm Optimization (PSO) has also attracted a growing interest (Hu et al., 2003; Aguirre et al., 2007; He and Wang, 2007a; Cagnina et al., 2008; Pant et al., 2009; Coelho, 2010), together with other variously inspired Swarm Intelligence techniques (Leguizamón and Coello Coello, 2009; Yang and Deb, 2010; Brajevic et al., 2011; Cuevas and Cienfuegos, 2014; Zhang et al., 2014). Increasingly sophisticated techniques for managing the number of feasible or unfeasible individuals retained in the population have been investigated in (Cai and Wang, 2006) and (Wang et al., 2008). Parallel research lines have explored the use of specific constraint handling techniques (Michalewicz, 1995; Coello Coello, 2002; Mezura-Montes and Coello Coello, 2011), adaptive penalty functions (Tessema and Yen, 2009; Kramer et al., 2013), repair mechanisms for infeasible solutions (Salcedo-Sanz, 2009; Wessing, 2013), stochastic ranking of solutions (Runarsson and Yao, 2000), ε-constrained optimization (Takahama and Sakai, 2010), feasibility rules to rank solutions (Deb, 2000), and surrogate models (Jin, 2011). Recent empirical studies analyzed the utility of retaining infeasible solutions during evolution (While and Hingston, 2013), whereas others have considered the use of multi-objective techniques where the constraint violations are minimized as separate objectives together with the problem objective function (Angantyr et al., 2003; Mezura-Montes and Coello, 2008; Clevenger et al., 2005; Cai and Wang, 2006). Competitions for constrained optimization, as those organized in the context

[1] Constrained optimization problems can also include equality constraints defined as $h_k(\boldsymbol{x}) = 0, k = 1,2,\ldots,p$, that can reduce the feasible areas down to zero-volume regions. Dedicated literature covers this class of problems proposing specific techniques for handling them. Handling equality constraints is out of the scope of this chapter.

[2] A constantly updated list of references on the topic, maintained by Carlos A. Coello Coello, is available at: http://www.cs.cinvestav.mx/~constraint/

of the IEEE Congress on Evolutionary Computation (CEC) (Liang et al., 2006; R. Mallipeddi, 2010), have finally provided standard benchmarks for comparing the performance of the various algorithms in the field.

Despite these advances, the increasing number of computationally-intensive applications still needs to be matched by computationally-efficient algorithms, capable of delivering high performance in terms of quality of discovered solutions using a limited amount of function evaluations. In contexts where each evaluation is computationally expensive, as in the case of complex simulations, or where the optimization process has to be run in a limited time, as in the case of hardware-in-the-loop evolution, there is indeed a strong need for efficient meta-heuristics for constrained optimization.

Here, we describe a novel memetic computing approach based on Viability Evolution principles (Mattiussi and Floreano, 2003; Maesani et al., 2014), an alternative abstraction of artificial evolution that operates by eliminating individuals not satisfying a set of criteria, defined on problem objectives or constraints, which are adapted during evolution. We call our proposed method *memetic Viability Evolution* (mViE). The algorithm is composed of multiple local search units constituted by (1+1) Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Igel et al., 2006), coordinated by a Differential Evolution (DE) (Storn and Price, 1997) global search kernel. The local search units are based on our previous algorithm (Maesani and Floreano, 2014), where we modified a variant of a (1+1)-CMA-ES proposed in (Arnold and Hansen, 2012) by adding Viability Evolution principles (Mattiussi and Floreano, 2003; Maesani et al., 2014) to dynamically relax the constraints and drive the local search units towards feasible areas of the search space; furthermore, we have introduced a mechanism to adapt the step size based on information collected at each constraint violation. Here, in mViE, on top of these independent local search units, locally-learned information is recombined in a memetic fashion by applying the evolutionary operators typical of DE for global search. The efficiency of our method is made possible thanks to an adaptive scheduler that selects the advancement of local search units versus global search steps.

We tested mViE on thirteen on the CEC 2006 benchmark problems with inequality constraints (Liang et al., 2006) as well as four classical mechanical engineering design problems(Coello Coello, 2000; Gandomi and Yang, 2011). mViE was compared against an extensive collection of state-of-the-art algorithms for constrained optimization. Our method shows consistent performance gains in terms of function evaluations needed to reach the optima on almost all tested CEC 2006 problems and three mechanical engineering design problems.

The chapter is organized as follows. Section 4.2 surveys the literature related to constrained optimization and presents viability evolution principles. Section 4.3 describes the details of the proposed method. The experimental setup and the algorithmic parameter setting are discussed in section 4.4, while the numerical results are presented in section 4.5. Finally, discussion and conclusions are presented in section 4.6.

4.2 Related Work

In the following, we start with a literature review on methods based on CMA-ES and DE for constrained optimization, as these are the two building blocks of mViE for local and global search, respectively. We also recapitulate recently proposed memetic computing approaches. We then provide a brief introduction to Viability Evolution principles.

4.2.1 CMA-ES-based methods

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen et al., 2003) is considered nowadays the state-of-the-art in unconstrained single-objective optimization. In the presence of constraints, however, the step-size control used by CMA-ES to refine the search does not work properly (Beyer and Finck, 2012), an effect known also in standard ES. To overcome this difficulty, research has been recently devoted to improve the effectiveness of CMA-ES in constrained optimization problems. The use of adaptive penalty functions was investigated in (Collange et al., 2010), where the weight of each constraint in the penalty function was modified according to the number of iterations during which that constraint was violated. Another penalty function has been proposed in (de Melo and Iacca, 2014), where the constraint violation of all the solutions in the population is used to scale the relative violation of each solution. CMA-ES has been also integrated (Kusakci and Can, 2013a) with ASCHEA, an approach proposed in (Hamida and Schoenauer, 2002) to adapt the tolerances on the equality constraints. Other approaches considered to rank individuals based on three independent rankings (Kusakci and Can, 2013b), namely objective function, constraint violation, and number of violated constraints, or the use of surrogate models to learn information about constraints (Kramer et al., 2009; Gieseke and Kramer, 2013). A repair mechanisms was used in a problem-specific variant of CMA-ES for financial portfolio optimization (Beyer and Finck, 2012).

Among the most effective approaches to date for CMA-ES, (Arnold and Hansen, 2012) proposed a modification of (1+1)-CMA-ES (Igel et al., 2006) specifically designed for unimodal constrained problems. Starting from a feasible solution, the algorithm maintains a low-pass filtered vector representing the direction of violations of each constraint and consequently use this information to reduce the variance of the search ellipsoid along the detected direction of violation. In our previous study (Maesani and Floreano, 2014) we improved the performance of this (1+1)-CMA-ES scheme even further, by collecting information on single constraint violations and using it to adapt the step-size.

4.2.2 Differential Evolution-based methods

Earlier research on DE considered the use of feasibility rules and diversity preservation mechanisms (Mezura-Montes et al., 2004; Mezura-Montes et al., 2006) and the incorporation of domain knowledge (Becerra and Coello Coello, 2006) for constrained optimization. With

the introduction of the CEC benchmarks on constrained optimization (Liang et al., 2006; R. Mallipeddi, 2010), that created a common environment for assessing the performance of novel constrained optimization algorithms, DE became a popular choice for solving constrained problems. A number of DE-based methods proposed for solving these benchmarks are now considered the state-of-the-art in evolutionary constrained optimization. Among these methods, ε-DE (Takahama and Sakai, 2006) ranks solutions that are feasible or violate at most by an ε-value the constraints by objective value, and prefers them over unfeasible solutions that are compared by the amount of constraint violation. Notably, ε-DE won both the CEC 2006 and 2010 competitions for constrained optimization. Competitive results were also obtained by two other variants of DE, namely MDE (Mezura-Montes et al., 2006), which uses an *ad-hoc* mutation that incorporates both information from best and parent individual, and SADE (Huang et al., 2006), a self-adaptive version of DE. Both algorithms employ the three feasibility rules presented in (Deb, 2000) for handling constraints. In a comparative study (Mezura-Montes and Lopez-Ramirez, 2007), DE was shown to outperform PSO, GA, and ES also on various numerical and engineering problems. Later research on optimal DE parameter control (Mezura-Montes and Palomeque-Ortiz, 2009; Mezura-Montes et al., 2010) lead to devise robust self-adapting schemes, such as those presented in (Mezura-Montes and Palomeque-Ortiz, 2009) and (Zou et al., 2011). Furthermore, specific mutation operators were introduced in (Mohamed and Sabry, 2012; Kong et al., 2013). Also, DE was combined with (adaptive) penalty functions in (de Melo and Carosio, 2012; Ali and Zhu, 2013), Lagrangian methods for handling equality constraints in (Long et al., 2013), or even ensembles of constraint handling techniques (Mallipeddi and Suganthan, 2010). More recently, ε-DE was further extended introducing ranking (Takahama and Sakai, 2012) and surrogate models by using kernel regression (Takahama and Sakai, 2013).

4.2.3 Memetic Computing approaches

Lastly, it is worth mentioning recent research on constrained optimization by Memetic Computing approaches, i.e. schemes composed of multiple interacting search operators, or "memes", whose dynamics is inspired by the diffusion of ideas (Neri et al., 2011). A typical example are Memetic Algorithms, that combine an Evolutionary Algorithm with one or more units of individual learning. For example, in the agent-based memetic algorithm (Ullah et al., 2007; Ullah et al., 2008; Ullah et al., 2009b; Ullah et al., 2009a; Barkat Ullah et al., 2011) a society of agents co-evolves and shares individual information for solving constrained optimization problems. Another co-evolutionary agent-based algorithm has been presented in (Pescador Rojas and Coello Coello, 2012). Similar ideas can be found in some hybrid algorithms combining, for instance, GA and Artificial Immune System (AIS) (Bernardino et al., 2007), PSO and Simulated Annealing (SA) (He and Wang, 2007b), PSO and GA (Takahama et al., 2005), PSO and DE (Liu et al., 2010). In two $(\mu+\lambda)$-DE approaches (Jia et al., 2013; Wang and Cai, 2011), multiple mutation operators are applied while the algorithms modify at run-time the policy used for ranking solutions, according to the composition of the population in terms of feasible/infeasible solutions.

Other powerful memetic algorithms make use of gradient-based information, either on the fitness or on the constraints, thus making an implicit assumption of continuous and differentiable functions. An example of such methods is given in (Sun and Garibaldi, 2010), where an Estimation of Distribution Algorithm (EDA) is combined with a classic gradient-based local optimizer. Similarly, in (Hamza et al., 2014) a consensus-based GA is combined with sequential quadratic programming, but in this case the algorithm uses gradient information on the constraints. More recently, a hybrid algorithm combining PSO, DE, CMA-ES, gradient-based mutation and constraint handling with ε-level comparison was presented in (Bonyadi et al., 2013).

4.2.4 Viability Evolution

Viability Evolution (Mattiussi and Floreano, 2003; Maesani et al., 2014) abstracts artificial evolutionary processes as a set of individuals, or candidate solutions, that must satisfy a number of viability criteria for surviving ever-changing environmental conditions. Viability criteria are defined as ranges of admissible values on problem objectives and constraints, the so-called viability boundaries. These boundaries, representing environmental conditions, are adapted during the evolutionary process to drive the evolving population towards desired regions of search space. At the beginning of the search the boundaries are relaxed to encompass all randomly generated initial solutions. Then, the boundaries are gradually tightened. Once viability boundaries reach the desired target, boundaries are not tightened further, and the evolutionary process is considered complete.

Although in its simplest implementation the Viability Evolution paradigm operates by eliminating non-viable individuals, the abstraction is fully compatible with classical competition-based evolutionary algorithms. Elimination by viability and competition by objective functions can be modelled at the same time in the viability framework to determine the fitness of individuals. However, it is important to note that here fitness is intended as *a-posteriori* reproduction capability and not as an *a-priori* measurable function as currently implemented in the evolutionary computation practice.

Interestingly, some of the methods for constrained optimization, such as ε-DE(Takahama and Sakai, 2006) and ASCHEA(Hamida and Schoenauer, 2002), can be loosely seen under the Viability Evolution abstraction. For example, ε-DE defines a tolerance ε for comparing individuals by objective or constraints. Somehow, such a threshold could be seen as a viability boundary defined on the constraint violation. However, to be fully compatible with the viability paradigm, ε-DE should discard solutions violating the ε tolerance on the constraint violations. Similarly, ASCHEA dynamically changes a tolerance on equality constraints for driving solutions towards the feasibility region determined by the equality constraint. In a similar way, the constraint adaptation approach (Storn, 1999) progressively shrinks the feasible region during the evolutionary process.

4.3 Memetic Viability Evolution (mViE)

The proposed method mViE is based on two key components that act at different levels (local and global search) to perform efficient constrained optimization. To search locally, we use multiple units that independently explore the constrained landscape, as shown in Fig. 4.2a. Objective and constraints are modelled as viability criteria under the viability paradigm (Maesani et al., 2014). Viability boundaries, i.e. admissible ranges on viability criteria, are relaxed at the beginning of the evolutionary process and tightened during the search as proposed in (Maesani and Floreano, 2014) for each of the local search units. The local search units are implemented as (1+1)-CMA-ES, which are modified to learn information about the constraints and use such information to adapt the covariance matrix, as proposed in (Arnold and Hansen, 2012), and the step size, as in our previous work (Maesani and Floreano, 2014).

When local search units sample solutions that violate viability boundaries, the search unit learns the direction of violation of the boundary (Arnold and Hansen, 2012) and adapts covariance matrix and step size, as shown in Fig. 4.2b. Viability boundaries are adjusted during the search towards the actual constraints, thus activating covariance matrix and step size adaptation and pushing the local search towards feasible areas (Fig. 4.2c).

To foster global search, we combine local search units by using operators inherited from Differential Evolution, comprising rand/1 mutation and exponential crossover (Price et al., 2005). In this way, the information obtained by multiple local search units is combined to generate new units. Search parameters are inherited from the closest of the search units, as can be seen in Fig. 4.2d. Finally, an adaptive scheduler chooses the allocation of function evaluations to local search units or global search operators (Fig. 4.2e).

In the following sections we provide a detailed description of the method's operations described in Figure 4.2. The mapping of these operations with relevant algorithmic pseudo-code presented in the chapter is show in Figure 4.1. The main loop of our method is shown in Algorithm 3. The parameters of the various local search units are stored in the pop data structure appositely initialized (INITIALIZEUNITS, routine not reported, performs memory allocation) to contain data for pop_{size} units. For each local search unit, the corresponding mean $\boldsymbol{x}$ is sampled randomly (SAMPLERANDOMSOLUTION, routine not reported, performs uniform random sampling of a solution within the search space)

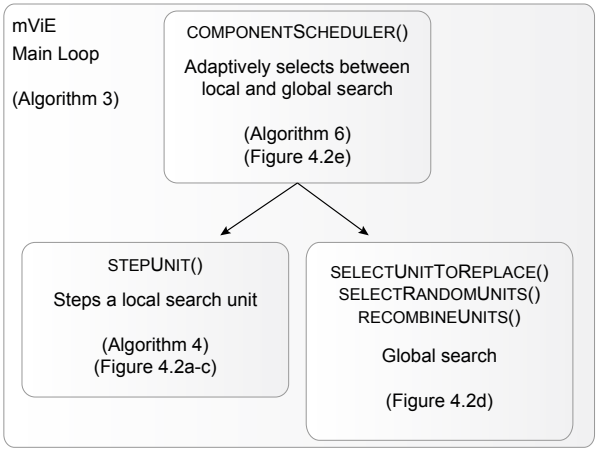


Figure 4.1: Schematic of mViE on and references to relevant figures and algorithms.

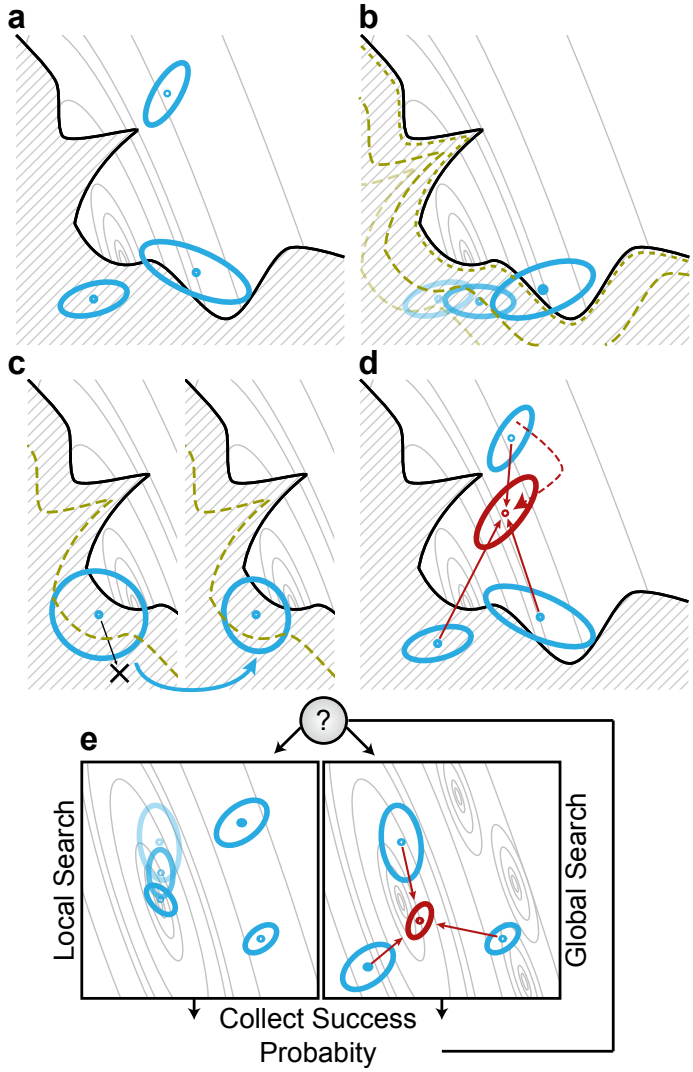


Figure 4.2: Graphical representation of the main features of mViE on a simplified two-dimensional search landscape. Non-linear constraints on the search domain are represented as solid black lines, which outline the boundaries of the infeasible area marked with oblique gray hatching. Objective function contour lines are represented in the feasible areas as thin gray lines. Panels **a**, **b** and **c** represents the local search component of mViE, while panel **d** the global search one. **a**) Multiple search units (cyan ellipsoids) sample solutions from different areas of the search landscape. A single search distribution is represented as an ellipsoid with its mean shown as a dot. **b**) Constraints can be described as viability boundaries (olive green dashed lines) and can be relaxed. By progressively tightening the boundaries, while adapting the search distribution to the viability boundary violations, it is possible to drive the search units within the feasible areas. **c**) When a search distribution samples a solution which violates the viability boundaries (left), the most-likely direction of boundary violation is learned and used to adapt the search distribution to reduce the likelihood of searching in that direction (right). **d**) Global search component. A new local unit (red ellipsoid) is created by applying Differential Evolution mutation and crossover operators (solid red arrows) on multiple search distributions. The parameters of the new distribution are inherited from the closest of the recombined search units (dashed red arrow). **e**) The probability of improving solutions generated by local or global search operators are collected and used to adapt the activation frequencies of global or local search, either stepping a local search unit (left) or recombining information using a global search operator (right).

and its parameters are initialized (INITUNIT, explained in detail in section 4.3.1 and shown in Algorithm 5).

Algorithm 3 Main loop

Require: T_{evals}: maximum number of evaluation
Require: pop_{size}: population size

while ¬ TERMINATIONCONDITION() **do**
  $P_{succ,local} \leftarrow 0.5, P_{succ,global} \leftarrow 0.5$
  $C_{active,local} \leftarrow pop_{size}$
  $pop \leftarrow$ INITIALIZEUNITS(pop_{size})
  for $i = 1 : pop_{size}$ **do** ▷ Sample initial population
    $\boldsymbol{x} \leftarrow$ SAMPLERANDOMSOLUTION()
    $pop_i \leftarrow$ INITUNIT($\boldsymbol{x}$)
  end for
  $N_{evals,local} \leftarrow 0, N_{evals,global} \leftarrow 0$
  $N_{succ,local} \leftarrow 0, N_{succ,global} \leftarrow 0$
  while ¬ RESTARTORTERMCONDITION() **do**
    $[E_{local}, E_{global}] \leftarrow$COMPONENTSCHEDULER() ▷ Choose component(s) to execute
    if E_{local} **then** ▷ Perform local search
      $N_{eval,local} \leftarrow N_{eval,local} + 1$
      $R \leftarrow$ RANKBYFITNESS(pop)
      if $m > 0$ **then**
        $R \leftarrow R +$ RANKBYVIOLATION(pop)
      end if
      $i \leftarrow$ GETACTIVEUNIT(pop, R)
      $pop_i \leftarrow$ STEPUNIT(pop_i)
      if BETTER(pop_i, $best$) **then**
        $best \leftarrow pop_i$
        $P_{succ,local} \leftarrow (1-\alpha) \cdot P_{succ,local} + \alpha$
        $N_{succ,local} \leftarrow N_{succ,local} + 1$
      else
        if ¬ VIOLATEBOUNDARIES(pop_i) **then**
          $P_{succ,local} \leftarrow (1-\alpha) \cdot P_{succ,local}$
        else
          $P_{succ,local} \leftarrow (1-\beta) \cdot P_{succ,local}$
        end if
      end if
    end if

```
33:       if E_global then                                   ▷ Perform global search
34:           N_eval,global ← N_eval,global + 1
35:           r ← SELECTUNITTOREPLACE(pop)
36:           [i_1, i_2, i_3] ← SELECTRANDOMUNITS(pop, r)
37:           trial ← RECOMBINEUNITS(pop, i_1, i_2, i_3, r)
38:           if BETTER(pop_r, trial) then
39:               pop_r ← trial
40:               if BETTER(pop_r, best) then
41:                   best ← pop_r
42:                   P_succ,global ← (1 − α) · P_succ,global + α
43:                   P_succ,local ← P_succ,global
44:                   N_succ,global ← N_succ,global + 1
45:               else
46:                   P_succ,global ← (1 − α) · P_succ,global + β
47:               end if
48:           else
49:               P_succ,global ← (1 − α) · P_succ,global
50:           end if
51:       end if
52:   end while
53: end while
```

Until a restart or termination condition is met (RESTARTORTERMCONDITION, see section 4.3.4), the scheduler is run to decide if allocating or not a function evaluation to global or local search (COMPONENTSCHEDULER, explained in section 4.3.3 and shown in Figure 4.2e, Algorithm 6).

4.3.1 Local search step

If a local search step is executed ($E_{local} = true$), the most promising unit is selected by ranking local search units by fitness and constraints violation (RANKBYFITNES, RANKBYVIOLATION, routines not reported) and selecting the first active (non-converged) top-ranking search unit (GETACTIVEUNIT, routine not reported). The local search unit is then allowed to perform a single function evaluation (STEPUNIT, shown in Figure 4.2a-c and Algorithm 4). Finally, the moving average $P_{succ,local}$ on the success probability of improving the global optima is updated. To compare solutions with the current global best solution found we employ the three feasibility rules presented in (Deb, 2000) and implemented in the function BETTER.

Local search units (Figure 4.2a-c) are basically (1+1)-CMA-ES with the modifications for adapting the covariance matrix presented in (Arnold and Hansen, 2012). Each (1+1)-CMA-ES maintains its viability boundaries and adapts them every time a better solution is sampled. The boundary on the objective is updated only when operating in the feasible area. Additional information on the probability of generating solutions that satisfy each constraints is maintained by each local unit and used for updating the step size (Maesani and Floreano, 2014). At the end of the execution of a local search unit, the unit is checked for convergence, as explained later in Section 4.3.4.

Algorithm 4 Local search stepping

function StepUnit(ls)
  $\boldsymbol{z} \sim \mathcal{N}(0, ls.\mathbf{I})$ ▷ Sample new solution
  $ls.\boldsymbol{y} \leftarrow ls.\boldsymbol{x} + ls.\sigma \cdot ls.\boldsymbol{A} \cdot \boldsymbol{z}$

  $\boldsymbol{V} \leftarrow [\mathbb{1}_{g_1(\boldsymbol{y})>ls.b_1}, \ldots, \mathbb{1}_{g_m(\boldsymbol{y})>ls.b_m}, \mathbb{1}_{f(\boldsymbol{y})>ls.b_{m+1}}]$ ▷ Check current boundary violations
  $\boldsymbol{V}_T \leftarrow [\mathbb{1}_{g_1(\boldsymbol{y})>B_1}, \ldots, \mathbb{1}_{g_m(\boldsymbol{y})>B_m}, \mathbb{1}_{f(\boldsymbol{y})>B_{m+1}}]$ ▷ Check target boundary violations

  if $\exists\, i : \boldsymbol{V}_i = 1$ **then** ▷ Update search distribution on boundary violations

    for all $i : \boldsymbol{V}_i = 1$ **do** ▷ Adapt algorithm parameters
      $ls.\boldsymbol{v}_i \leftarrow (1 - c_c) \cdot ls.\boldsymbol{v}_i + c_c \cdot ls.\boldsymbol{A} \cdot \boldsymbol{z}$
      $ls.\boldsymbol{w}_i \leftarrow ls.\boldsymbol{A}^{-1} \cdot ls.\boldsymbol{v}_i$
    end for
    $ls.\boldsymbol{A} \leftarrow ls.\boldsymbol{A} - B \sum_{i=1}^{m} \mathbb{1}_{g_i(\boldsymbol{y})>0} \frac{ls.\boldsymbol{v}_i \cdot ls.\boldsymbol{w}_i^T}{ls.\boldsymbol{w}_i \cdot ls.\boldsymbol{w}_i^T}$
    $ls.\boldsymbol{p} \leftarrow (1 - c_p) \cdot ls.\boldsymbol{p} + c_p [\mathbb{1}_{\boldsymbol{V}_1=0}, \ldots, \mathbb{1}_{\boldsymbol{V}_{m+1}=0}]$ ▷ Boundary satisfaction probability
    if $\exists\, i : ls.\boldsymbol{p}_i < \frac{1}{2}$ **then**
      $ls.P_{succ} \leftarrow (1 - c_p) \cdot ls.P_{succ}$
    end if
    if $\forall\, i : \boldsymbol{V}_{Ti} = 0$ **then** ▷ only when target boundaries are satisfied
      $ls.\sigma \leftarrow ls.\sigma \cdot exp\left(\frac{1}{d} \frac{ls.P_{succ} - P_{target}}{1 - P_{target}}\right)$
    end if

  else

    $ls.P_{succ} \leftarrow (1 - c_p) \cdot ls.P_{succ} + c_p$ ▷ Increase probability of success
    $ls.\boldsymbol{p} \leftarrow (1 - c_p) \cdot \boldsymbol{p} + c_p$
    $ls.\sigma \leftarrow ls.\sigma \cdot exp\left(\frac{1}{d} \frac{ls.P_{succ} - P_{target}}{1 - P_{target}}\right)$ ▷ Adapt search distribution parameters
    $ls.\boldsymbol{s} \leftarrow (1 - c) \cdot ls.\boldsymbol{s} + \sqrt{c(2 - c)} \cdot ls.\boldsymbol{A} \cdot \boldsymbol{z}$
    $ls.\boldsymbol{w} \leftarrow ls.\boldsymbol{A}^{-1} \cdot ls.\boldsymbol{s}$
    $ls.\boldsymbol{A} \leftarrow \sqrt{1 - c_{cov}^{+}} \cdot ls.\boldsymbol{A} +$
    $\frac{\sqrt{1 - c_{cov}^{+}}}{\|ls.\boldsymbol{w}\|^2} \left(\sqrt{1 + \frac{c_{ccov}^{+}}{1 - c_{cov}^{+}} \|ls.\boldsymbol{w}\|^2} - 1\right) ls.\boldsymbol{s} \cdot ls.\boldsymbol{w}^T$
    $ls.\boldsymbol{b}_{1..m} \leftarrow \Big[$ ▷ Adapt boundaries
    $max\Big(B_1, min\Big(ls.b_1, g_1(\boldsymbol{y}) + \frac{ls.b_1 - g_1(ls.\boldsymbol{y})}{2}\Big)\Big), \ldots$
    $max\Big(B_m, min\Big(ls.b_m, g_m(\boldsymbol{y}) + \frac{ls.b_m - g_m(ls.\boldsymbol{y})}{2}\Big)\Big)\Big]$
    if $\forall\, i : 1, \ldots, m V_{Ti} = 0$ **then** ▷ Update boundary on objective only in feasible area
      $ls.\boldsymbol{b}_{m+1} \leftarrow f(ls.\boldsymbol{y}) + \frac{f(ls.\boldsymbol{x}) - f(ls.\boldsymbol{y})}{2}$
    end if
    $ls.\boldsymbol{x} \leftarrow ls.\boldsymbol{y}$ ▷ Update mean and check for local convergence

  end if

  $ls.active \leftarrow$ CheckConvergence(ls)
  return ls
end function

The pseudo codes for initialization and stepping of local search are listed in Algorithms 5 and 4. Each unit stores information on the viability boundaries in the $\boldsymbol{b}$ vector for each unit. This is initialized relaxing the bondaries to either the target values $B_1, \ldots, B_m$ or the sampled solution constraint value for each constraint. Note that for the problems selected in this chapter, $B_i = 0, i = 1, \ldots, m$. Information on constraint violations is saved in the $\boldsymbol{v}$ and $\boldsymbol{p}$ vectors, respectively containing the running average of the violation vector for each constraint and the probability of violating each constraint.

Algorithm 5 Local search initialization

function INITUNIT($\boldsymbol{x}$)
  $ls \leftarrow [\,]$
  $ls.active \leftarrow true$
  $ls.s \leftarrow 0$
  $ls.\boldsymbol{A} \leftarrow \mathbf{I}$
  for $i = 1 \ldots m+1$ **do**
    $ls.\boldsymbol{v}_i \leftarrow [0, \ldots, 0]_{n \times 1}$ ▷ The last $\boldsymbol{v}_i$ and b_i correspond to the objective
  end for
  $ls.\boldsymbol{b} \leftarrow [max(B_1, g_1(\boldsymbol{x})), \ldots, max(B_m, g_m(\boldsymbol{x})), \infty]$
  $ls.\boldsymbol{p} \leftarrow [\frac{1}{2}, \ldots, \frac{1}{2}]$
  $ls.\boldsymbol{x} \leftarrow \boldsymbol{x}$
  return ls
end function

4.3.2 Global search step

On the other hand, when the global search step is executed ($E_{global} = true$), the search unit to replace is selected by picking two search units at random and choosing the worst one (SELECTUNITTOREPLACE, routine not reported). The selected local search unit is then replaced if a better one is generated by means of Differential Evolution rand/1 mutation and exponential crossover (SELECTRANDOMUNITS selects the parents, RECOMBINEUNITS performs mutation and crossover, both routines are not reported).

After the local or global search steps are executed, the method collects some statistics on the performance of the steps. Namely, it collects the number of function evaluations $N_{evals,\{local,global\}}$ and the number of improvements to the global optima $N_{succ,\{local,global\}}$ of each search step. Furthermore, the collection of the moving averaged probability of success of local and global search components is performed to accommodate different meaning of “success" for the local and global search steps. In particular, the local probability of success is increased according to:

$$P_{succ,local} \leftarrow (1-\alpha) \cdot P_{succ,local} + \alpha \tag{4.2}$$

when a solution better than the global best is found by a local search unit. On the contrary, the probability is decreased when a local search unit samples a solution that does not improve the

global best according to:

$$P_{succ,local} \leftarrow (1-\alpha) \cdot P_{succ,local}. \tag{4.3}$$

However, if the solution violates boundaries defined on some constraints, to account for the function evaluations needed for adapting to the current boundaries, we discount the original α coefficient by a factor β_R, replacing it with a reduced coefficient $\beta = \beta_R \cdot \alpha$.

Instead, after the global search step the probability $P_{succ,global}$ is increased in two conditions. When a solution better than the overall best solution is found, the probability is modified with a rule similar to Equation 4.2:

$$P_{succ,global} \leftarrow (1-\alpha) \cdot P_{succ,global} + \alpha. \tag{4.4}$$

The probability is also increased when a local search unit better than the parent unit is discovered, although this second increase is lower (we use the same coefficient β). In the other cases, the probability is decreased similarly to Equation 4.3:

$$P_{succ,global} \leftarrow (1-\alpha) \cdot P_{succ,global}. \tag{4.5}$$

4.3.3 Scheduler for selection of local/global search operator

The efficient allocation of function evaluations to local (Figure 4.2a-c) or global (Figure 4.2d) search is ensured by the scheduler presented in Algorithm 6 (Figure 4.2e). The scheduler uses information collected during the search process, i.e. the moving averaged probability of success $P_{succ,\{local,global\}}$, the total number of global optima improvements $N_{succ,\{local,global\}}$ and the total number of function evaluations $N_{evals,\{local,global\}}$ allocated to the local and global search component.

During the first $100 \times n$ function evaluations (empirically set during preliminary experiments) both components are always used to ensure an initial learning phase and avoid unbalancing the search towards local or global search because of initial evaluations. Then, the total probability of success $P_{Hist,\{local,global\}} = \frac{N_{succ,\{local,global\}}}{N_{evals,\{local,global\}}}$ is computed. We aggregate information about current success probability and total probability of success by multiplication. To prevent the frequency of execution of one of the two components from falling to zero, therefore disabling the component until the end of the search process, we limit the minimum frequency of execution of one component to a relative fraction L of the other component's frequency. Finally, the two components are scheduled according to the computed frequency of execution $P_{sel,\{local,global\}}$. In the case all the local search units converged, the local search step is disabled and only global search is performed.

Algorithm 6 Selection of local/global search operator

Require: $P_{curr,local}$ and $P_{curr,global}$: moving averaged probability of success of local and global search components

Require: $N_{evals,local}$ and $N_{evals,global}$: total number of function evaluations assigned to each component

Require: $N_{succ,local}$ and $N_{succ,global}$: total number of success evaluations for each component

Require: L : minimum relative limit of selection frequency

function COMPONENTSCHEDULER
  $E_{local} \leftarrow false$
  $E_{global} \leftarrow false$
  if $(N_{evals,local} + N_{evals,global}) < 100 \cdot n$ **then**
    $E_{local} \leftarrow true$
    $E_{global} \leftarrow true$
  else
    if $N_{evals,local} = 0$ **then**
      $P_{Hist,local} \leftarrow 0$
    else
      $P_{Hist,local} \leftarrow \frac{N_{succ,local}}{N_{evals,local}}$
    end if
    if $N_{evals,global} = 0$ **then**
      $P_{Hist,global} \leftarrow 0$
    else
      $P_{Hist,global} \leftarrow \frac{N_{succ,global}}{N_{evals,global}}$
    end if
    $P_1 \leftarrow P_{H,local} \cdot P_{curr,local}$
    $P_2 \leftarrow P_{H,global} \cdot P_{curr,global}$
    $P_{sel,local} \leftarrow max(P_1, L \cdot P_2)$
    $P_{sel,global} \leftarrow max(P_2, L \cdot P_1)$
    $r \leftarrow$ RAND
    if $r < \frac{P_{sel,local}}{P_{sel,global}+P_{sel,local}} \wedge C_{active,local} > 0$ **then**
      $E_{local} \leftarrow true$
    else
      $E_{global} \leftarrow true$
    end if
    return $[E_{local}, E_{global}]$
  end if
end function

4.3.4 Termination conditions

Our method uses both local convergence conditions to disable local units, and global convergence conditions to restart the algorithm. We disable a local search unit according to standard

stopping criteria for (1+1)-CMA-ES, namely when:

1. the evolution path $\boldsymbol{s}$ multiplied by the step size σ is smaller than 10^{-12}
2. the maximum element on the covariance matrix diagonal multiplied by the step size σ is larger than 10^{8}
3. the condition number of the covariance matrix is larger than 10^{14}

After a local search unit has been disabled, it can only be substituted with another active chain by the global search operator. The algorithm is restarted when the local units are converged to the same solution. This check is performed by measuring the difference between the mean of the objective and constraint violations of the local units and the best constraint violation objective of the best solution.

4.4 Experimental Setup

We selected the algorithm's main parameters by performing a preliminary parameter analysis, reported in Appendix 4.7. The identified parameter values, used in the rest of this chapter, are $\alpha = 0.1, \beta_R = 0.05, L = 0.18, F = 0.5, CR = 0.9, P_{size} = 40$. The parameters of the local search component, based on (1+1)-CMA-ES, are instead set as in (Arnold and Hansen, 2012): $d = 1+\frac{n}{2}$, $c = \frac{2}{n+2}$, $c_c = \frac{1}{n+2}$, $c_p = \frac{1}{12}$, $B = \frac{0.1}{n+2}$, $P_{target} = \frac{2}{11}$, $c^{+}_{cov} = \frac{2}{n^2+6}$, and $c^{-}_{cov} = \frac{0.4}{n^{1.6}+1}$.

To assess the performance of the method, we selected all the constrained optimization problems with inequalities[3] from the CEC 2006 benchmark(Liang et al., 2006). We executed the CEC 2006 benchmark as specified in (Liang et al., 2006) for 500.000 function evaluations on the functions containing inequalities only[4], reported in Table 4.1. For each function, we measured success rate (SR) over 25 runs and number of function evaluations (NFES) needed for solving the problem at the desired accuracy (set to 10^{-4}). Furthermore, we tested the method on a set of four engineering problems reported in Fig. 4.3. The engineering problems were run for a maximum of 200.000 function evaluations.

All the experiments were performed on Intel® machines with Core™ i7-2600 CPU @ 3.40GHz and 8GB of RAM.

[3] Although a newer CEC benchmark is available (R. Mallipeddi, 2010), with some interesting features such as scalable functions, the CEC 2006 benchmark still represents the most popular tool for testing new constrained optimization algorithms, given the availability of results for several methods from the literature

[4] Our algorithm is specifically designed for handling optimization problems with inequality constraints only. A more detailed analysis of the behaviour of mViE on problems including (also) equality constraints, together with the original MATLAB® source code of all our experiments, is available at http://lis.epfl.ch/files/content/users/195419/files/mvie/index.html.

Table 4.1: Problems with inequality constraints from the CEC 2006 competition on constrained optimization. We show the number of dimensions (n), linear (LI), non-linear (NI) and active constraints at the optimum (a).

Problem	n	LI	NI	a
g01	13	9	0	6
g02	20	0	2	1
g04	5	0	6	2
g06	2	0	2	2
g07	10	3	5	6
g08	2	0	2	0
g09	7	0	4	2
g10	8	3	3	6
g12	3	0	1	0
g16	5	4	34	4
g18	9	0	13	6
g19	15	0	5	0
g24	2	0	2	2

4.5 Results

The proposed method was first tested on problems containing only inequalities taken from the CEC 2006 benchmark (Liang et al., 2006). On all the 25 runs of every test problem, our method exhibited 100% success rate, i.e. it could reach in every run the target fitness difference from the optimum (10^{-4}). The number of function evaluations (NFES) needed for reaching success are reported in Table 4.2. Furthermore, in Appendix 4.8 we report the statistics on the error values, according to the CEC 2006 format (Liang et al., 2006).

We compared the median NFES needed to solve the CEC 2006 inequality problems against representative algorithms from the state-of-the art in constrained optimization. The main features of the compared algorithms are detailed in Table 4.3. Table 4.4 shows the comparison of median NFES to reach the optimum with several algorithms presented in the literature, grouped by underlying meta-heuristic (DE, CMA-ES, PSO, and others) for convenience. The bold face indicates the lowest NFES for each problem, and "-" indicates that the result for that problem is not available.

A closer examination at the comparison between mViE and the group of algorithms derived from DE reveals a strong performance advantage of mViE in finding optimal solutions. In only one problem out of the thirteen considered, `g12`, mViE needs about 3.18 times more function evaluations to reach the optimum. In all the other problems mViE needs between 0.21 (`g09`) and 0.80 (`g01`) evaluations (mean factor: 0.43 ± 0.18 SD) with respect to the other methods.

Comparing mViE with the algorithms derived from CMA-ES highlights an even stronger

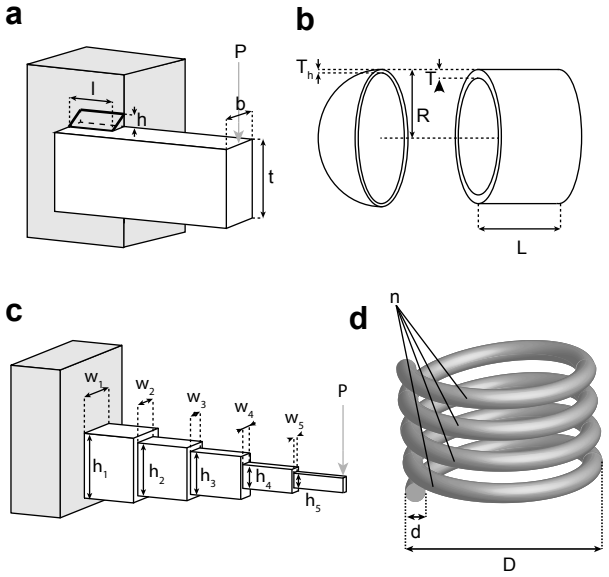


Figure 4.3: Standard engineering problem benchmarks (Coello Coello, 2000; Gandomi and Yang, 2011). **a**) Welded beam design optimization to minimize fabrication cost. The beam is fabricated out of carbon steel and welded on a rigid supporting structure. The shear force P is loading the free tip of the beam. The dimensions of the beam (width t and thickness b) and the width h and length l of the welded joint have to be optimized subject to constraints on shear stress, bending stress, buckling load on the bar, deflection and geometric constraints. **b**) Minimization of volume of a stepped cantilever. The cantilever is composed of five segments having variable cross-section, defined by the design variables w_i and h_i. The design is subject to constraints limiting the bending stress and aspect ratio of each beam segment, and the total deflection of the cantilever at the tip. **c**) Minimization of fabrication cost of a pressure vessel. The thickness of the spherical head T_h, the thickness of the spherical skin T_s, and the inner radius of the vessel have to be designed to comply with constraints derived from the ASME (American Society of Mechanical Engineers) standards on pressure vessels. **d**) Design of a tension compression spring. The weight of the spring must be minimized optimizing mean coil diameter D, wire diameter d and number of active coils n, subject to constraints on deflection, shear stress, surge frequency, and maximum size on the outside diameter.

performance gain. mViE is slower only in three problems `g06`, `g08` and `g10`. It must be noted however that for `g06` and `g10` (both unimodal problems) the fastest algorithm is (1+1)-ViE, i.e. the same employed by our local search units, presented in our previous work (Maesani and Floreano, 2014). It is therefore to be expected a somehow higher number of function evaluations, as in the method proposed here the global search component is also at work and may slow down the search for the optimum when the landscape is unimodal. On $g08$, mViE is also 1.38 times slower than APM-CMA-ES(Kusakci and Can, 2013a), but given the already low number of function evaluations needed to solve this problem the performance decay in this case is not particularly relevant.

mViE outperforms, on all problems, also all the PSO-based methods, as well as other al-

Table 4.2: Best, Median, Worst, Mean and Std. Dev. of NFES to achieve the fixed accuracy level ($(f(\vec{x}) - f(\vec{x}*)) \leq 0.0001$), and Success Rate over 25 runs of the selected CEC 2006 problems.

Prob.	Best	Median	Worst	Mean	Std	SR
g01	15032	20304	26301	20645.8	2757.39	100
g02	42462	61072	200394	67972.4	30770.3	100
g04	3089	3945	12461	4540.88	2154.32	100
g06	1072	1901	7222	2782.84	1884.95	100
g07	5954	7281	44035	10511.9	8732.84	100
g08	185	482	812	504.2	183.513	100
g09	2586	3436	26837	5193.68	5064.41	100
g10	10995	14734	99587	23884.3	22860.9	100
g12	187	3809	12607	3967.32	2371.44	100
g16	2415	3128	10214	3843.12	2100.47	100
g18	4683	7272	73350	13916.9	17785.5	100
g19	22658	25914	35753	26770.1	3029.09	100
g24	492	718	2511	838.68	400.114	100

gorithms not classifiable in any of the aforementioned groups (PCX(Sinha et al., 2006), ASRES(Runarsson, 2006), and two DE-based memetic algorithms using an adaptive policy to rank solutions depending on the population composition, namely (μ+λ)-DE (Wang and Cai, 2011) and ICDE (Jia et al., 2013)).

Finally, we compared our method with algorithms that make use of traditional non-linear programming techniques which: i) assume that the objective and constraint functions are differentiable and ii) compute and use the gradient on constraints or objectives. Although evolutionary algorithms cannot be directly compared with such techniques, as they assume that no gradient information is available and are typically derivative-free, we deemed interesting to relate our method also with this part of the literature. Results of these comparisons can be found in Table 4.6. It is remarkable that our method, despite the lack of any information about the gradient, outperforms SADE (Huang et al., 2006) and DMS-PSO (Liang and Suganthan, 2006) on all problems but two (`g12` and `g19`, respectively) and can exceed the performance of GB-MA (Sun and Garibaldi, 2010) in four problems, namely `g08`, `g10`, `g16`, and `g18`. In the other cases, using gradient information is clearly beneficial.

4.5.1 Engineering problems

Given the particularly favorable results obtained by mViE on the CEC 2006 benchmark, we decided to test the method also on a group of well-known engineering problems. Specifically, we tested mViE on a welded beam design problem, the design of a stepped cantilever, the optimization of a pressure vessel (reference formulations for these three problems can be found in (Coello Coello, 2000)) and the design of a tension compression spring (Gandomi and

Table 4.3: mViE was compared against multiple algorithms selected from the literature. We report below the main features of each algorithm, together with the method used to compare solutions and handle constraints.

Algorithm	Heuristic	Ranking method	Notes
Eps-DE (Takahama and Sakai, 2006)	DE	ε-ranking	Gradient based mutations
Eps-RDE (Takahama and Sakai, 2012)	DE	ε-ranking	Surrogate function is used in ε-ranking comparisons
MPDE (Tasgetiren and Suganthan, 2006)	DE	Penalty Function (Near Feasibility Threshold (Smith and Tate, 1993))	Use multiple sub-populations to maintain diversity
GDE (Kukkonen and Lampinen, 2006)	DE	Non dominated sorting	Problem reformulated as multi-objective problem using sum of constraint violations as objective
MDE (Mezura-Montes et al., 2006)	DE	3 feasibility rule[A]	Modified DE mutation operator that consider the best and three other randomly selected individuals. Uses a diversity procedure based on stochastic ranking with probability modified during search
JDE-2 (Brest et al., 2006)	DE	3 feasibility rule[B]	Adapts the F and CR parameters of DE during search
(1+1)-aCMA (Arnold and Hansen, 2012)	CMA-ES	If offspring violates constraints, adapt covariance matrix, otherwise substitute parent if better fitness	
(1+1)-ViE (Maesani and Floreano, 2014)	CMA-ES	As in (1+1)-aCMA	Includes a mechanism for adapt the step size based on information gathered on constraint violations. Relax constraints and uses them to drive the search towards feasible area
APM-CMA-ES (Kusakci and Can, 2013a)	CMA-ES	Penalty function (adaptive)	Adapts tolerances on equality constraints during the search
AP-CMA-ES (de Melo and Iacca, 2014)	CMA-ES	Adaptive penalty function	
PSO (Zielinski and Laur, 2006)	PSO	3 feasibility rule[A]	
COPSO (Aguirre et al., 2007)	PSO	3 feasibility rule[A]	Adapts tolerances on equality constraints during the search. Maintains an archive of solutions that are estimated to be close to constraint boundary. Applies local perturbation of best solutions found.
DMS-PSO (Liang and Suganthan, 2006)	PSO + Local Search (SQP)	Use one constraint (adaptively chosen) for each sub-population as objective	Maintains multiple sub-populations

PESO+ (Munoz-Zavala et al., 2006)	PSO	3 feasibility rule[A]	Adapts tolerances on equality constraints during the search. Mantains an archive of solutions close to constraint boundary. Applies local perturbation of best solutions found.
ASRES (Runarsson, 2006)	ES	Stochastic ranking performed using surrogate models of constraints and objectives	
(μ+λ)-CDE (Wang and Cai, 2011)	DE	Adaptive trade-off model to select offspring for next generation that depends on population composition (only feasible, only infeasible or mixed)	
ICDE (Jia et al., 2013)	DE	As in (μ+λ)-CDE	Uses an archiving strategy in the infeasible population case
PCX (Sinha et al., 2006)	GA	3 feasibility rule[C]	Adapts tolerances on equality constraints during the search. Steady state optimization procedure. PCX recombination operator.
SADE (Huang et al., 2006)	DE + SQP	3 feasibility rule[D]	Select probabilistically among various DE mutation strategies
DMS-PSO (Liang and Suganthan, 2006)	PSO + SQP	Use one constraint (adaptively chosen) for each sub-population as objective	Maintains multiple sub-populations
GB-MA (Sun and Garibaldi, 2010)	EDA + DONLP2	Over-penalize approach (Runarsson and Y., 2005): rank first feasible solutions by objective, then infeasible ones by sum of constraint violations.	

[A] the error for infeasible solutions is computed as the sum of constraint violations
[B] the error for infeasible solutions is computed as the mean of the constraint violations
[C] the error for infeasible solutions is computed as the sum of constraint violations, the fitness is rescaled according to special rules
[D] the error for infeasible solutions is computed as the mean of constraint violations, normalized by the maximum violation observed for each constraint

Yang, 2011).

For each engineering problem, we compared the median number of function evaluations needed by mViE to reach the optimum solution against the results reported in literature, as shown in Table 4.5 (also in this case the bold face indicates the lowest NFES and "-" indicates that the result is not available). We selected algorithms from the literature considering only those reporting the values for the best solution found. Furthermore, algorithms whose performance is completely dominated by other reported ones were ignored. Notably, the number of function evaluations needed to reach on average the best solution fitness value is normally not reported in papers dealing with engineering optimization. Therefore, in those cases we report the full budget of function evaluations given to the algorithm. On three out of four problems mViE is capable of discovering the best known solution in the lowest number of function evaluations. Interestingly, in one problem, namely the design of a cantilever beam, mViE discovered a solution which is better than the known optimum reported in the literature. On the other hand, on the tension spring design problem mViE converges to the optimal known solution but is 2.79 times slower than the fastest algorithm, MBA (Sadollah et al., 2013).

For completeness, we report in Table 4.7 the fitness value, the constraint violation, and the values of the design variables of the best solutions found, together with the median NFES to reach them (over 25 runs).

4.5.2 Sample algorithm runs

Overall, mViE compared very favorably on both the CEC 2006 benchmark problems and the four selected engineering problems. To provide an idea of the algorithm dynamics, we show in Fig. 4.4 the sample execution of our method on three selected problems, respectively `g01`, `g02` and `g10`, where the latter one is unimodal. The convergence plots (Fig. 4.4a) and the variables used by the scheduler (Fig. 4.4b-c) to make informed decisions on the frequency of selection of local versus global search are shown. We also display the actual probability of executing local/global search (Fig. 4.4d) and the total number of function evaluations performed by local and global search (Fig. 4.4e). It is noticeable how on the unimodal problem (`g10`) a higher number of function evaluations are allocated to the local search units. Furthermore, problems characterized by the presence of distinct local optima (`g02`) present periods in which mViE tries to locally optimize a local optimum intertwined with phases of global exploration during which the algorithm is capable to overcome local optima. The limits imposed on the frequency of selection are necessary to avoid that the algorithm falls in a phase where only global or local search are used, without the possibility to switch.

4.5.3 Performance dissection

To show the contribution of the algorithmic components (local search, global search and scheduler) we performed three separated experiments. First, in the condition mViE-L we

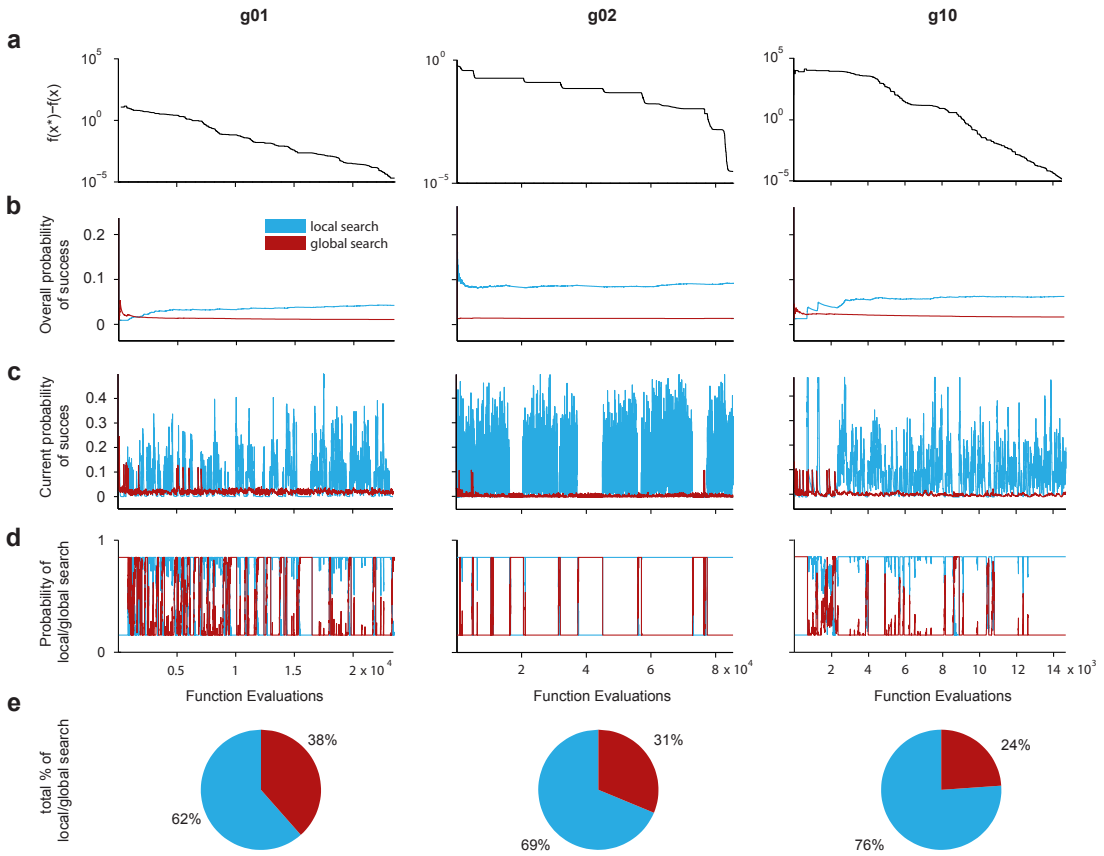


Figure 4.4: Sample execution of the method on three difficult functions taken from the CEC 2006 benchmark, namely `g01`, `g02`, and `g10`. We show a) the distance from the optima, b) the overall probability of success of local and global search, c) the moving average of the probability of success of the two components, and d) their resulting frequency of execution. Finally, panel e) shows the resulting allocation of function evaluations to local and global search.

enabled only the local search component. Multiple units (with restart) were still used, but without applying global search operators. Second, in the condition mViE-G no local search unit was stepped, and only the global search component (DE operators) was used to explore the search space. Finally, in the third condition mViE-R both local and global components were used, coordinated by a trivial scheduler that randomly chooses the execution of global or local search at each iteration. We executed the algorithms corresponding to the three conditions for 25 runs on the selected CEC 2006 benchmark problems. Results are reported in Figure 4.5, that shows the comparison of the three conditions against the fully-featured algorithm mViE.

A rather evident result is that the success rate of the local and global search component alone is greatly lower than that of the algorithms using the two components together, as expected. Furthermore, the amount of function evaluations used to discover the optima is very high

in the mViE-L condition due to the high number of restarts used by the algorithm before discovering a successful solution. Remarkably, the use of the two components allows to discover almost always the optima also using a random scheduler (mViE-R). However, the introduction of the adaptive scheduler allows a further ~ 25% improvement on the number of function evaluations used.

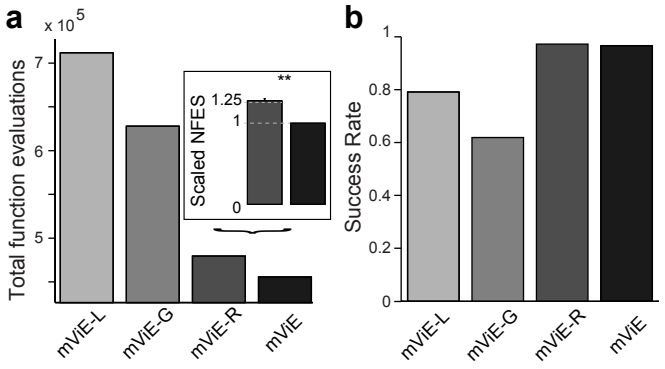


Figure 4.5: Performance dissection over 25 runs of each of the selected CEC 2006 problems for the four experimental conditions. **a**) Total number of function evaluations (aggregate over all runs and problems) to reach for each problem the target fitness difference from the optimum (10^{-4}). If the target is not reached, the maximum budget (250000 NFES) is added. The inset shows the scaled NFES, calculated as mean (over all problems) of the median NFES (calculated independently for each problem, considering only successful runs), scaled by the corresponding median NFES of mViE. mViE results ~ 25% more efficient than mViE-R (Wilcoxon Rank-Sum test, $\alpha = 0.05, p = 0.0015$). **b**) Average success rate over all runs and problems.

4.6 Discussion and Conclusions

In this chapter we presented mViE, a novel method for constrained optimization based on Viability Evolution principles. The proposed method is composed of multiple (1+1)-CMA-ES acting as local optimizers, that are combined with a Differential Evolution scheme to perform global search.

In numerical experiments, our method displayed a particularly robust performance, solving a broad number of test functions (Tables 4.2-4.4) and engineering design problems (Tables 4.5-4.7) more efficiently than the state-of-the-art compared methods, both in terms of success rate and number of function evaluations needed to reach the optimum. mViE outperformed in all the comparisons all the DE-based methods (except for one case, `g12`), as well as the CMA-ES-based algorithms (except for three cases, `g06`, `g08` and `g10`), the variants of PSO (except in one case, `g19`) and all the other algorithms.

Interestingly, the few cases in which mViE was outperformed correspond to problems where either strongly explorative algorithms (favoring global search only) or strongly exploitative

ones (favoring local search) perform well enough. For example, the comparison with the algorithm we proposed in our previous work (Maesani and Floreano, 2014) revealed that on fitness landscapes such as `g06` and `g10` a single local search unit performs more efficiently than multiple ones coordinated by DE (as in mViE). We believe that in these cases the additional overhead introduced by the initial learning phase is the cause of this performance decay. On the other hand, under the general assumption that there is no prior knowledge on the features of the landscape, a learning phase might be necessary. Also, most of the problems are actually characterized by fitness landscapes where a proper trade-off between global and local search has to be found, and these are the cases where mViE excel.

The reason for the success of mViE is twofold. First, the modelling of independent constraints as viability boundaries in the local search CMA-ES units allows to drive smoothly the search towards the feasible space and its most promising areas. As the local search units define viability boundaries separately on every constraint, they can collect additional information on each constraint. This information is therefore beneficial for a more fruitful and faster search, for example for adapting step size and covariance matrix. On the contrary, approaches that combine constraints into the objective function or as a single constraint violation measure loose a potential source of information. Second, the global recombination of multiple pieces of local information enables a synergistic exploration of the constrained search space. On top of that, a heuristic scheduler adaptively activates the local and global search, providing a proper balance between the two regimes.

Dealing with constrained optimization problems without aggregating the constraints in the fitness function or in an aggregate constraint violation function unlocks additional information that is readily available to an evolutionary algorithm. Still, the large majority of approaches in the literature make use of some form of aggregation of constraints[5].

We therefore deem useful to ask the question whether or not the current abstraction under which evolutionary algorithms operate is the most appropriate for optimization problems different from unconstrained ones. The logic behind most of the available constraint handling techniques, such as penalty functions, is in our view symptomatic of how the current evolutionary algorithms paradigm may be misleading in the design of novel algorithms. Under this traditional paradigm, algorithms are designed for having solutions in competition with each other based on their fitness function value. Solutions are therefore ranked and compared uniquely using this single value. Intuitively modelled on a very high-level abstraction of natural evolution, this paradigm may hinder the development of the field itself towards more comprehensive paradigms and thus more powerful algorithms. In fact, the current abstraction forces researchers and adopters into thinking an evolutionary process as naturally modelled using a single fitness function.

On the other hand, Viability Evolution models an evolutionary process as elimination of

[5]We also aggregate the constraints into a single constraint violation function when applying global search operators and when comparing against the global best solution in the main algorithmic loop. In a future version of mViE it may be beneficial to remove as well this form of aggregation.

solutions that do not comply with certain viability criteria, defined on both problem objectives and constraints. By adapting these criteria during the search, it is possible to drive the solutions towards desired areas of search space, for example global optima or feasible areas.

A first direct implication of the Viability Evolution paradigm is that constraints and objectives are kept implicitly separated[6]. Also, the "fitness" of individuals is in this case a property measured *a-posteriori* and not defined *a-priori* as done by using a fitness function in the classic sense. Furthermore, the availability of statistics on the viability of solutions made possible by this different abstraction, e.g. the number of individuals satisfying specific viability criteria or the number of viable/non-viable individuals, leads to increased information available for evolutionary methods. Third, more emphasis on elimination of non-viable solutions rather than competition of solutions by a unique fitness score may lead to enhanced diversity in the evolving population (Maesani et al., 2014).

Here, we showed that by following the design principles of Viability Evolution it is possible to derive a very efficient method for constrained optimization. Overall our method models constraints separately and more importantly uses information about constraint violations (non-viability of individuals) for adapting the algorithm parameters during the search, without requiring a user to aggregate them.

The present work contributes to the field of constrained optimization and suggests a wide spectrum of possible research lines that are worth following, going in the direction of: 1) extending the viability concept to different classes of problems, such as large-scale, multi-objective, and dynamic optimization; 2) testing alternative recombination schemes, based for instance of swarm intelligence, to coordinate the multiple local search operators; and 3) applying the proposed method to real-world applications where a resource-efficient constrained optimization solver might be needed, for instance in various domains of engineering or computational biology.

[6] A similar shift in paradigm was observed in multi-objective evolutionary algorithms (MOEAs), when classic aggregation methods such as weighted summing of objectives were replaced by the use of Pareto optimality concepts. Such a shift led to radically novel MOEAs, eventually obtaining dramatic performance improvements.

Table 4.4: Median NFES to achieve the fixed accuracy level ($(f(\vec{x}) - f(\vec{x}*)) \leq 0.0001$) and Success Rate for the selected CEC 2006 problems. All the compared results are obtained from the corresponding papers. Please note that BEST CEC06 aggregates the best results obtained on each problem by all the algorithms presented at CEC 2006. Results reported for PCX(Sinha et al., 2006) and Eps-RDE (Takahama and Sakai, 2012) show, respectively, 25^{th} percentile and mean, rather than median NFES.

					DE											
Prob.	**mViE**		**BEST CEC06**		**Eps-DE**		**Eps-RDE**		**MPDE**		**GDE**		**MDE**		**jDE-2**	
	NFES	SR	NFES	SR	NFES	SR	NFES	SR	NFES	SR	NFES	SR	NFES	SR	NFES	SR
g01	**20304**	100%	25115	100%	59345	100%	56508	100%	43794	100%	40200	100%	75000	100%	50354	100%
g02	**61072**	100%	96222	100%	146911	100%	99742	100%	280272	100%	106332	72%	71100	16%	138102	92%
g04	**3945**	100%	15281	100%	26098	100%	51614	100%	20823	100%	15157	100%	39300	100%	40958	100%
g06	**1901**	100%	5202	100%	7316	100%	10152	100%	10550	100%	6431	100%	5250	100%	29844	100%
g07	**7281**	100%	26578	100%	74476	100%	99830	100%	57079	100%	112969	100%	176400	100%	126637	100%
g08	**482**	100%	918	100%	1182	100%	4063	100%	1632	100%	1486	100%	900	100%	3564	100%
g09	**3436**	100%	16152	100%	23172	100%	42266	100%	20814	100%	30784	100%	15000	100%	55515	100%
g10	**14734**	100%	25520	100%	105799	100%	99820	100%	48508	100%	81827	100%	163500	100%	144247	100%
g12	3809	100%	1308	100%	4155	100%	7873	100%	4227	100%	3016	100%	**1200**	100%	6684	100%
g16	**3128**	100%	8730	100%	13001	100%	-	-	13135	100%	13307	100%	8700	100%	261549	100%
g18	**7272**	100%	28261	100%	59232	100%	-	-	42550	100%	377732	76%	118050	100%	449306	100%
g19	25914	100%	**21830**	100%	354060	100%	-	-	115054	100%	206556	88%	-	-	101076	100%
g24	**718**	100%	1794	100%	2928	100%	-	-	4371	100%	3059	100%	1650	100%	319611	100%

											PSO					
Prob.	**mViE**		**(1+1)-aCMA**		**(1+1)-VIE**		**APM-CMA-ES**		**AP-CMA-ES**		**PSO**		**COPSO**		**PESO+**	
	NFES	SR	NFES	SR	NFES	SR	NFES	SR	NFES	SR	NFES	SR	NFES	SR	NFES	SR
g01	**20304**	100%	-	-	-	-	51400	100%	184778	52%	46405	52%	95000	30%	102100	100%
g02	**61072**	**100%**	-	-	-	-	1328100	30%	-	-	-	-	175800	22%	219400	56%
g04	**3945**	100%	-	-	-	-	25700	100%	4896	100%	19681	100%	65100	30%	79300	100%
g06	1901	100%	1060	100%	**900**	100%	7300	100%	2424	100%	20007	100%	54200	30%	56800	100%
g07	**7281**	100%	11283	100%	7545	100%	116800	100%	14420	100%	327283	8%	227600	30%	358600	96%
g08	482	100%	-	-	-	-	1500	100%	**348**	100%	2311	100%	6850	30%	6100	100%
g09	**3436**	100%	4106	100%	3660	100%	77400	100%	5346	100%	57690	100%	78500	30%	96400	100%
g10	14734	100%	18781	100%	**8295**	100%	407400	100%	23780	100%	461422	32%	221300	30%	468350	16%
g12	**3809**	100%	-	-	-	-	7500	100%	26278	100%	3933	100%	6900	30%	8100	100%
g16	**3128**	100%	-	-	-	-	-	-	5648	100%	33021	100%	41000	30%	48700	100%
g18	**7272**	100%	-	-	-	-	-	-	57430	100%	177989	80%	153600	27%	211800	92%
g19	**25914**	100%	-	-	-	-	-	-	74472	100%	365284	8%	259650	14%	-	-
g24	**718**	100%	-	-	-	-	-	-	996	100%	7487	100%	19350	30%	19900	100%

Prob.	**mVIE**		**ASRES**		**(μ+λ)-CDE**		**ICDE**		**PCX**	
	NFES	SR	NFES	SR	NFES	SR	NFES	SR	NFES	SR
g01	**20304**	100%	62800	100%	89320	100%	106540	100%	62026	100%
g02	**61072**	100%	321200	12%	272860	100%	281470	100%	500000	64%
g04	**3945**	100%	57600	100%	30130	100%	36820	100%	40140	100%
g06	**1901**	100%	48400	100%	11200	100%	12880	100%	36180	100%
g07	**7281**	100%	135600	100%	139720	100%	135730	100%	258840	100%
g08	**482**	100%	4800	100%	2170	100%	1960	100%	3510	100%
g09	**3436**	100%	72000	100%	39550	100%	37870	100%	58700	100%
g10	**14734**	100%	276000	100%	188860	100%	325570	100%	109970	100%
g12	**3809**	100%	15600	100%	5110	100%	6580	100%	11940	100%
g16	**3128**	100%	39200	100%	18970	100%	25060	100%	36790	100%
g18	**7272**	100%	119600	96%	218050	100%	134680	100%	96180	100%
g19	**25914**	100%	212000	92%	265930	100%	297640	100%	187734	100%
g24	**718**	100%	14000	100%	5110	100%	5740	100%	13690	100%

References
Eps-DE: (Takahama and Sakai, 2006), **Eps-RDE**: (Takahama and Sakai, 2012), **MPDE**: (Tasgetiren and Suganthan, 2006), **GDE**: (Kukkonen and Lampinen, 2006), **MDE** :(Mezura-Montes et al., 2006), **SADE**: (Huang et al., 2006), **jDE-2**: (Brest et al., 2006), **(1+1)-aCMA**: (Arnold and Hansen, 2012), **(1+1)-VIE**: (Maesani and Floreano, 2014), **APM-CMA-ES**: (Kusakci and Can, 2013a), **AP-CMA-ES**: (de Melo and Iacca, 2014), **PSO**: (Zielinski and Laur, 2006), **COPSO**: (Aguirre et al., 2007), **DMS-PSO**: (Liang and Suganthan, 2006), **PESO+**: (Munoz-Zavala et al., 2006), **ASRES**: (Runarsson, 2006), **(μ+λ)-CDE**: (Wang and Cai, 2011), **ICDE**: (Jia et al., 2013), **PCX**: (Sinha et al., 2006), **GB-MA**: (Sun and Garibaldi, 2010)
Algorithms marked with (**) in the tables use gradient information.

Table 4.5: Median NFES and best fitness achieved for the engineering problems. All the compared results are obtained from the corresponding papers. Please note that if the median NFES to reach the optimum is not reported we show the full budget.

Prob.	**mViE**		**DEs1**		**DETPS**	
	NFES	Fitness	NFES	Fitness	NFES	Fitness
W. Beam	**6568**	1.724852	-	1.724852	66600	1.724852
Vessel	**14087**	5850.38306	8000	6059.714337	10000	5885.3336
Spring	21413	0.012665	8000	0.012665	10000	0.012665
C. Beam	**66894**	63893.490839	-	-	-	-
	MBA		**COPSO**		**SiC-PSO**	
W. Beam	47340	1.724853	30000	1.724852	24000	1.724852
Vessel	70650	5889.3216	30000	6059.714335	24000	6059.714335
Spring	**7650**	0.012665	30000	0.012665	24000	0.012665
C. Beam	-	-	-	-	-	-
	PSO-DE		**FA**			
W. Beam	-	10000	50000	1.731210		
Vessel	42100	6059.714335	25000	5850.38306		
Spring	42100	0.012665	-	-		
C. Beam	-	-	50000	63893.52		

References
DEs1: (de Melo and Carosio, 2012), **DETPS**: (Zhang et al., 2013), **PSO-DE**: (Liu et al., 2010), **MBA**: (Sadollah et al., 2013), **COPSO**: (Aguirre et al., 2007), **SiC-PSO**: (Cagnina et al., 2008), **FA**: (Gandomi et al., 2011)

Table 4.6: Median NFES to achieve the fixed accuracy level ($(f(\vec{x}) - f(\vec{x}*)) \leq 0.0001$) and Success Rate for the selected CEC 2006 problems. Comparison with methods that use traditional non-linear programming techniques that access gradient information. All the compared results are obtained from the corresponding papers. Results reported for GB-MA (Sun and Garibaldi, 2010) show the mean, rather than median NFES.

EA + NLP TECHNIQUES								
Prob.	**mVIE**		**SADE**		**DMS-PSO**		**GB-MA**	
	NFES	SR	NFES	SR	NFES	SR	NFES	SR
g01	20304	100%	25115	100%	25816	100%	**7859**	100%
g02	61072	100%	128970	84%	87107	84%	**45555**	100%
g04	3945	100%	25107	100%	25443	100%	**1201**	100%
g06	1901	100%	14404	100%	27636	100%	**489**	100%
g07	7281	100%	101240	100%	26685	100%	**3588**	100%
g08	**482**	100%	1272	100%	3892	100%	1068	100%
g09	3436	100%	16787	100%	29410	100%	**1632**	100%
g10	**14734**	100%	52000	100%	25500	100%	17319	100%
g12	3809	100%	1717	100%	6826	100%	**348**	100%
g16	**3128**	100%	14433	100%	28433	100%	7092	100%
g18	**7272**	100%	26000	92%	28000	100%	11095	100%
g19	25914	100%	51588	100%	**21587**	100%	**13355**	100%
g24	718	100%	4843	100%	18729	100%	**425**	100%

References
SADE: (Huang et al., 2006), **DMS-PSO**: (Liang and Suganthan, 2006),
GB-MA: (Sun and Garibaldi, 2010)

Table 4.7: Fitness, constraint violation (E), and design variables ($\boldsymbol{x}*$) of the best solutions discovered for the engineering problems.

Problem	Best Fitness (Median NFES)	E	$\boldsymbol{x}^*$
Welded Beam	1.724852 (6568)	0	$h = 0.205729627974134$ $l = 3.470488964774360$ $t = 9.036623829898325$ $b = 0.205729643534243$
Pressure Vessel	5850.383060 (14087)	0	$T_s = 0.75$ $T_h = 0.375$ $R = 38.8601036269430$ $L = 221.3654713560083$
Helical Spring	0.012665 (21413)	0	$d = 0.051699916331388$ $D = 0.356978944672547$ $n = 11.273668588601133$
Stepped Cantilever	63893.490839 (66894)	0	$w_1 = 3$ $h_1 = 60$ $w_2 = 3.1$ $h_2 = 55$ $w_3 = 2.6$ $h_3 = 50$ $w_4 = 2.204553242032800$ $h_4 = 44.091064840656017$ $w_5 = 1.749768676906988$ $h_5 = 34.995373538139674$

4.7 Supporting Information: Parameter Analysis

Given the impossibility of performing a full combinatorial exploration of the parameter space, we executed a preliminary tuning of the main parameters by following an iterative procedure. First, we set the initial parameters by empirical experimenting with the algorithm. We then analyzed the influence on the algorithmic performance varying each parameter, fixing identified optimal parameter values in several steps. We first investigated independently α and β_R, secondly L, F and CR, and finally P_{size}.

We evaluated the algorithm's performance by testing it on the full CEC 2006 benchmark for 25 runs. We allowed the algorithm to run for 150.000 function evaluations. We measured the success rate for each function and the factor of function evaluations with respect to the ones achieved by the best algorithm of the CEC 2006 competition to reach the optima at the accuracy of 10^{-4}. To obtain a more robust evaluation of the success rate, we tested each parameter combinations 5 times, for a total of 5 repetitions × 25 runs × 13 benchmark problems. We aggregated the success rate and the factor of function evaluations across all problems. We combined this information by ranking it separately and summing the rank of each parameter combination to select the (potentially) optimal parameters.

The results of this parameter tuning procedure is shown in Figure 4.6. Success rate, mean factor of FES and rank of a parameter combination is shown when varying α and β_R (Figure 4.6a), the minimum relative execution frequency limit L (Figure 4.6b), F and CR (Figure 4.6c), and the population size (Figure 4.6d). The final identified parameter values are $\alpha = 0.1, \beta_R = 0.05, L = 0.18, F = 0.5, CR = 0.9, P_{size} = 40$. Although this may be a locally optimal combination of parameters and better parameter tuning may be achieved by a more thorough parameter exploration, we considered this procedure sufficient for our purposes. Also, we obtained an initial idea of how the method's performance is affected by changing its parameters.

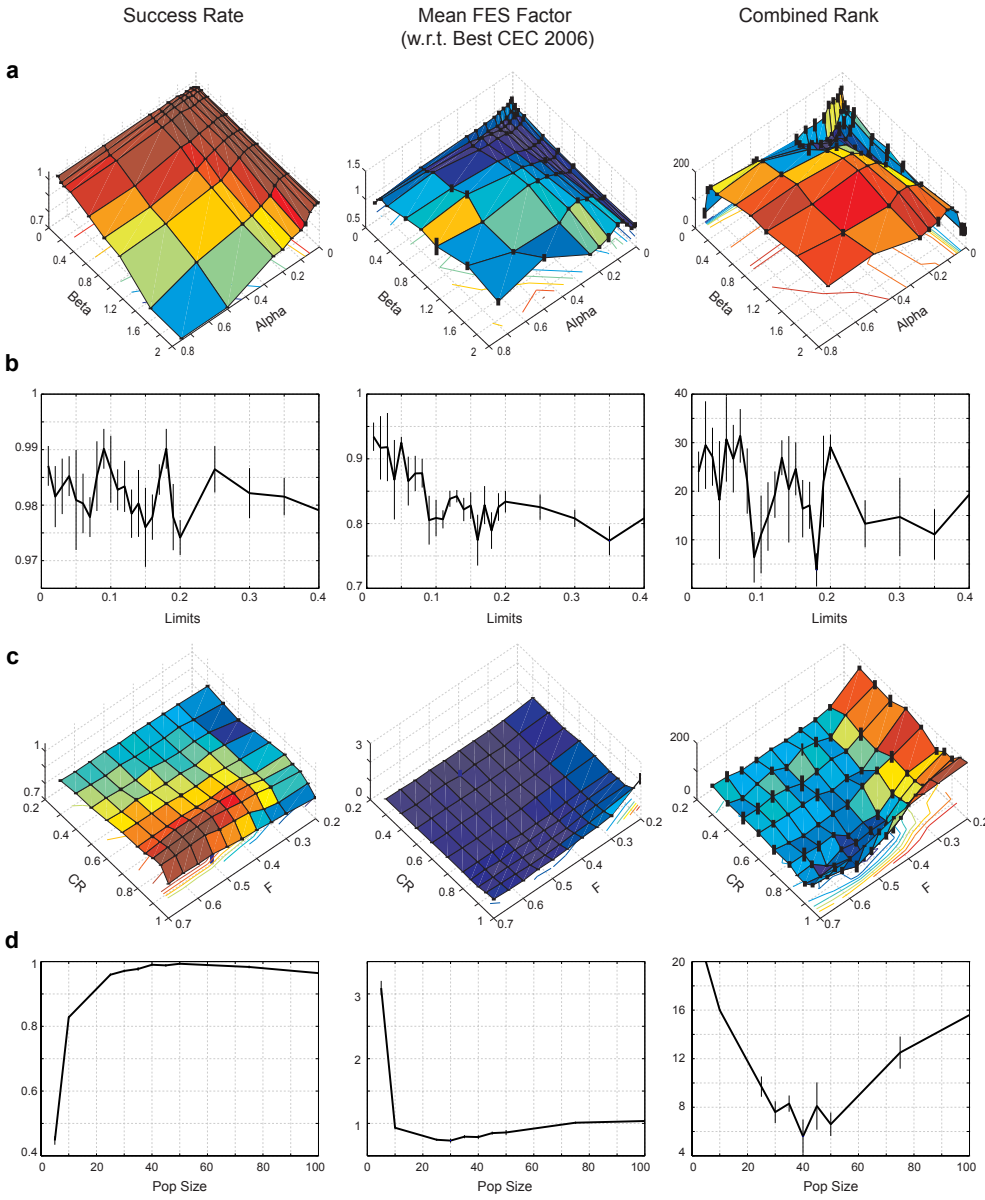


Figure 4.6: Parameter analysis: each reported result is the aggregation obtained by repeating 25 times each function (with inequalities only) of the CEC 2006 benchmark for 5 independent repetitions (i.e. 13 benchmark functions × 25 runs × 5 repetitions). The standard deviation across the 5 repetitions is reported as a black bar. We show mean success rate across the 13 benchmark functions and mean factor of function evaluations computed with respect to the best algorithm in the CEC 2006 competition. To select the best parameter combination for each of the parameters, we independently rank each parameter combination by success rate and mean factor of function evaluations. In the third column, we show the combined (summation) rank for each parameter combination. We report four parameter studies for **a**) α and β, **b**) minimum global/local search execution frequency limit L, **c**) F and CR parameters for the global search operator and **d**) population size P_{size}.

4.8 Supporting Information: CEC 2006 problem results - Error values achieved at different level of NFES

Table 4.8: Error values (difference between the known optimum fitness and the best fitness value) achieved when NFES= 5×10^3, NFES= 5×10^4, and NFES= 5×10^5 for the selected CEC 2006 problems. Best, median, worst and mean error values are reported in the table, together with the number of violated constraints c and the average sum of constraint violation $\bar{v}$ of the median solution.

NFES		g01	g02	g04	g06	g07
$\mathbf{5 \times 10^3}$	Best	0.695563 (0)	0.132752 (0)	0 (0)	0 (0)	0.001159 (0)
	Median	1.91028 (0)	0.302738 (0)	1e-06 (0)	0 (0)	0.015729 (0)
	Worst	3.7779 (0)	0.418978 (0)	0.202275 (0)	0.02595 (0)	1.47459 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	2.429400e-02	0	0	6.130000e-04	2.052570e-01
	Mean	2.1593	0.287957	0.00825104	0.00141272	0.110063
	Std	0.822658	0.0663392	0.0404259	0.00529882	0.307058
$\mathbf{5 \times 10^4}$	Best	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
	Median	0 (0)	0.011011 (0)	0 (0)	0 (0)	0 (0)
	Worst	7.6e-05 (0)	0.132974 (0)	0 (0)	0 (0)	0.0001 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	4.013410e-01	0	0	1.375290e-01	3.400200e-01
	Mean	1.968e-05	0.0228126	0	0	4e-06
	Std	2.7308e-05	0.0296265	0	0	2e-05
$\mathbf{5 \times 10^5}$	Best	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
	Median	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
	Worst	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	4.964673e+00	0	6.900000e-05	8.971150e+00	4.285058e+00
	Mean	0	0	0	0	0
	Std	0	0	0	0	0

4.8. Supporting Information: CEC 2006 problem results - Error values achieved at different level of NFES

NFES		g08	g09	g10	g12	g16
5×10^3	Best	0 (0)	0 (0)	10.0031 (0)	0 (0)	0 (0)
	Median	0 (0)	0 (0)	216.449 (0)	0 (0)	0 (0)
	Worst	0 (0)	0.410366 (0)	4434.49 (0)	0.005625 (0)	0.000864 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	9.640000e-04	0	1.444769e+03	0	1.902000e-03
	Mean	0	0.0224805	922.665	0.00116472	8.572e-05
	Std	0	0.0833781	1447.02	0.00227881	0.000239219
5×10^4	Best	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
	Median	0 (0)	0 (0)	3e-06 (0)	0 (0)	0 (0)
	Worst	0 (0)	0 (0)	0.265618 (0)	0 (0)	0 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	1.057000e-03	0	2.776000e-03	2.800000e-05	1.864644e+01
	Mean	0	0	0.01235	0	0
	Std	0	0	0.0530731	0	0
5×10^5	Best	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
	Median	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
	Worst	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	1.057000e-03	9.073334e+00	3.134842e+03	2.800000e-05	1.082927e+02
	Mean	0	0	0	0	0
	Std	0	0	0	0	0

NFES		g18	g19	g24
5×10^3	Best	5e-05 (0)	13.4913 (0)	0 (0)
	Median	0.010392 (0)	43.6379 (0)	0 (0)
	Worst	0.595968 (3)	107.837 (0)	0 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	4.908650e-01	0	0
	Mean	0.095619	46.6881	0
	Std	0.154321	19.8156	0
5×10^4	Best	0 (0)	0 (0)	0 (0)
	Median	0 (0)	0 (0)	0 (0)
	Worst	0.191044 (0)	2e-06 (0)	0 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	4.561900e-01	0	1.494000e-03
	Mean	0.0152835	8e-08	0
	Std	0.0528977	4e-07	0
5×10^5	Best	0 (0)	0 (0)	0 (0)
	Median	0 (0)	0 (0)	0 (0)
	Worst	0 (0)	0 (0)	0 (0)
	c	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	7.948723e+00	0	4.546000e-03
	Mean	0	0	0
	Std	0	0	0

5 Application of evolutionary computation to neuroscience

In this final chapter, we use evolutionary computation to shed some light onto a real neuroscientific problem: the role of neural noise in animal behaviors. Although stochastic neural activity is present throughout nervous systems, the influence of noise on behavior is yet far to be fully understood. Here, we addressed this question by pairing high-resolution measurements of walking dynamics across genetically diverse *Drosophila* strains with an unbiased circuit model discovery approach. This discovery approach used evolutionary computation for fitting the weights and parameters of neural network models. Coincidentally, this investigation represents a good playground for testing the mViE algorithm, devised previously, on a real problem. After presenting the results of this comparison we move on to describe the outcome of this exciting investigation. We show that noise is required by discovered models to reproduce *Drosophila* spontaneous walking patterns. Mechanistically, noise switches neural activity between stable equilibrium levels reflecting walking and rest. We validated these noise-driven models in two ways. First, these models reproduced the complex dynamics of odor-evoked walking for diverse *Drosophila* strains due to a noise-mediated circuit memory of odor stimulation. Second, we confirmed *in vivo* a predicted effect of circuit output threshold on behavioral noise sensitivity, implicating a presynaptic voltage-gated calcium channel in threshold determination. Our data-driven, model discovery approach uncovers an important role for noise in animal behavior: stochastic neural activity can regulate spontaneous exploration and shape sensory response dynamics.

The contents of this chapter are taken from:

Maesani A[*], Ramdya P[*], Cruchet S, Gustafson K, Massouras A, Deplancke B, Benton R[‡] and Floreano D[‡] (2014). Neural noise shapes *Drosophila* behavior. In revision.

[*], [‡] These authors contributed equally.

DISCLOSURE: This chapter is the outcome of an exciting collaboration that required the competences of many people. We report for completeness all the results of the whole investigation. The contents of this chapter are taken from a journal paper in revision authored by Andrea Maesani[*], Pavan Ramdya[*], Steeve Cruchet, Kyle Gustafson, Andreas Massouras, Bart Deplancke, Richard Benton[‡] and Dario Floreano[‡]. [*], [‡] These authors contributed equally. I designed and developed the system for automatic neural network discovery. I used this system to automatically derive models for explaining spontaneous and odor-evoked behaviors. I analyzed *Drosophila* data and carried out the dynamical system analysis of the best neural models. I designed, performed and analyzed the computational experiments.

5.1 Introduction

Noisy neural activity – arising, for example, from the stochastic opening and closing of ion channels – is widely documented throughout animal nervous systems (Faisal et al., 2008). Nevertheless, its role remains controversial: does noise simply muddle meaningful neural activity evoked by sensory stimuli, representing a barrier to be overcome (Faisal et al., 2005), or can noise contribute effectively to neural signaling and behavior (McDonnell and Ward, 2011)? Noise may have a particularly important effect on neural circuits driving spontaneous behaviors, which occur in the absence of external stimuli (Martin et al., 1999; Flavell et al., 2013). However, how noise in neural circuits can produce and/or influence spontaneous behaviors, and how it impacts sensory-evoked actions are unknown.

Drosophila Melanogaster is an attractive model organism for studying the role of neural noise in behavior due to the numerical simplicity of its nervous system, with a relatively small number of synapses separating the sensory and motor peripheries (Ruta et al., 2010; Gordon and Scott, 2009). Additionally, powerful genetic tools are available and *Drosophila* spontaneous and sensory behaviors are increasingly well-described (Sorribes et al., 2011; Martin et al., 2001; Martin, 2004; Maye et al., 2007; Proekt et al., 2012; Censi et al., 2013; Coen et al., 2014). Studies of spontaneous behavior in flies have focused primarily on walking because this behavior has reproducible statistics and can be measured at high-throughput (Martin et al., 1999; Sorribes et al., 2011; Martin et al., 2001). The neural circuits that control walking patterns are only beginning to be understood: two brain regions (the mushroom body and ellipsoid body) have been implicated in spontaneous walking patterns (Sorribes et al., 2011; Martin et al., 2001) and a set of command neurons that drive backwards walking have been identified (Bidaye et al., 2014). However, whether and how neural noise regulates the activity of walking circuits, with potentially profound consequences on the timing and duration of walking, remains unknown.

Investigating the role of noise in behavior is challenging for several reasons. First, a noise-dependent behavior would be expected to be largely unpredictable, independent of environmental features, and variable even for single individuals. Therefore, uncovering consistent

behavioral patterns requires averaging across individuals from large-scale, high-resolution datasets. Second, because noise in the nervous system likely emerges from molecular processes fundamental to neuronal signaling and viability (Faisal et al., 2008), the subtle manipulation of these processes required to reveal a relationship between noise and behavior is technically difficult. In this light, computational experiments can be instructive to inform and complement in vivo studies. Effective modeling approaches are often grounded in biological data both at conception and during validation. However, model derivation has typically depended on human intuition (Reichardt, 1961), an approach that is very difficult to apply to inherently complex noise-driven processes.

Here, we overcome this limitation by exploiting the power of unbiased computational discovery to generate and study noise-driven circuit models. Unlike previous circuit discovery studies (Beer and Gallagher, 1992; Dunn et al., 2004; Izquierdo and Lockery, 2010), our approach is data-driven, building neural circuit models to best reproduce measurements of *Drosophila* spontaneous walking. Using this model discovery framework, we first examined the potential role of noise in *Drosophila* spontaneous walking by testing if noise-driven neural circuit models, rather than models relying only on circuit dynamics, could better reproduce measured walking patterns. We validated the resulting circuit models in two ways. First, we tested if models for spontaneous walking could also explain the time-course of sensory-evoked walking in genetically distinct *Drosophila* strains. Second, we tested predictions made by the model using *Drosophila* gene discovery and genetic perturbations in vivo.

5.2 Analysis of *Drosophila* walking

5.2.1 A high-resolution, high-throughput assay for measuring *Drosophila* spontaneous and odor-evoked walking patterns

A substantial amount of high-resolution behavioral data is required to permit detection of patterns in largely probabilistic behaviors of individuals and to uncover a role for noise in the timing of walking bouts. We acquired data of the required scale by combining synchronized video-capture and odor delivery (Ramdya et al., 2012) (Fig. 5.1a) with behavioral tracking (Branson et al., 2009) (Fig. 5.1b) to measure the position and orientation of flies in a planar arena. We studied three conditions: first, during 60 seconds of spontaneous behavior (in the absence of an odor stimulus), second, during and after 30-seconds of uniform exposure to the aversive odorant 10% acetic acid (Ai et al., 2010), and third, during a two-minute odor aversion assay in which the odor was presented on either side of the arena in an alternating fashion (Fig. 5.1c).

While much information can be extracted from our measurements, we focused in this work on the presence or absence of walking. Spontaneous walking is characterized by bursts of activity separated by longer intervening sedentary periods (Martin et al., 1999; Sorribes et al., 2011). As for previous studies (Valente et al., 2007; Wolf et al., 2002), we applied a cutoff on speed

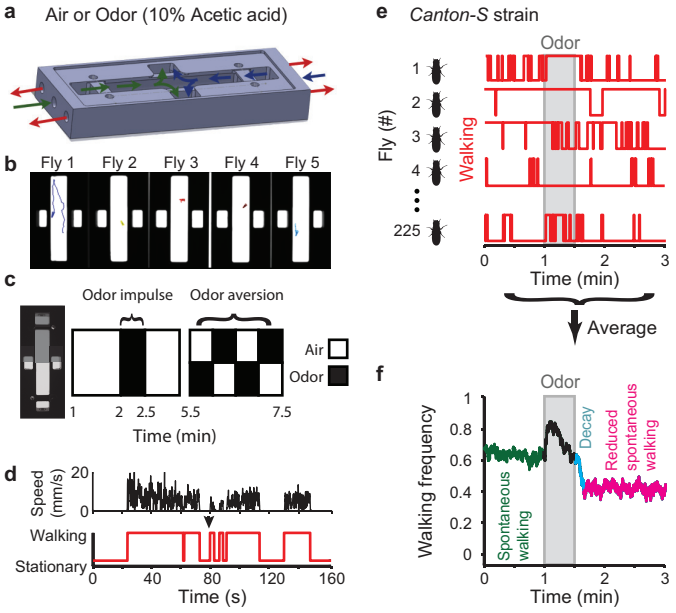


Figure 5.1: A high-resolution, high-throughput assay for measuring *Drosophila* spontaneous and odor-evoked walking patterns. (a) Schematic of planar behavioral arenas. Laminar flow of odor or air can be presented to either or both halves of the arena. Colored arrows indicate flow inlets and outlets (green/blue and red, respectively). (b) Camera-view of five experimental arenas demonstrating behavior tracking with Ctrax. Each fly is represented by a colored triangle and a colored line represents its position over the previous ten seconds. (c) Schematic time-course of the behavioral experiment. We studied spontaneous and odor impulse evoked walking as well as odor aversion for each fly. (d) Speed time-series (black) were transformed into binary 'Walking' or 'Stationary' time-series (red). (e) Representative walking traces for five flies from the *Canton-S* strain during the odor impulse experiment. Flies were exposed to 60s of airflow, 30s of an odor throughout the arena, and 90s of airflow. Behavior for each fly is shown in red. High values indicate walking and low values indicate stationary periods. (f) Walking frequency averaged across 225 *Canton-S* strain flies during the odor impulse experiment. Prior to odor stimulation (grey bar) there is spontaneous walking (green) followed by decay in walking frequency (cyan) to a reduced spontaneous walking frequency (magenta).

measurements to classify flies as either walking or stationary at each time point (Fig. 5.1d). Even after this simplification, the time-courses of walking for individual *Drosophila* were unpredictable (Fig. 5.1e). To reveal patterns behind these seemingly probabilistic behaviors, we averaged walking/stationary time-series across 225 genetically identical flies of a single, *Canton-S* strain. We found that, prior to odor stimulation, flies exhibit a high basal, spontaneous walking frequency (Fig. 5.1f, green). Upon odor presentation, walking frequency increased rapidly (Fig. 5.1f, black). Although odor disappeared from the arena within one second of valve switching (Fig. 5.2a), walking did not immediately decrease to its previous spontaneous frequency but decayed more slowly (Fig. 5.1f, cyan). Surprisingly, it did not return to the pre-odor spontaneous walking frequency but rather to a substantially lower rate

(Fig. 5.1f, magenta).

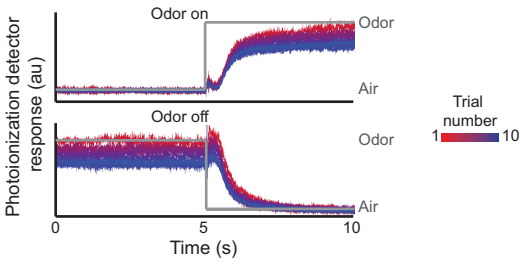


Figure 5.2: Arena odor flow kinetics. Photoionization detector measurements (arbitrary units [au]) of odor flow (10% acetic acid). A grey line indicates odor flow (high) or airflow (low). Each colored trace represents one of ten trials. Odor onset (top) and removal (bottom) are shown.

5.2.2 Walking patterns are diverse but reproducible across genetically distinct strains of *Drosophila*

To examine the variation in spontaneous and sensory-evoked walking behaviors across genetically-distinct *Drosophila* strains, we tracked groups of approximately 200 flies from each of 98 different inbred *Drosophila* strains of the *Drosophila* Genetic Reference Panel (DGRP) (Mackay et al., 2012). These experiments resulted in a high-resolution behavioral dataset comprising 20,223 flies. Across all 98 inbred strains (Fig. 5.3a, see Table S1 for corresponding RAL numbers) we observed spontaneous walking, prolonged walking decay after odor termination (Fig. 5.3b), and reduced post-odor spontaneous walking (Fig. 5.3c) as seen in *Canton-S*. However, the duration of these behavioral features varied dramatically across strains (Fig. 5.3a).

This rich behavioral diversity might arise from random experimental variation or, alternatively, it might reflect intrinsic biological differences between each strain. To distinguish between these possibilities, we examined the reproducibility of walking behaviors within each strain. We found that average spontaneous walking was highly reproducible for each strain (Fig. 5.3d). Similarly, while the reproducibility of odor-evoked and post-odor walking could differ substantially across strains (Fig. 5.3e), the time-course of this behavior was more similar between flies of the same strain (Fig. 5.3e, red) than flies of different strains (Fig. 5.3e, blue). Taken together, these data imply that inter-strain behavioral differences are genetically encoded, perhaps through the tuning of neural circuits that regulate the timing and duration of walking.

Circuits that determine walking patterns (Fig. 5.1f, green) are likely to lie just upstream of rhythmically active central pattern generators (CPGs) that coordinate leg movements during locomotion (Grillner and Jessell, 2009). Given this proximity to output motor circuits, the same circuits are probably responsible for both spontaneous and sensory-evoked walking. To test this possibility, we examined whether spontaneous walking might predict the intensity of odor aversion. For each fly, we calculated odor aversion as the time spent in the air minus time

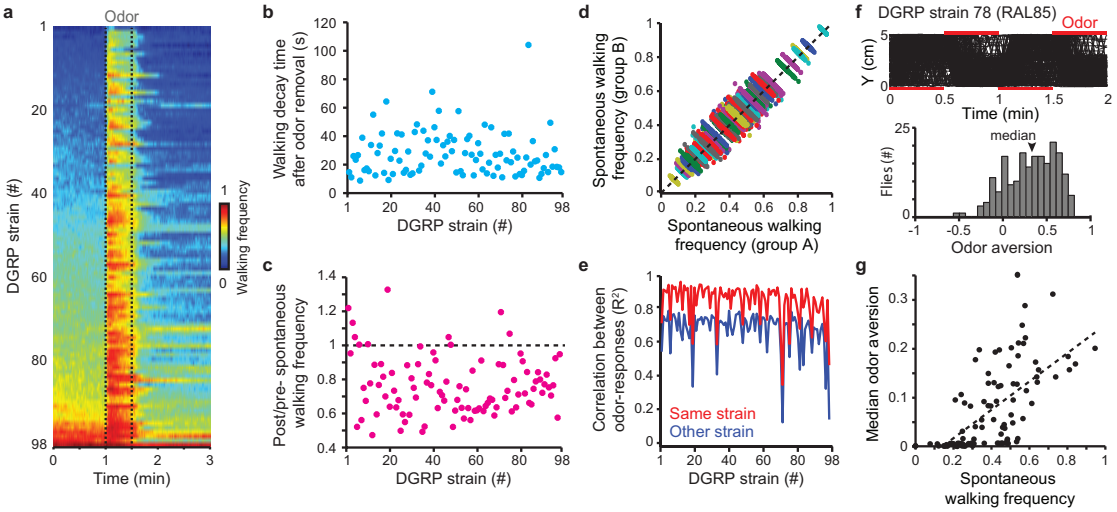


Figure 5.3: Walking patterns are diverse but reproducible across genetically distinct strains of *Drosophila*. (a) Walking frequency for 98 distinct inbred fly strains from the *Drosophila* Genetic Reference Panel (DGRP) during the odor impulse experiment. Strains are sorted by mean pre-odor spontaneous walking frequency. (b) The duration of walking frequency decay after odor removal for all DGRP *Drosophila* strains. Strains are ordered as in panel a. (c) The ratio between post-odor and pre-odor spontaneous walking frequency for all DGRP *Drosophila* strains. Strains are ordered as in panel a. A black dashed line indicates no change in spontaneous walking frequency. (d) A scatter plot showing the correlation between spontaneous walking for two groups of 65 flies taken from a single DGRP strain. 100 randomly sampled groups (A and B) were tested for each strain. Each DGRP strain is color-coded. (e) The correlation (R2) between odor-evoked walking time-series for groups of 65 randomly sampled flies from either the same strain (red) or from different *Drosophila* strains (blue). Strains are ordered as in panel a. The mean of 100 correlation tests is shown. (f) Walking trajectories (black lines) along the y-axis during the odor aversion experiment for 201 flies from DGRP strain 78 (RAL85). Red bars indicate the half of the arena with odor flow. A histogram of odor aversion for these flies is shown below. For each fly, odor aversion was calculated as the time spent in odor subtracted from the time spent in air, divided by the total time of the odor aversion experiment. The median for this population of flies is indicated (black arrowhead). (g) A scatter plot showing the correlation between mean pre-odor spontaneous walking frequency and median odor aversion across all 98 inbred strains (Pearson's correlation coefficient R = 0.65, $P < 10-5$). The black dashed line indicates the best linear fit.

spent in the odor, divided by the total time of the odor aversion experiment. We characterized each strain by the median odor aversion of the entire fly population (Fig. 5.3f). Median odor aversion varied substantially across 98 inbred strains and this variation correlated significantly with spontaneous walking frequency (Fig. 5.3g, Pearson's correlation coefficient R = 0.64, P<10-4). This suggests that the same circuits might be responsible for regulating the timing and duration of spontaneous and sensory-evoked walking. We therefore set out to generate models for *Drosophila* walking circuits that can explain both spontaneous walking patterns

and the time-course of odor-evoked walking in order to ask what role noise plays, if any, in each of these behavioral processes.

5.3 An automated circuit model discovery approach

Since spontaneous walking arises in the apparent absence of external sensory stimulation, it best reflects the basal ongoing activity of neural circuits that regulate *Drosophila* walking. Therefore, we used spontaneous walking as a target behavior for our model discovery approach (Fig. 5.4a). In our initial experiments, we observed that some flies could remain stationary for over 20 minutes (data not shown). In order to capture the full range of behavioral intervals, we therefore acquired 5 hours of spontaneous walking sequences from *Canton-S* strain flies (Fig. 5.4b, top). We confirmed the spontaneous nature of these walking events: although flies had a tendency to walk near arena edges (Fig.5.5a), the duration of stationary intervals did not depend on arena location (Fig. 5.5b-c, $P > 0.05$ Bonferroni corrected Wilcoxon test).

The exact time-courses of individual fly behaviors depend on many, often unknown, factors. Therefore, we aimed to discover circuit models that reproduce the duration of walking bout and stationary period intervals. Our model discovery method is based on evolutionary optimization and therefore requires well-defined quantitative metrics for comparing candidate circuit models with *Drosophila* behavior (i.e., a cost function for guiding the search for models). Additionally, to efficiently search for models, small changes of model parameters should result in similarly incremental changes in these quantitative metrics (i.e., a smooth fitness landscape) (Nelson et al., 2009). Therefore we normalized behavioral histograms in two ways. First, each histogram bin was multiplied by its own interval duration to ensure that frequent, short duration walking bouts were not over-valued. Second, empty bins were removed from each histogram by using variable bin-widths (Fig. 5.6a; see Methods). The resulting histograms provide a quantitative measure reflecting *Drosophila* spontaneous walking patterns (Fig. 5.4b, bottom).

To generate models for *Drosophila* spontaneous walking circuits we employed a well-established neural network modeling framework, the Continuous Time Recurrent Neural Network (CTRNN) (Beer and Gallagher, 1992). CTRNNs are an intermediate representation of neural circuits that do not model precise ionic conductances or action potential generation but still retain the dynamical characteristics of neurons. Interpretations of these phenomenological circuit models range from a one-to-one correspondence, where each model neuron represents an identified in vivo neuron, to more abstract representations where each model neuron represents an in vivo neural circuit. In both cases, but particularly in the latter case, the dynamical properties of discovered circuit models, rather than their precise connectivity, are the instructive features (Marder and Taylor, 2011). Our CTRNN models had recurrent or reciprocal connections between neurons and could, depending on the experiment, have a Gaussian noise input representing neural noise (Fig. 5.4c). For each model, one neuron was randomly selected as the model's output neuron (N_{OUT}) driving a downstream CPG for walking. If this

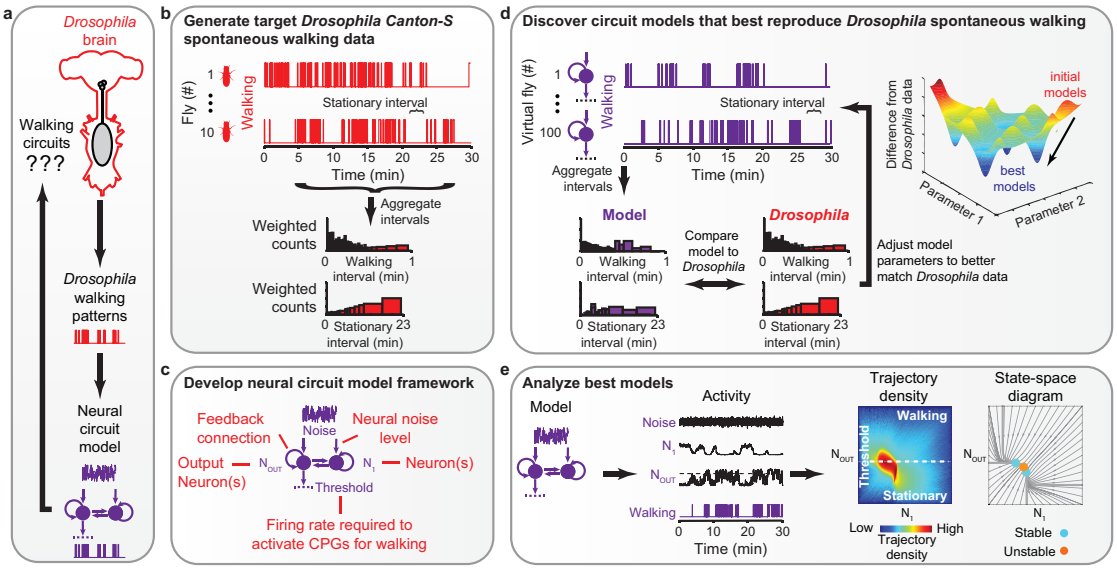


Figure 5.4: Automated circuit model discovery and analysis workflow. (a) The *Drosophila* nervous system generates walking patterns (red). In silico circuit models that reproduce these walking patterns (purple) can provide insights into *Drosophila* neural circuits for walking. (b) Spontaneous walking patterns (filtered as in Fig. 1d) are shown for two of ten *Canton-S* strain flies recorded for 30 minutes each. A stationary interval is highlighted for fly number ten. From these data, walking and stationary interval durations are aggregated and represented using weighted variable-width histograms for walking (top) and stationary (bottom) behaviors (see Fig. S3). (c) Continuous-time Recurrent Neural Network (CTRNN) models used to reproduce *Drosophila*spontaneous walking. Circuit models include up to five neurons, Gaussian noise, reciprocal and recurrent connections, and an output threshold to define the walking state of the virtual fly. (d) An iterative heuristic approach to identify model parameters that best reproduce spontaneous walking patterns. Virtual fly walking patterns were compared to *Drosophila* data and the parameters for models in the next iteration were adjusted to more closely approximate the best performing models of previous iterations (left), in a manner resembling gradient descent through a fitness landscape (right). (e) 50 'best models' were discovered for each model size. Noise input, neural activity, and walking for a single 2-neuron model are shown. This, and other, models were characterized by the number of times specific levels of neural activity were observed ('Trajectory density') color-coded from very frequent (dark red) to very rare (dark blue). Models were also characterized by the tendency of neural activity, in the absence of noise, to move toward ('Stable', cyan) or away from ('Unstable', orange) equilibrium levels of activity ('State-space diagram').

output neuron's activity level, or mean firing rate, exceeded a threshold value, the virtual fly was walking. Otherwise, the virtual fly was stationary.

Specifically, we used an evolutionary optimization algorithm to discover the best circuit model parameters (Fig. 5.4d). To find these 'best' circuit models, we first generated a population of 50 models with randomly chosen parameter values for each network size (1-5 neurons).

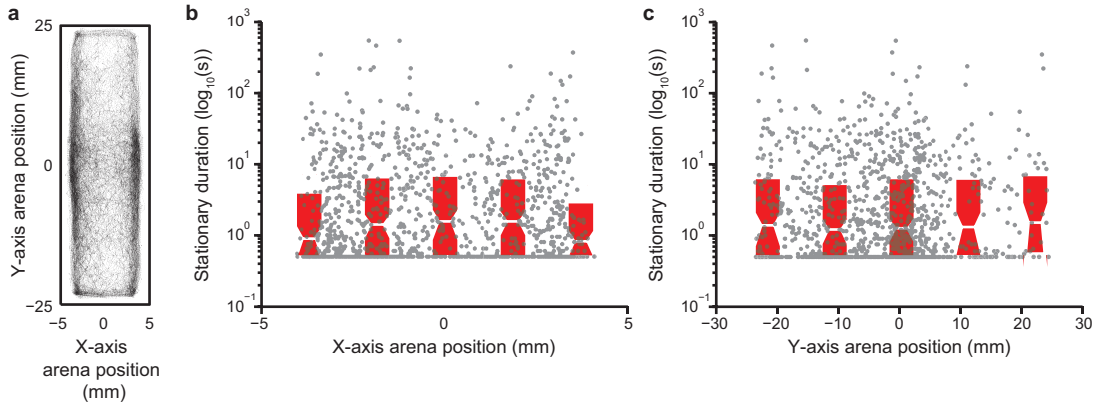


Figure 5.5: Spontaneous walking of Canton-S flies. (a) Spontaneous walking of ten *Canton-S* strain flies within the experimental arena across five hours. Each black circle represents the location of one fly at one time-point. (b-c) The relationship between x-axis (b) or y-axis (c) arena position and the duration of each stationary interval. Grey circles represent single stationary events. Red boxplots summarize data over 2 mm (b) or 10 mm (c) sections of the arena.

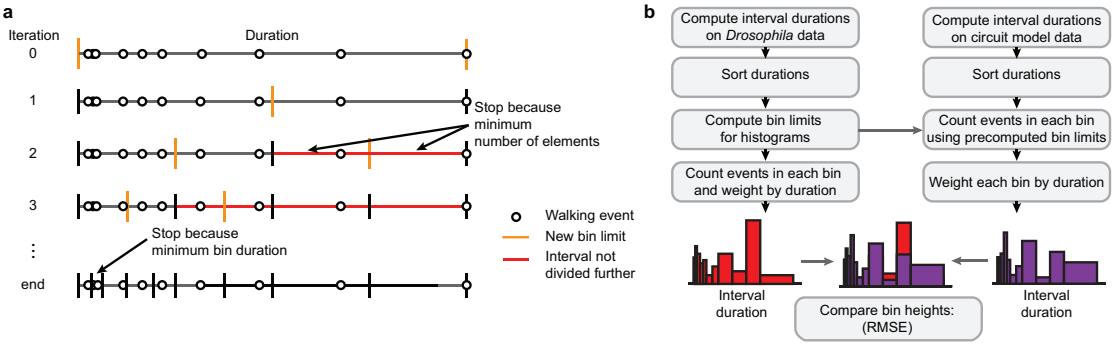


Figure 5.6: Procedure for generating and comparing weighted variable-width histograms. (a) The procedure for determining bin-width sizes for variable bin-width histograms of Drosophila Canton-S strain walking and stationary interval durations. (b) The workflow for generating weighted, variable bin-width histograms for Drosophila Canton-S data (left) to compare with circuit model data (right). Histograms were compared using a Root-Mean-Square Error (RMSE) to determine the cost function or 'Difference from Drosophila data'.

We simulated each model 100 times. An output threshold was then applied to the activity of the model's output neuron (N_{OUT}), resulting in a binary (walking or stationary) time-series. We then aggregated the walking and stationary interval durations from this time-series and compared these histograms to the target *Drosophila Canton-S* strain spontaneous behavior histograms. By comparing the performance of each model, we identified the best models and iteratively adjusted the population of models towards the best models' parameters. This process was repeated a predetermined number of times to discover model parameters that

minimize the difference between virtual fly walking and *Drosophila* spontaneous walking patterns. This cost function, 'Difference from *Drosophila* data', is a non-linear measure and may be intuitively understood based on the percent of the *Drosophila* data reproduced. To derive this relationship, we measured the difference between the full *Drosophila* dataset and subsets of the same data (Fig. 5.14a).

We performed circuit model discovery 50 times for each network size (1-5 neurons) in the absence or presence of noise. This resulted in 500 candidate circuit models. We analyzed the best of these models in two ways (Fig. 5.4e). First, we identified the most common neural activity levels for each model using a trajectory density representation. This allowed us to study the effect of noise on circuit activity. We complemented this with dynamical systems analysis of each model in the absence of noise. This allowed us to identify equilibrium points in neural state-space: activity levels that model neurons tended to settle towards ('Stable') or move away from ('Unstable'). For example, a single stable equilibrium point could represent a neuron's spontaneous firing rate. This dynamical systems approach allows useful conceptual links to be drawn between biological and in silico neural circuits that have very different or even unknown topologies (Mante et al., 2013).

5.3.1 Evolutionary Computation for neural circuit discovery

The optimization of neural network weights and parameters is a very difficult problem, due to the large amount of networks with equally bad or trivial behaviors present in the search landscape. This creates large plateaus in the fitness landscape, making the search very complicated. Although the mViE algorithm presented in the previous chapter has been developed for constrained optimization, we decided to test it on this network discovery problem against a well-known evolutionary method: Particle Swarm Optimization (Eberhart and Kennedy, 1995). We tested each method repeating 50 independent runs evolving networks with 1 or 2 neurons with or without noise inputs. The results of this comparison are shown in Figure 5.7.

PSO and mViE perform as badly in the noiseless network evolution scenario. No networks with sufficiently good fitness can be discovered. In the noisy network evolution scenario, the methods compare almost equivalently in the one neuron case, while PSO has a slight advantage in the two neuron case. However, although the two methods produce solutions of about the same quality, we decided to use particle swarm for the rest of this neuroscience investigation, due to the fact that it is an established method in the literature and is already used in many fields of science and engineering. The presentation to a broader audience of this neuroscience investigation can be less problematic if an established method is used, rather than the new method presented in this thesis.

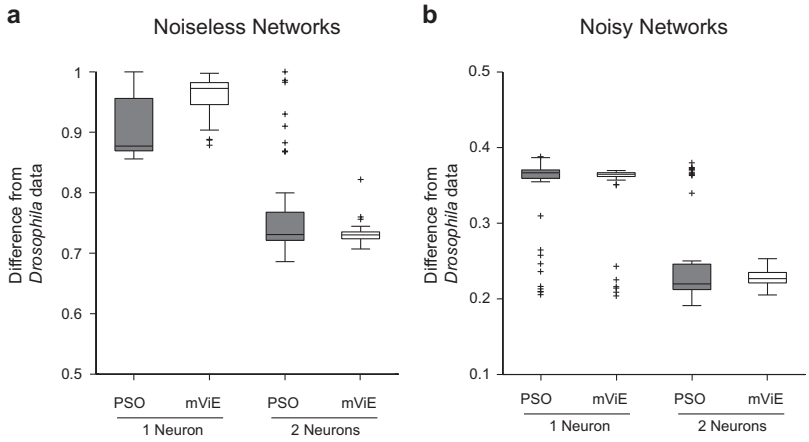


Figure 5.7: Comparison of mViE and PSO on noiseless and noisy network models. Due to the high computational cost of our simulations, PSO and mViE were compared on 1 and 2 neurons networks. a) Comparison between PSO and mVeE on the noiseless networks. Both mViE and PSO cannot discover good network models using one or two neurons. b) Comparison between PSO and mViE on the noisy networks.

5.4 Results

5.4.1 Noise-driven multistable circuit models reproduce *Drosophila* spontaneous walking

Using this circuit model discovery approach, we first asked whether neural noise is required by models to best reproduce *Drosophila* spontaneous walking patterns. We found that noiseless circuit models perform very poorly compared to noise-driven circuit models (Fig. 5.8a, n = 250 models with 1-5 neurons, $P < 0.001$,Wilcoxon Rank Sum Test). Noiseless models could only capture 40% of *Drosophila* spontaneous walking data (Fig. 5.14a,c-e) while noise-driven circuit models could match up to 90% of Drosophila walking patterns (Fig. 5.14a). Although 2-neuron noise-driven models were the minimum size that most effectively reproduced *Drosophila* spontaneous walking (Fig. 5.8b), even noise-driven models with only one neuron could successfully match fly behavior (Fig. 5.8b), suggesting that simple noise-driven neural circuit dynamics can explain *Drosophila* spontaneous walking patterns. Importantly, a threshold applied to Gaussian noise in the absence of any circuit model could at best only match 30% of fly data (Fig. 5.14a-b). This shows that a coupling between noise and circuit dynamics – and not simply the statistics of noise – is required to reproduce *Drosophila* walking.

Since the best noise-driven circuit models exhibited a wide range of parameter values, we asked whether they shared common dynamical properties. Across all noise-driven models, we observed two distinct groups (Fig. 5.8c; and data not shown for >2 neurons). Models in the first group generally had lower agreement with *Drosophila* data (Fig. 5.8c, grey, 'Monostable'). These models exhibited neural activity that fluctuated around a single stable activity level

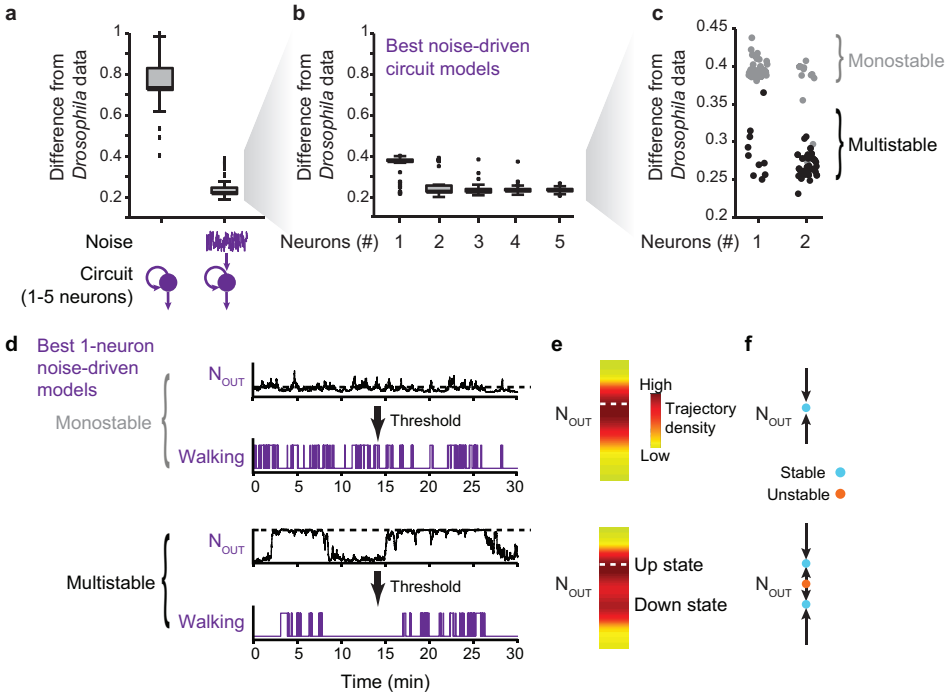


Figure 5.8: Noise-driven multistable models reproduce *Drosophila* spontaneous walking. (a) The difference between spontaneous walking patterns for *Drosophila* and circuit models without (left) or with (right) noise inputs (n = 250 models for each condition with 50 models for each circuit size). (b) The difference between spontaneous walking patterns for *Drosophila* and noise-driven circuit models of a given size (n = 50 models for each circuit size). (c) The best 1-neuron and 2-neuron noise-driven circuit models are 'Monostable' (grey) or 'Multistable' (black) (see Fig. S5 for further subdivision of 2-neuron noise-driven multistable models). (d) The time-course of output neural activity (N_{OUT}), and walking/stationary periods (purple) from a 'Monostable' (top) or a 'Multistable' (bottom) 1-neuron noise-driven circuit model. (e) Trajectory densities for the models shown in panel d. The 'Multistable' model shows two frequent activity levels labeled 'Up-state' or 'Down-state'. The threshold between walking and stationary is shown (white dashed line). (f) State-space diagrams for the models in panel d. Stable (cyan) and unstable (orange) equilibrium points are indicated.

(Fig. 5.8c, top). By contrast, models of the second group matched fly data more closely (Fig. 5.8c, black, 'Multistable') and exhibited neural activity that fluctuated between two levels, commonly referred to as 'Up' and 'Down' states (Cossart et al., 2003) (Fig. 5.8d-e, bottom). These levels represent stable equilibrium points in neural activity state-space (Fig. 5.8f).

The best 2-neuron noise-driven multistable models could be further subdivided into three classes based on the number of times that their neural activity switched between Up and Down states (Fig. 5.15a). This diversity of classes demonstrates that our automated approach for generating circuit models could effectively explore the landscape of possible solutions. Nevertheless, all of these model classes shared common dynamical properties (Fig. 5.15c-d).

First, neural noise allowed them to switch between two stable levels of neural activity. Second, their output threshold, determining the behavioral state (walking or stationary), intersected the region representing high neural activity. In sum, the best circuit models reproduced *Drosophila* spontaneous walking patterns through noise-driven fluctuations around an Up-state to produce bursts of walking, and excursions into a Down-state to produce long sedentary periods.

5.4.2 Circuit models for spontaneous behavior also reproduce odor-evoked walking dynamics

As a first validation of our best circuit models, we tested the hypothesis that common neural circuit dynamics underlie both spontaneous and sensory-evoked walking patterns. Specifically, we asked whether circuit models generated to reproduce *Drosophila* spontaneous walking also replicate the time-course of odor-evoked walking (Fig. 5.1g, 'Decay' and 'Reduced spontaneous walking'). Due to computational time constraints, we randomly selected three *Drosophila* DGRP strains that represent a large proportion of the behavioral variation across the 98 analyzed fly strains (Figs. 5.3a & 5.9) and attempted to match the time-course of their odor-evoked walking responses using the best 1-neuron noise-driven multistable model (Fig. 5.8c) and the best 2-neuron noise-driven multistable models from each class (Fig. 5.15b-c). To mimic sensory stimulation, for each model we added a virtual olfactory input to each neuron (Fig. 5.10a, top). While keeping all other model parameters fixed as for spontaneous walking, we iteratively tuned the odor input strength and circuit output threshold to best match a given *Drosophila* strain's odor-evoked walking response (Fig. 5.10a, bottom).

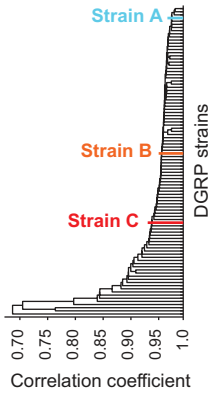


Figure 5.9: A dendrogram of the correlation between odor-evoked walking dynamics across 98 DGRP inbred *Drosophila* strains. Hierarchical clustering distance based on the Pearson's correlation coefficient between odor-response time-series for each strain. Three strains chosen for further analysis are indicated. Strains A (RAL57), B (RAL790), and C (RAL707) are color-coded cyan, orange, and red, respectively.

Noise-driven models of spontaneous walking could reproduce the time-course of *Drosophila* odor-evoked walking remarkably well (Fig. 5.10b). Successful models produced an increase in

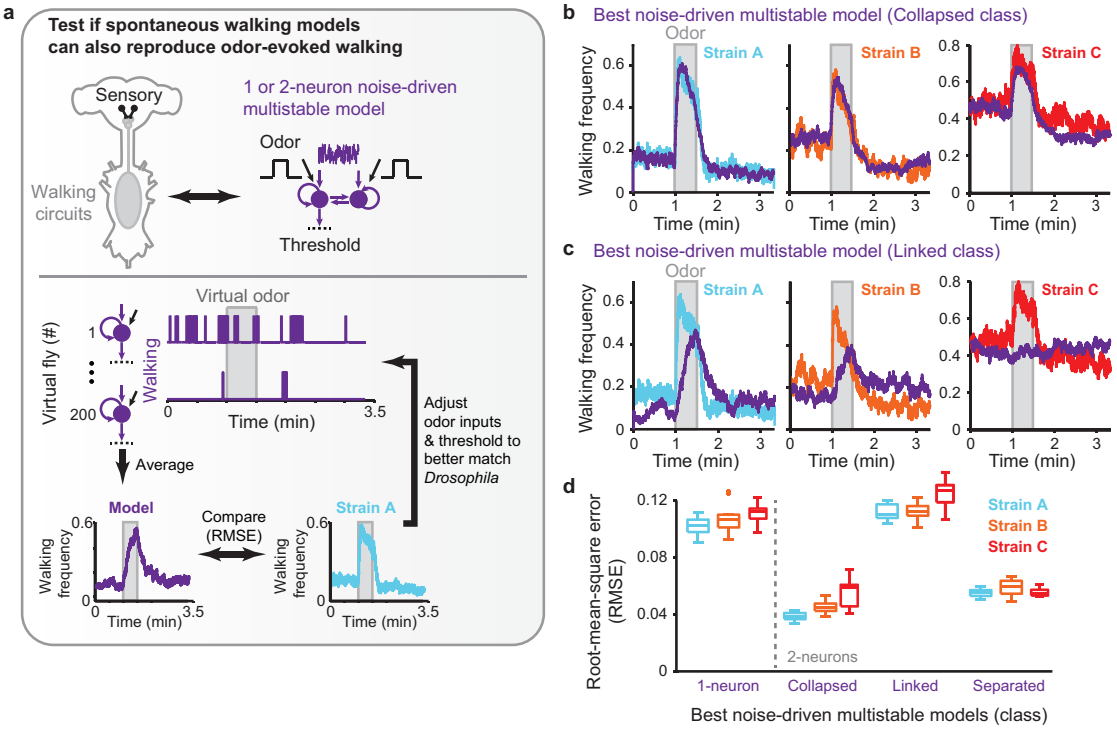


Figure 5.10: Circuit models for spontaneous behavior also reproduce odor-evoked walking dynamics. (a) Odor inputs were added to the best model for each noise-driven multistable 1 or 2-neuron class. The strength of odor inputs and the output threshold were adjusted iteratively to reduce the Root-Mean-Square Error (RMSE) between the model's odor-response dynamics averaged across 200 virtual flies and a *Drosophila* strain's odor-response dynamics averaged across 198 flies (e.g., Strain A, cyan). (b-c) Odor-response dynamics of the best 'Collapsed' (b) or 'Linked' (c) noise-driven 2-neuron model (purple) matching odor-response dynamics for three different *Drosophila* DGRP strains A (RAL57), B (RAL790), and C (RAL707). Traces for each strain are color-coded cyan, orange, and red, respectively. (d) Root-Mean-Square Error (RMSE) between odor-response dynamics for the best circuit model of each multistable class and odor-response dynamics for *Drosophila* Strains A, B, and C (cyan, orange, and red boxplots, respectively). The dashed line separates 1-neuron and 2-neuron model classes. N = 5 comparisons each.

walking during odor presentation and decay in walking frequency after odor removal that settled to a reduced rate of spontaneous walking. Importantly, some models failed to replicate odor-response dynamics (Fig. 5.10c; Root-Mean-Square Error or RMSE > 0.08) but the ability of a given model to match odor responses was consistent across all three *Drosophila* strains (Fig. 5.10d). These data support the hypothesis that common neural circuits orchestrate both *Drosophila* spontaneous and sensory-evoked walking and show that specific noise-driven multistable models capture their dynamics.

5.4.3 Noise creates a circuit memory of odor-evoked dynamics

We next asked how noise shapes the dynamics of odor-evoked walking in our models. To do this we examined each period of the odor-impulse experiment (Fig. 5.11a) for our most successful circuit model (Fig. 5.15b, 'Best Collapsed'). Although odor stimulation initially increased the output neuron's neural activity and the average walking frequency across a virtual fly population (Fig. 5.11b, 'Odor impulse, white arrowhead), shortly afterwards walking frequency began to decay. This was due to an odor-evoked change in the circuit model's dynamics: odor stimulation caused the appearance of a single stable equilibrium point below the output threshold for walking (Fig. 5.11c, 'Odor impulse', cyan circle) to which neural activity levels were attracted (Fig. 5.11b, 'Odor impulse', red arrowhead).

When the odor was removed, the circuit model's dynamics returned to that for spontaneous walking (Fig. 5.11c, 'Decay'). However, the behavior of virtual flies did not immediately match the pre-odor spontaneous walking frequency. Instead there was a prolonged decay in walking frequency (Fig. 5.11a, cyan). Close analysis of this post-odor period reveals that for some virtual flies output neural activity remained as high as during odor stimulation (Fig. 5.11b, 'Decay', white arrowhead) while for other virtual flies, output neural activity remained near the odor-induced low stable equilibrium level (Fig. 5.11b, 'Decay', red arrowhead). Over time, walking virtual flies became sedentary as output neural activity tended towards the two stable equilibrium levels (Fig. 5.11c, 'Reduced spontaneous walking'). However, there remained a subpopulation of flies whose output neural activity remained below the threshold for walking (Fig. 5.11b, 'Reduced spontaneous walking', red arrowhead), resulting in a reduced post-odor spontaneous walking frequency (Fig. 5.11a, magenta).

In our models, noise was largely responsible for these delayed changes in neural activity following odor removal. Rather than returning rapidly to stable equilibrium levels (Fig. 5.11d, 0% noise, 'Individual circuit activity'), noise caused neural activity to randomly diffuse more slowly to these levels (Fig. 5.11d, 100% noise, 'Individual circuit activity'). This in turn made individual walking less predictable and less stimulus-locked than would otherwise be expected (Fig. 5.11d, compare 0% and 100% noise, 'Individual walking'). When averaged across many individuals, the walking frequency of virtual flies therefore exhibited complex dynamics including pre-odor spontaneous walking, post-odor walking decay, and reduced spontaneous walking after odor stimulation (Fig. 5.11d, 100% noise, 'Population average'; Fig. S8a). Counter-intuitively, while noise increased behavioral variability across individuals, the time-course of sensory responses across virtual fly populations was remarkably consistent across different initial conditions and using different noise patterns (Fig. 5.12b). Therefore, the reproducibility of walking that we observed for *Drosophila* populations of the same strain (Fig. 5.3d-e) does not preclude shaping by neural noise.

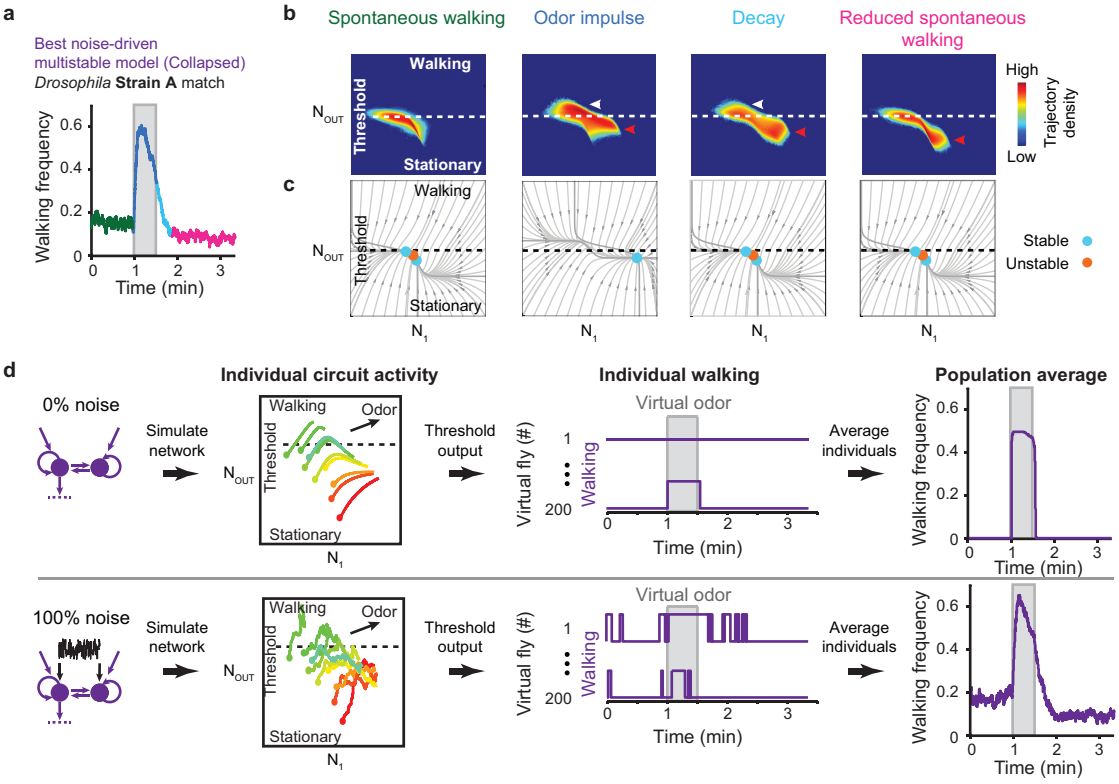


Figure 5.11: Noise creates a circuit memory of odor-evoked dynamics. (a) Odor-response dynamics for the best noise-driven multistable model ('Collapsed' class) matching Strain A from Figure 5.10b (see Fig. 5.16 for Strains B and C). Pre-odor spontaneous walking (green), odor impulse (blue), walking decay (cyan) and post-odor reduced spontaneous walking (magenta) periods are color-coded. (b) Trajectory density plots and (c) state-space diagrams for this model during each time period in panel a. In all trajectory density diagrams, arrowheads highlight frequent neural activity levels not observed during pre-odor spontaneous walking that are either above (white) or below (red) the threshold for walking. (d) Circuit activity, individual and population-averaged walking patterns in the absence (top, '0% noise') or presence (bottom, '100% noise') of neural noise. 'Individual circuit activity' plots show neural activity trajectories during odor stimulation starting from ten color-coded initial conditions. A solid circle indicates the start point for each trajectory. 'Individual walking' plots show walking for two representative virtual flies in the absence (top) or presence (bottom) of neural noise. 'Population average' shows the average walking across 200 virtual flies in the absence (top) or presence (bottom) of neural noise.

5.4.4 A circuit output threshold determines behavioral sensitivity to neural noise

This observed role of noise in spontaneous and sensory-evoked walking suggests that the effects of stochastic neural activity might be tuned to modify behavior. In support of this possibility, we observed a large range of spontaneous walking frequencies across 98 genetically distinct *Drosophila* strains (Fig. 5.11a). We wondered if these differences might arise from

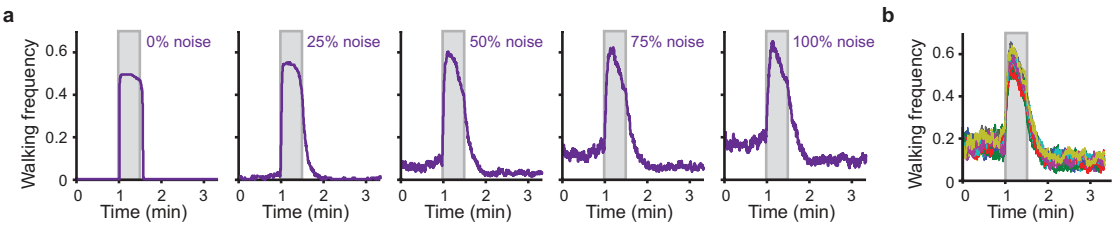


Figure 5.12: The effects of noise and reproducibility of odor-evoked walking dynamics in the best circuit model. (a) Odor-evoked walking dynamics for the best circuit model ('Collapsed' class) with noise amplitudes varying between 0% (no noise) to 100% (standard) noise. A grey box represents the period of odor presentation. (b) Odor-evoked walking dynamics across 20 repetitions of the 100% noise model in panel a. In each trace different initial conditions and noise time-series were used. Each experimental replicate is color-coded.

differences in the levels of noise or the output threshold for walking circuits. To test this, we measured the effects on spontaneous walking of varying the amplitude of noise or the output threshold for our best circuit model (Fig. 5.13a, Collapsed class). Reducing the amplitude of noise lowered spontaneous walking frequency since neural activity was predominantly below the circuit output threshold. However, increasing the noise strength did not substantially increase walking frequency (Fig. 5.13a-b, blue). By contrast, varying the circuit model's output threshold – biologically equivalent to the sensitivity of downstream CPGs to walking circuit activity - could account for the full range of spontaneous walking frequencies observed across all 98 strains (Fig. 5.13a-b, red).

Interestingly, beyond simply changing the level of spontaneous walking, shifting the model's output threshold also had a strong effect on the post-odor reduction in spontaneous walking frequency (Fig. 5.13c-d). A low output threshold resulted in high levels of pre-odor spontaneous walking and considerably reduced post-odor spontaneous walking. By contrast, when the output threshold was high, spontaneous walking was infrequent and almost no reduction in post-odor spontaneous walking was observed. These data suggest that the circuit output threshold can tune the sensitivity of walking dynamics to the effects of neural noise (Fig. 5.11d).

As an additional validation of our model, we tested this prediction in *Drosophila* . To do this, we first sought a molecular component that could regulate walking circuit output threshold by genome-wide association study of the behavioral phenotypes from our 98 inbred, sequenced strains (Mackay et al., 2009). Many different distributions of spontaneous walking across a population of flies can give rise to the same average spontaneous walking frequency. Therefore, we studied these population distributions directly. We examined the correlation between spontaneous walking distribution statistics (e.g., 25th percentile, median, 75th percentile) and naturally occurring genetic variations across the entire genome of all inbred strains. We identified a Single Nucleotide Polymorphism (SNP) with a significant correlation to the spontaneous walking frequency of the most active flies (97.5 percentile) of each strain (Fig.

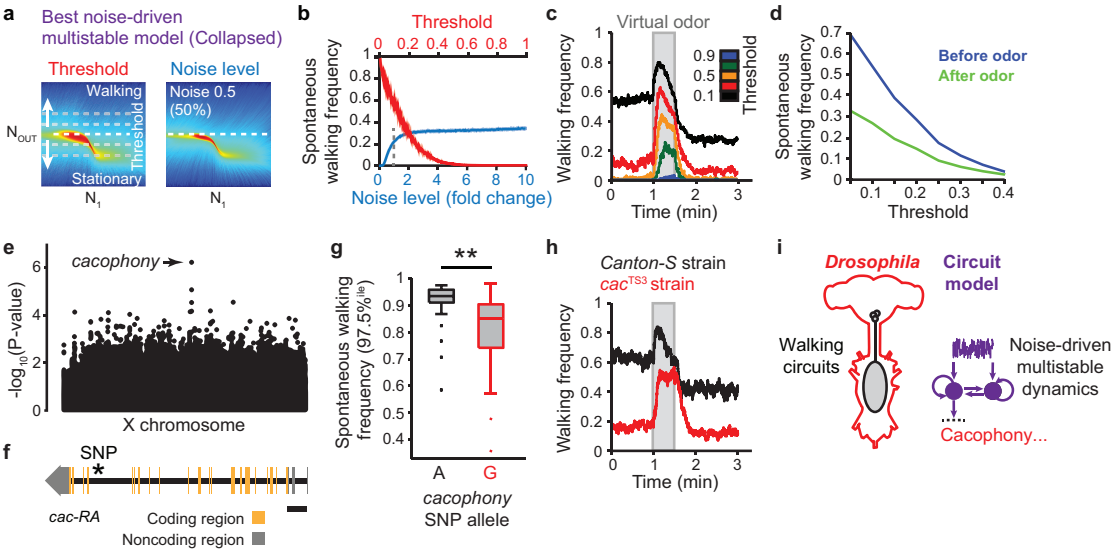


Figure 5.13: A circuit output threshold determines behavioral sensitivity to neural noise. (a) A schematic of varying the output threshold (left) or the noise level (right) of the best noise-driven multistable circuit model ('Collapsed' class). A white dashed line indicates the model's output threshold. Trajectory density plots of the model with 100% noise and variable thresholds (left) or 50% noise and the original threshold (right) are shown. (b) Spontaneous walking frequency for the best 2-neuron noise-driven multistable model as a function of output threshold level (red) or noise amplitude (blue, up to 10x original level). Shown are the mean (solid line) and standard deviation (transparency) of 10 experiments measuring the mean pre-odor spontaneous walking frequency for a group of 100 flies. (c) Odor-evoked walking dynamics for the best model with output thresholds ranging from 0.1 (black) to 0.9 (blue). (d) Mean spontaneous walking frequencies before (blue) or after odor presentation (green) for the model as a function of output threshold. (e) Manhattan plot for a genome-wide association study of spontaneous walking frequency using 98 inbred DGRP fly strains. Each point represents a single single-nucleotide polymorphism (SNP). An arrow indicates a SNP in the cacophony gene. (f) A schematic of the cacophony gene, cac-RA variant: one of 15 potential splice forms. The identified SNP (asterisk) is within an intron of the gene. Scale bar is 4 kb. (g) The 97.5 percentile of spontaneous walking across 98 DGRP inbred strains with either an 'A' or 'G' allele at the cac SNP locus. ** P<0.01 for a Mann-Whitney U-test. (h) Odor-evoked walking dynamics for the *Canton-S* strain of flies (black) or the cac^{TS3} mutant strain of flies (red). (i) Noise-driven multistable circuit models best reproduce *Drosophila* circuits for walking. Cacophony and other proteins may be biological representations of the model's output threshold.

5.13e). This SNP was located within an intron of the *cacophony* (*cac*) gene (Fig. 5.13f) and fly strains with the 'G' allele had a markedly lower level of basal walking than those with an 'A' allele (Fig. 5.13g, Kruskal-Wallis test, $P < 2 \times 10^{-7}$; Permutation test of 10,000 trials False Discovery Rate = 0%). *Cacophony* is a plausible biological output threshold for walking circuits since it is a presynaptic voltage-gated calcium channel subunit (Smith et al., 1996) that is

present in the adult *Drosophila* central nervous system where it has been shown to regulate synaptic transmission (Kawasaki et al., 2000; Gu et al., 2008).

Loss-of-function mutations in *cac* would be expected to impair neural transmission (Rieckhof et al., 2003) between *Drosophila* walking circuits and downstream CPGs: an equivalent to a high output threshold in our model. Thus our model predicted that while wild-type animals should show a higher pre-odor spontaneous walking frequency with a pronounced depression of spontaneous walking frequency following odor stimulation, *cac* mutant animals should exhibit a relatively low pre-odor spontaneous walking frequency and a smaller depression of post-odor spontaneous walking. Remarkably, we confirmed these predictions: *Canton-S* strain control flies had high pre-odor spontaneous walking and greatly reduced post-odor spontaneous walking, while *cacophony* mutant animals (cac^{TS3}, in the same *Canton-S* genetic background) had low pre-odor spontaneous walking and a similar post-odor spontaneous walking frequency (Fig. 5.13h).

5.5 Discussion

Despite the long-appreciated presence of stochastic activity in the nervous system (Fatt and Katz, 1952), how neural noise influences animal behavior has remained unknown. Here, we have demonstrated a role for noise in *Drosophila* behavior by pairing high-resolution behavioral measurements with unbiased computational discovery to identify circuit models that faithfully reproduce *Drosophila* spontaneous walking patterns. Importantly, these models require noise. During spontaneous walking, noise allows the neural activity of circuit models to switch between high and low stable levels that lead to walking and stationary behaviors, respectively. During odor-evoked walking averaged across populations of flies, noise slows the tendency of neural activity to reach equilibrium levels, resulting in a prolonged circuit memory of odor-evoked shifts in neural dynamics.

Across all of our discovered noise-driven circuit models, we observed that those with multistable dynamics most effectively reproduced the bursty nature of *Drosophila* spontaneous walking. When the activity level of the model's output neuron fluctuated around a high, Up-state it generated bursts of walking. Conversely, when the output neuron's activity level fluctuated around a low, Down-state it produced long sedentary periods. Multistable neural activity has been observed in biological contexts, for example in *Drosophila* visual neurons (Maimon et al., 2010) and in vertebrate neostriatal and neocortical neurons (Cossart et al., 2003; Wilson and Groves, 1981).

Our noise-driven multistable circuit models, although abstracted from biology, can serve to guide the study of walking circuits in the relatively small nervous system of *Drosophila*. One might systematically identify walking circuit neurons by finding those whose activity correlates with the time course of spontaneous tethered walking (Seelig et al., 2010; Portugues et al., 2014). Our models make several predictions about the dynamics of these in vivo walking circuits. First, their firing rates should switch between high and low stable levels of activity. Second, the rate

of switching should depend in part on the amplitude of neural noise. Finally, the frequency of spontaneous walking and the degree to which odor-evoked walking is stimulus-locked should depend on circuit noise and the efficacy of circuit output.

In addition to identifying relevant circuits in the *Drosophila* brain, it is also interesting to understand how noise-driven circuit function can be tuned to shape behavior. Taking advantage of natural phenotypic variation, we identified a role for the voltage-gated calcium channel subunit, *Cacophony*, in spontaneous walking frequency. An intronic SNP in the *cac* gene correlates with the frequency of spontaneous walking, suggesting a regulatory change in ion channel expression. However, the complexity of the locus and lack of knowledge of relevant cis-regulatory sequences preclude facile determination of the influence of this polymorphism. Still, as for the lobster stomatogastric ganglion (Marder, 2011), tuning circuit dynamics through ion channel expression patterns may be a simpler alternative to developmental neural circuit 'rewiring' (Ramdya and Engert, 2008; Ramdya and Benton, 2010) for regulating animal behavior.

In conclusion, we have shown that *Drosophila* walking patterns are best explained by multistable circuit dynamics driven by neural noise. We predict that our implication of noise in the time-course of *Drosophila* walking reflects a widespread influence of stochastic neural activity on animal behavior. These results also highlight how coupling large-scale biological data with unbiased computational discovery can provide an important complement to anatomical, physiological, and genetic studies for understanding how neural circuits orchestrate behavior.

5.6 Methods

5.6.1 *Drosophila* strains

Drosophila Canton-S strains were used in odor-impulse (Figs. 5.1, 5.13h) and spontaneous walking experiments (Fig. 5.4b). *Drosophila* Genetic Reference Panel (DGRP) 25 strains were used in odor-impulse experiments (Figs. 5.3 and 5.10) and the genome-wide association study (Fig. 5.13e). cac^{TS3} mutant animals were used for testing the role of *cacophony* in spontaneous walking (Fig. 5.13h).

5.6.2 *Drosophila* behavior apparatus

Experimental arenas (50 mm × 10 mm enclosures with a height of 1.3 mm (Fig. 5.1a-b)) were designed using the 3D CAD software, SolidWorks (Dassault Systemes, Waltham, Massachusetts, USA) and CNC machined from polyoxymethylene and acrylic glass. To backlight the arenas, we used a white LED panel (Lumitronix, LED-Technik GmbH, Hechingen) filtered with far-red semitransparent film (Eastman Kodak, Rochester, NY USA), a color for which fruit flies are visually insensitive. For olfactory stimulation, air (Messer Schweiz AG, Lenzburg, Switzerland) was bubbled through either water or 10% acetic acid and controlled using Mass Flow

controllers (PKM SA, www.pkmsa.ch) at a regulated flow rate of 500 mL/min via computer controlled solenoid valves (The Lee Company, Westbrook, CT, USA). A custom-fabricated circuit board and software 20 (sQuid, http://lis.epfl.squid/) simultaneously controlled valves and acquisition cameras (Allied Vision Technologies, Stadtroda, Germany). We measured the flow of odor using a miniPID (Aurora Scientific Inc. Aurora, Ontario, Canada).

5.6.3 *Drosophila* behavior experiments

We performed experiments on adult female *Drosophila* raised at 25 degC on a 12h light:12h dark cycle at 2-5 days post-eclosion. Experiments occurred either the morning or late afternoon Zeitgeber Time. Prior to experiments, flies were starved for 4-6 hours in humidified 25 degC incubators. For odor stimulation experiments, we measured the walking behaviors of between 131 and 242 flies (median 205 flies). 98 DGRP strains were screened over the course of approximately 1 year. To minimize the effects of weekly and seasonal variation, we randomly selected and simultaneously screened groups of $\tilde{2}0$ strains at a time. We repeated measurements for a single strain (RAL208) four times over the course of the screen to confirm season-independent behavioral reproducibility.

For spontaneous walking behavior experiments, we recorded ten *Canton-S* strain flies for 30 min each, 5 h in total in a temperature-controlled room at 25 degC under low red light illumination without airflow. During the odor pulse experiment, flies were first exposed to air throughout the arena for 1 min, then 10% acetic acid for 30 s, and finally, air for 2 min. During the odor aversion experiment, 10% acetic acid was present on one side of the arena for 30s and air on the other. This pattern alternated for an additional three cycles (Fig. 5.1c, 'Odor aversion').

5.6.4 *Drosophila* behavioral analysis

Following each experiment, we measured each fly's position over time using Ctrax and Matlab (The Mathworks, Natick, Massachusetts, USA) Behavioral Microarray software scripts (Branson et al., 2009). Afterwards we discretized the speed of a fly into a binary time-series using a hysteresis threshold. Based on previous studies (Martin et al., 1999; Sorribes et al., 2011; Valente et al., 2007; Wolf et al., 2002) and confirmed by our own measurements, we considered a fly to have begun walking when its speed exceeded 1 mm/s. For walking flies, we considered walking to have terminated when the speed decreased below 0.5 mm/s (a conservative value chosen to reduce the effects of measurement noise). We could thus classify speed in a binary fashion: walking or stationary (Fig. 5.1d). When averaged over a population of flies, we obtained a 'Walking frequency': the proportion of active flies at a given time point ranging from 0 when no flies are walking, to 1 when all flies are walking (Fig. 5.1f).

To calculate the correlation between spontaneous walking frequencies for genetically identical groups of flies, we randomly sampled two populations (groups A and B) of 65 flies (0.5*

the minimum population size) from the same strain. We then compared the mean walking frequency during the first minute of the odor-impulse experiment for each of these groups. We performed these measurements 100 times per strain (Fig. 5.3d).

To calculate the correlation between odor-response time-courses for fly strains, we randomly sampled two populations (groups A and B) of 65 flies (0.5* the minimum population size) from each strain. Odor-impulse traces (58th – 200th s of the odor impulse experiment) were then normalized between 0 and 1. Comparisons were performed either between groups from the same strain or between groups from different strains. Each comparison was performed 100 times and the mean R2 value is shown (Fig. 5.3e). To calculate odor aversion, for each fly we measured the proportion of time spent in the air zone minus the time spent in the odor zone over the course of the odor aversion experiment. This was divided by the total time of the aversion experiment yielding a value between -1 (always in the odor) and 1 (never in the odor) (Fig. 5.3f-g).

5.6.5 Dendrogram generation

We generated a dendrogram representation of the correlation between odor-response time-series across all 98 DGRP *Drosophila* strains (Fig. 5.9). The length of each branch represents the correlation between the odor-response time-series of two inbred strains of flies. For subsequent circuit model matching we selected at random one strain from each of the following correlation intervals: $\rho \leq 0.9$, $0.9 \leq \rho leq 0.95$, $\rho > 0.95$ (Fig. 5.10).

5.6.6 Genome Wide Association Study

Genome wide association analyses were performed using a non-parametric Kruskal-Wallis test. In brief, we created a list of genetic loci with two or more alleles in the population with a minimum allele count of 3 strains. We then grouped each spontaneous walking trait (e.g., 97.5th percentile of spontaneous walking distributions) according to the allele of its strain and performed a Kruskal-Wallis test. For each genetic locus, 1000 permutations of the phenotype data were performed to generate an initial estimate of the False Discovery Rate (FDR). Subsequently, for our candidate locus (*cacophony*, X chromosome, position 11836142) we performed a second FDR analysis using 10,000 permutations of the Kruskal-Wallis test.

5.6.7 Neural circuit modeling framework

For neural circuit models we used Continuous Time Recurrent Neural Networks (CTRNNs). This modeling framework was chosen for its ability to mimic biological neural circuits 45. A Continuous Time Recurrent Neural Network (CTRNN) model with M neurons $N_1, N_2, \ldots, N_M$

is defined by a system of ordinary differential equations (ODE):

$$\frac{dx_i}{dt} = F_i(t, x_1, x_2, \ldots, x_N) = \frac{1}{\tau}\left(-x_i + \sum_{j=1}^{M} w_{ji} \cdot \sigma(x_j + b_j) + I_i\right), i = 1, \ldots, M \tag{5.1}$$

The state of a neuron N_i is defined by the variable x_i and is updated by an increment dx_i inversely proportional to the time constant $\tau_i \in [0.05, 50]$. The output of a neuron N_i is obtained by evaluating a sigmoid transfer function $\sigma(x) = \frac{1}{1+e^{-x}}$ on the state x_i added to a constant bias $b_i \in [-10, 10]$. Neurons N_j and N_i are connected with synaptic links of weight $w_j i \in [-20, 20]$. Furthermore, each neuron can receive an optional input I_i (e.g., odor input). We tested models up to five neurons in size since even three neurons are sufficient to exhibit a wide variety of dynamical behaviors (including chaos) (Beer, 1995).

To investigate the capacity of noise alone to match *Drosophila* spontaneous walking, we optimized a threshold ranging from -4σ to 4σ directly upon a Gaussian noise source representing stochastic activity in biological circuits (Faisal et al., 2008; Destexhe and Rudolph-Lilith, 2012). For noiseless circuit models, the input is set to zero ($I_i = 0$). For noise-driven models, each neuron receives Gaussian noise with standard deviation $w_{NOISE,i}(I_i = w_{NOISE,i} \cdot G$, where $G\,\mathcal{N}(0, 1)$ follows a Normal distribution).

CTRNNs were simulated using a custom high-performance C++ implementation. Our implementation used an approximation of the sigmoid function $\sigma(x)$ (Schraudolph, 1999) to speed-up simulations. Furthermore, to decrease the computational load of the simulations the noise value G was generated every $T_{NOISE} \in [0.01, 1]$ seconds. For intermediate time-steps the noise value G was interpolated. Although this introduced correlations in the noise, the time-scale at which the noise value changed was orders of magnitude smaller than the time-scale at which the slowest dynamics happened (hundreds of seconds). The ODEs regulating the evolution of the CTRNN were integrated using ODEINT (Ahnert and Mulansky, 2011), a publicly available solver for ODE. We used a Runge-Kutta 4th-order method at a constant integration time-step of 10 ms (five times smaller than the smallest time constant of a neuron) to integrate the system of ODE.

During the simulation of a circuit model, the trajectory of a model's neural activity evolves over time from an initial condition, represented by the neuron states $x_i(t_0)$, to eventually reside within the dynamical regime of the model (an equilibrium point, a limit cycle, etc.). In our experiments, we took two precautions to discard the long transients that sometimes occurred as trajectories passed from their initial positions into the model's dynamical regime. First, we identified the equilibrium points of the model (finding $\frac{dx_i}{dt} = 0$). Then we generated initial conditions in the neighborhood of identified equilibrium points by sampling from a multivariate Gaussian distribution having an identity covariance matrix centered at the equilibrium points. Second, at the beginning of each simulation, we integrated the model for

5 minutes of real time (3×10^4 time-steps) to discard dynamics during transit from the initial condition.

To generate a binary time-series equivalent to walking and stationary periods for *Drosophila*, we applied a threshold to the output of a neuron, arbitrarily chosen to be N_0 (referred to in the text as N_{OUT}), with a value $THR \in (0,1)$. Whenever the output of this neuron was greater than the threshold (), the virtual fly was walking and otherwise it was stationary.

5.6.8 Circuit model parameter optimization

We optimized circuit model parameters using a stochastic optimization method for tuning model parameters in an iterative manner (Clerc, 2010). First, we generated a population of circuit models of a given size (e.g., three neurons). Next, we measured the activity of these models and transformed these into binary time-series comprising walking and stationary periods using a threshold. Finally, walking and stationary period durations were measured and aggregated into weighted variable bin-width histograms for walking or stationary intervals. Bin-widths were derived from the *Drosophila* target dataset. We compared these histograms to target histograms measured from *Canton-S* flies. After assessing this population of circuit models, model parameters were adjusted towards those of the best performing models in this and previous iterations. This process was repeated until model performance converged. We then studied the topological and dynamical properties of the best circuit models found.

In more detail, the $N_p = w_{ij}, \tau_i, b_i, w_{NOISE,i}, THR, T_{NOISE} | i,h \in 1,\ldots,M$ parameters of the CTRNN models were optimized using Particle Swarm Optimization (PSO) (Poli et al., 2007). We used standard parameter settings $c_1 = c_2 = 2$. The inertia parameter ω of the algorithm was modified during an optimization run, following an update rule $\omega(t) = 0.9 - \frac{0.7t}{T}$ to favor global search at the beginning of the optimization process and local search towards the end, where t is the current iteration and $T = 200$ is the maximum number of iterations. PSO operated concurrently on a set of $M = 50$ solutions. Therefore, a total of 10^4 solutions were evaluated during each optimization run. Each function evaluation required between 11 and 30 seconds of computational time. The optimization of the CTRNN models was performed on a cluster (http://hpc.epfl.ch), using two nodes with 48 cores AMD Opteron 6176 (Magny-Cours) 2.3 GHz and 192 GB of memory.

To optimize the odor input strength and output threshold of best models for matching *Drosophila* odor-evoked walking dynamics, we measured the activity of the circuit model's output during 60s of no stimulation (spontaneous walking), 30s of odor stimulation, and then 120s of no stimulation. We repeated this experiment while iteratively optimizing the few free parameters (odor input strength per neuron, and output threshold) to minimize the Root-Mean-Square Error (RMSE) between the target *Drosophila* odor-response time-series (average of $\tilde{2}00$ flies) and the model's odor-response time-series (average of 200 virtual flies). The code for Automatic Neural Circuit Discovery can be downloaded at: http://lis.epfl.ch/files/content/sites/lis/files/research/index.html

Cost function or 'Difference from *Drosophila* data' The cost function assigns a score to each model evaluating how well it captures *Drosophila* walking by comparing histograms generated by the model with the histograms generated by the *Canton-S* strain. The comparison of these histograms represents a crucial aspect of cost function design. While it is possible to use standard statistical tests such as distance measures between empirical cumulative distributions of data (for example the Kolmogorov-Smirnov test), these statistical tests can mislead the optimization process by assigning reduced importance to rare events. For distributions of time durations, it is evident that these approaches would fail, since rare events (e.g., long walking or stationary periods) would be effectively ignored when comparing distributions. Therefore, we generated "weighted" histograms in which each bin was weighted by the duration it represents. For example, 10 walking events of 1 s duration and 1 event of 10s duration, would classically be represented as two bins of different "height" (10 and 1 respectively). In our weighted histograms these two bins have the same height ($1\text{s} \cdot 10 = 10\text{s} \cdot 1$).

We also wanted to remove empty bins. Thus, we generated variable bin-width histograms. The boundaries of each bin for walking and stationary interval histograms were determined using *Drosophila Canton-S* spontaneous walking data (see Fig. 5.6a). We used the same bin boundaries when evaluating each model.

For each cost function evaluation, we simulated a model $K = 100$ times. The model was started from K different initial conditions and simulated for 60 minutes of real time. For each of the K simulations, we selected at random with equal probability either the first or second 30 minutes of simulation, to mitigate overfitting of model behavior to the same trajectory and to foster model unpredictability. Each simulation produced a binary time-series representing walking (1) or stationary (0) behavior in the virtual fly. Thus, we computed the histogram for walking and for stationary periods using the data from all the selected K chunks. The generated histograms $H_{S,W}$ for walking and $H_{S,I}$ for stationary were compared respectively to the target *Drosophila* histograms $H_{T,W}$ and $H_{T,I}$ obtaining the distance between the histograms $d_W = d(H_{T,W}, H_{S,W})$ and $d_I = d(H_{T,I}, H_{S,I})$.

The distance measure between a target and simulated histogram is defined as:

$$d(H_T, H_S) = \sum_{i=1}^{B} |R \cdot h_S(i) - h_T(i)| t_B(i) \tag{5.2}$$

B is the number of bins in the histograms, $h_S(i)$ and $h_T(i)$ returns the count for bin i in the synthetic H_S and target histogram H_T and $t_B(i)$ returns the interval duration represented by bin i, here corresponding to the lower boundary of the bin. The scale factor R reconciles data obtained from simulations to available fly data. In our experiments we tested $K = 100$ simulated initial conditions per cost function evaluation, therefore $R = 0.1$ as we used data from 10 real flies.

The cost function f maps a model m to a cost function value in $[0,\infty]$, For the sake of simplicity, we presented a normalized cost function value F. F is obtained by normalizing the cost function using the value F_{NORM} that a virtual fly would have if always walking or always stationary such that $F_(m) = \frac{F(m)}{F_{NORM}} = \frac{d_W+d_I}{F_{NORM}}$. A value of 0 corresponds to a perfect match, a value of 1 to the same score an always walking or always stationary virtual fly would get. Intermediate values ranging between 0 and 1 correspond to plausible distributions. Even higher values generally represent circuit models with periodic dynamics at very high frequencies.

To produce an intuitive scale for the cost function, we evaluated the cost function resulting from comparing subsets of *Drosophila* data with the full dataset (Fig. 5.14a). We generated subsets of data by selecting at random the desired number of flies F and replicating the data from each selected fly 10/F times, rounded to the closest larger integer. In cases where we generated too much data (F is not a divisor of 10), we randomly removed walking or stationary events until we obtained a dataset with the same length as the full *Drosophila* spontaneous walking dataset.

5.6.9 Variable bin-width weighted histogram generation

Given a vector v containing walking or stationary interval durations for 5 h of *Canton-S* fly spontaneous walking, we computed the boundaries of the variable bin-width weighted histograms. This routine took as inputs the minimum resolution r of a bin (the minimum separation between boundaries) and the minimum count c of events to generate a bin. Next it tried to generate histogram boundaries by recursively splitting the initial single-bin boundaries $[0, max(v)]$ into smaller bins containing a minimum of c events and having minimum duration of r seconds (Fig. 5.6a).

We applied this procedure to get the boundaries of the bins of the histograms of walking and stationary period durations of *Drosophila*. We then used these same boundaries to compute both real ($H_{T,W}$ and $H_{T,I}$) and simulated histograms ($H_{S,W}$ and $H_{S,I}$).

5.6.10 Dynamical systems stability analysis

Since a variety of diverse circuit model topologies can generate identical output we classified models not only by their topologies but also by their dynamics (Strogatz, 2001; Funahashi and Nakamura, 1993; Tsuda, 2001). In this manner the behavior of a model with n neurons can be understood by observing the time evolution of its trajectory through an n-dimensional neural activity state-space. By studying the unfolding of state-space trajectories, one can identify common behavioral motifs among circuit models with widely different parameters. Using this formalism, features in a state-space landscape (e.g., attractors, limit cycles, and deterministic chaos) provide a clear language with which to interpret and compare different circuit models (Strogatz, 2001). This dynamical systems perspective has been successful in classifying both artificial neural networks and biological neural populations (Mante et al., 2013).

To analyze the global dynamical behavior of our circuit models and to classify closely related ones, we performed stability analysis on our models in the absence of noise in the following manner. First, the m equilibrium points $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_M$ of the CTRNN were identified by numerically finding the roots of the system of differential equations $F(\bar{\boldsymbol{x}})$ using the multi-dimensional root finder provided by the Gnu Scientific Library (http://www.gnu.org/software/gsl/). The Jacobian matrix J of a CTRNN is defined as:

$$J(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial F_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial F_1(\boldsymbol{x})}{\partial x_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_M(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial F_M(\boldsymbol{x})}{\partial x_M} \end{pmatrix} \tag{5.3}$$

where

$$\frac{\partial F_i(\boldsymbol{x})}{\partial x_j} = \begin{cases} \frac{w_{ji}}{\tau_i} \frac{e^{x_j+b_j}}{\left(1+e^{x_j+b_j}\right)^2} & \text{if } i \neq j \\ \frac{-1}{\tau_i} + \frac{w_{ji}}{\tau_i} \frac{e^{x_j+b_j}}{\left(1+e^{x_j+b_j}\right)^2} & \text{if } i = j \end{cases} \tag{5.4}$$

We studied the stability of the CTRNN by linearizing the system in the neighborhood of each equilibrium point and computing the eigenvalues of the Jacobian matrix J of the CTRNN for each equilibrium point $\bar{\boldsymbol{x}}$ by solving $det(J(\bar{\boldsymbol{x}}) - \lambda I) = 0$. For a classification of stability of state-spaces given the equilibrium points eigenvalues refer to (Strogatz, 2001).

5.6.11 Trajectory density maps

Density maps for circuit model trajectories were obtained by discretizing a plane described by two neuron states (x_i, x_j) into a grid of $10^3 \times 10^3$ cells ranging over the state values $[-50, 50]$. Then we counted how many times a trajectory (its projection onto (x_i, x_j)) enters each cell. Density maps were generated by initializing the models from 10^4 random initial conditions. The color of the density plot is related to the logarithm of the probability of a neural activity trajectory to step into each cell.

5.6.12 Testing the role of noise and threshold on spontaneous walking frequency

We assessed the effect of output threshold variation on spontaneous walking frequency by varying the threshold in $[0, 1]$ in increments of 0.001, evaluating our models for 30 minutes, and then measuring the spontaneous walking frequency. To assess the effect of noise amplitude, we varied the noise amplitude in [0, 10] times the original amplitude in increments of 0.05. In both experiments, we tested each model 10 times and averaged the observed spontaneous

walking frequency.

5.6.13 2-neuron multistable circuit model classification

To quantify differences in the dynamical behavior of 2-neuron noise-driven circuit models and to classify them, we generated 1000 initial conditions around each stable equilibrium point and let the trajectories evolve for 30 minutes. We then counted how many times a trajectory switched from one equilibrium point to the other. Trajectories of Collapsed Multistable circuit models switched many times during each 30-minutes period while Linked Multistable circuit model switched more rarely. Trajectories of Separated Multistable circuit models did not switch equilibrium points.

5.6.14 Lyapunov exponent computation

We computed Lyapunov exponents for noiseless circuit models by integrating the variational equations $\frac{d\delta}{dt}$ of the CTRNN together with the original system.

$$\frac{d\delta}{dt} = \begin{pmatrix} \frac{d\delta_{11}}{dt} & \dots & \frac{d\delta_{1M}}{dt} \\ \vdots & \ddots & \vdots \\ \frac{d\delta_{M1}}{dt} & \dots & \frac{d\delta_{MM}}{dt} \end{pmatrix} = J \begin{pmatrix} \delta_{11} & \dots & \delta_{1M} \\ \vdots & \ddots & \vdots \\ \delta_{M1} & \dots & \delta_{MM} \end{pmatrix} \tag{5.5}$$

Following a standard procedure (Benettin et al., 1980), we integrated the original system together with the variational equations for 1000 time-steps. Then, we orthonormalized the perturbations using the Gram-Schmidt algorithm and computed the full spectrum of M Lyapunov exponents $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_M$. The Kaplan-Yorke dimension (Kaplan and Yorke, 1979) was then computed as $D_{KY} = k + \sum_{i=1}^{k} \frac{\lambda_i}{|\lambda_{k+1}|}$, where k is the largest integer such that $\sum i = 1^k \lambda_i \geq 0$.

5.7 Supporting Information

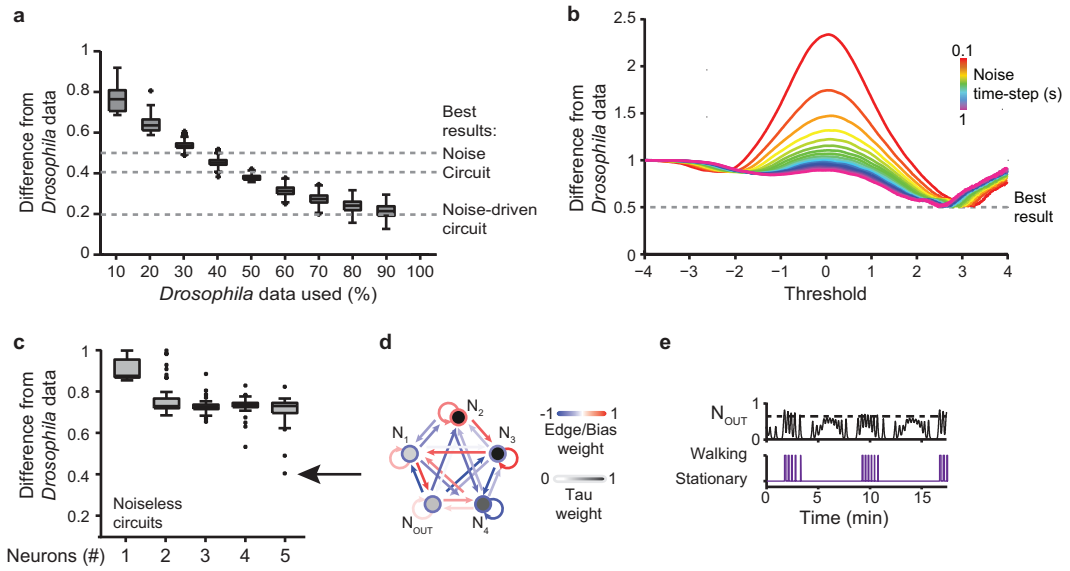


Figure 5.14: Matching of *Drosophila* spontaneous walking data using subsets of the data, noise alone, or noiseless circuit models. (a) *Canton-S* spontaneous walking data reproduced by time-normalized subsets of the same dataset. '*Drosophila* data used' indicates the percent of flies selected and time-normalized to allow comparison with the full 5-hour dataset of 10 flies. N = 1000 datasets per boxplot. Dashed lines indicate the best cost function value for noise alone ('Noise', cost function = 0.5), noiseless circuit models ('Circuit', cost function = 0.41), or noise-driven circuit models ('Noise-driven circuit', cost function = 0.19). (b) The ability of a threshold applied to a Gaussian noise source ($\mu = 0$, $\sigma = 1$) to reproduce *Drosophila* spontaneous walking data. The noise generation time-step (i.e., noise correlation) is color-coded. Each data point is the lowest/best cost function value for a given threshold on a given noise source. (c) The ability of noiseless circuit models to reproduce *Drosophila* spontaneous walking data. N = 50 circuit models for each size (1-5 neurons). A black arrow indicates the best noiseless circuit model found. (d) A graph representation of the best noiseless circuit model. Recurrent and reciprocal connection strengths are color-coded. The tau value for each neuron is shown in grey-scale. (e) Neural output activity (N_{OUT}) and walking for the model in panel d. This model exhibits chaotic behavior (Largest Lyapunov Exponent = 0.011).

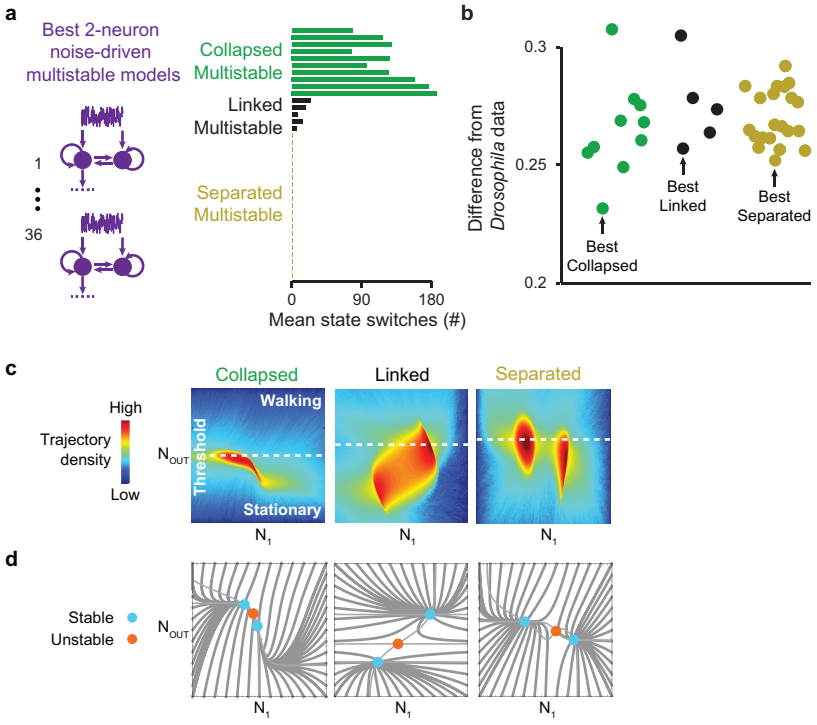


Figure 5.15: Classification of 2-neuron noise-driven multistable models. (a) The number of equilibrium points that neural activity trajectories visited over the course of 30 simulated minutes for 36 multistable best models. N = 1000 simulations per model. 'Collapsed' models visited each stable equilibrium point with high frequency. 'Linked' models visited each equilibrium point a few times. 'Separated' models visited one equilibrium point. (b) The cost function for each model sorted by class. The best model for each class is indicated (black arrow). (c) Neural activity trajectory density plots for the best model in each class indicated in panel b. The threshold between walking and stationary behavior is indicated (white dashed line). (d) State-space diagrams for the best model in each class indicated in panel b. Stable (cyan) and unstable (orange) equilibrium points are shown.

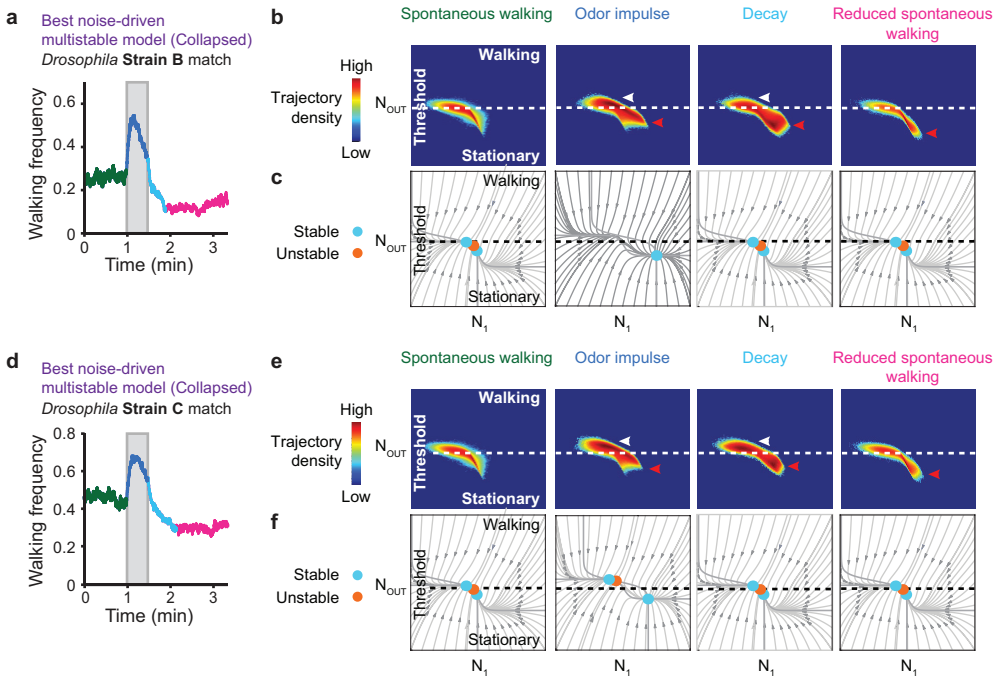


Figure 5.16: Noise creates a circuit memory of odor-evoked dynamics: Strains B & C. (a,d) Odor-response dynamics for the best 2-neuron noise-driven multistable model ('Collapsed' class) matching *Drosophila* DGRP Strains B (a) or C (d). Pre-odor spontaneous walking (green), odor impulse (blue), walking decay (cyan) and post-odor reduced spontaneous walking (magenta) periods are color-coded. (b,e) Trajectory densities (top) and (c,f) state-space diagrams (bottom) for this model during each period. In all trajectory density diagrams, arrowheads highlight frequent neural activity levels not observed during pre-odor spontaneous walking that are either above (white) or below (red) the threshold for walking.

Conclusions

5.8 Main contributions

This thesis has investigated potential advantages of the Viability Evolution paradigm, first introduced in (Mattiussi and Floreano, 2003). To this aim, we proposed several viability evolutionary algorithms. We began by sketching a first implementation of a viability-based evolutionary algorithm that used only eliminations and viability boundary adaptation to drive the search (See **Chapter 2**). We used this algorithm to test the hypothesis that a viability-based method could maintain higher diversity during evolutionary search with respect to a traditional competition-based method. We showed how the increased diversity eventually helps to discover more diverse solutions at the end of the evolutionary process. This result suggests that viability-based algorithms could be used for simulating open-ended scenarios, generating creative solutions and could help in avoiding convergence during evolution.

We then moved on to show a second advantage of designing evolutionary algorithms with viability principles. When viability boundaries are used to model different constraints or objectives, a greater amount of information is available during the search (see **Chapter 3**). In particular, we observed the amount of viable and non-viable offspring with respect to each viability criteria. We showed how this information can be used to enhance the performance of a state-of-the-art method for constrained optimization, by proposing a novel mechanism to adapt the step size. Furthermore, by incorporating viability boundaries the method is now compatible with starting the evolutionary search from infeasible solutions. The proposed method displayed competitive performance on unimodal constrained optimization problems.

Building on this, we proposed a novel evolutionary method for solving a broader class of constrained optimization problems (see **Chapter 4**). This algorithm, that we called mViE, maintains a population of distinct local search units based on the method presented in **Chapter 3** and recombines locally learned information using global search operators. By allocating the available function evaluations to local or global search depending on their previous performance, we could obtain a method that is also competitive on multimodal constrained optimization problems. This method has also been recently applied in a protein assembly prediction framework, briefly presented in **Appendix A**.

Finally, we applied evolutionary computation methods to a neuroscience investigation regard-

ing the role of neural noise in animal behavior (see **Chapter 5**). We developed a neural circuit discovery method that employed evolutionary algorithms for optimizing the parameters of neural networks. We then used it to search for neural circuit models capable of reproducing observed dynamics of spontaneous behavior in *Drosophila*. Our best discovered circuit models provided us unprecedented insights on the role of noise as a force shaping animal behaviors, even in the presence of sensory stimulation. Moreover, thanks to a multidisciplinary collaboration with neuroscientists and biologists, the prediction of our models were validated *in vivo*.

5.9 Future Directions

Given the variety of topics covered in this thesis several research directions can be pointed out for future work. Many of them were outlined in the discussion and conclusion sections of each chapter. Here we briefly highlight new ones.

In this thesis, we provided a first evaluation of the Viability Evolution paradigm. We specifically focused on the design of viability evolutionary algorithms for constrained optimization. Although constraints can be directly modelled as viability criteria, the paradigm does not easily lend itself to model multi-objective optimization problems, where conflicting objectives are present. In this case, tightening the viability boundaries defined on these conflicting objectives most probably leads to a stall of the search in a small area of the Pareto front. Apart from the trivial solution of using a competition function that returns a partial ordering (non-dominated sorting), thus reducing a viability algorithm back to a classical multi-objective method, it is unclear how to use tightening viability boundaries to drive the search in multi-objective problems to the optimal Pareto front. One possible solution could be to define a "virtual" viability boundary that approximates the Pareto front. This virtual boundary could be tightened to drive the search towards the optimal Pareto front. Once a convincing solution for dealing with multi-objective problems is found, the viability paradigm may represent a truly unifying abstraction for unconstrained, constrained and multi-objective optimization in evolutionary computation. This could lead to the development of a new class of methods that could automatically adapt to the problem at hand, relieving the EC user from selecting an appropriate (specific) algorithm for the problem being tackled.

The viability paradigm naturally incorporates the concept of dynamic population sizes. The viability algorithms presented in this thesis for constrained optimization used only fixed-size populations. Interesting avenues of research may consider exploring dynamic population-size viability algorithms, which may be especially useful in multimodal optimization scenarios.

Also, the method for constrained optimization presented in **Chapter 4**, mViE, can be improved in several ways. Currently, we exploited multiple independent local search units to tackle multimodal constrained problems. To improve its robustness on multimodal landscapes one could extend the algorithm used by each search unit (presented in **Chapter 3**) to generate

multiple solutions per iteration, conversely to the single offspring sampled at the moment[1]. However, the self-adaptation of this algorithm's parameters in the presence of constraints and multiple sampled offspring could be an arduous task.

Furthermore, all algorithms considered in this thesis used fairly simple update policies for changing the viability boundaries. Other improvements to the efficiency of the presented algorithms can derive from considering more advanced update rules for adapting the viability boundaries, taking into account other information extracted from the boundaries. Another idea that could be tested is the use of a second set of viability boundaries, defining the viability of the local search units. This may be useful to drive the whole population of local search units towards desired areas of search space. Thus, this method would use viability boundaries at two levels: at the level of the single search unit, to locally adapt to constraints, and at the level of the population of search units, to drive them towards feasible areas. The introduction of this second set of viability boundaries may as well have the effect of improving the diversity of the population of local search units (similarly to what showed in **Chapter 2**).

Finally, our method for constrained optimization was tested only on inequality constraints. The modelling of constraints as viability boundaries offers the advantage of being able to add or remove them dynamically during the search. This may represent a new way for tackling problems with equality constraints. The presence of equalities makes the search very difficult for an evolutionary method, given the very limited size of the feasible space, which often reduces to zero-volume regions. In many cases an evolutionary algorithm can converge to sub-optimal region of search space, due to the shape of an equality constraint which creates locally optimal areas. Temporarily removing the viability boundary defined on an equality may allow solutions to ignore the equality and follow the gradient provided by other objectives or constraints and overcome local optima.

In this thesis, we also applied EC to tackle a neuroscientific question. The neural circuit discovery method for investigating the role of neural noise in *Drosophila* behavior (presented in **Chapter 5**) is on itself a very important contribution. Our tools and methodology can be readily used for deriving neural circuit models starting from behavioral data of other animals. Our results contribute in many ways to our understanding of the implications of noise in neural systems. Intriguingly, they suggest the possibility that animal behavior can be modified or tuned by regulating the noise levels present in a neural system. Although at the present moment no technological solution appears to be available for testing this hypothesis, we speculate that further investigation into the role of noise in neural systems could provide powerful tools to neuroscientist for controlling behaviors.

Moreover, it is also interesting to look at our results under an evolutionary perspective. We demonstrated how, although important levels of noise influences the operations of single simulated *Drosophila* neural circuits introducing variability at the single fly level, the behavioral

[1] This corresponds, in technical jargon, to extending the (1+1)-CMA-ES method for adapting covariance matrix(Arnold and Hansen, 2012) and step size (presented in this thesis) in presence of constraints to a (μ, λ)-CMA-ES

response of the simulated group of flies is consistent and repeatable. In short, noise could provide evolution a substrate for fine-tuning behaviors, without modifying the underlying neural topologies. We hypothesize that the emergence of noise in a neural system is directly tied with the necessity of fine-tuning animal behaviors, when physiological or environmental constraints impedes the modification of underlying neural topologies. We believe that further investigations on the role of noise in neural systems will provide important insight into our understanding of complex behaviors.

Appendix [redacted]

A Application of mViE to macromolecular assembly prediction

In this appendix we show recent results of the application of mViE on a biological problem. Often, deciphering the three dimensional structure of protein or other macromolecular assemblies can be hard when using traditional techniques like X-ray crystallography. Computational methods can ease the work of an experimenter by providing good approximate predictions of macromolecular assemblies starting from the precise structure of a single monomer composing the assembly. Here, we present a software pipeline for macromolecular assembly prediction that builds upon the mViE method presented in Chapter 4. The method uses multiple search units to identify assemblies satisfying a number of geometrical constraints provided by the user, information that can be obtained from widely available low-resolution imagery techniques. Our method shows competitive performance with respect to a previously available pipeline for assembly prediction, without requiring to compose the geometrical constraints in a single fitness function and tuning the fitness coefficients.

The content of this appendix derives from a recent collaboration with the Laboratory for Biomolecular Modelling at EPFL (Giorgio Tamo, Matteo De Giacomi and Prof. Matteo Dal Peraro) and will be used for an article in preparation.

In this appendix we present recent results of the application of a modified version of mViE to a molecular assembly prediction problem.

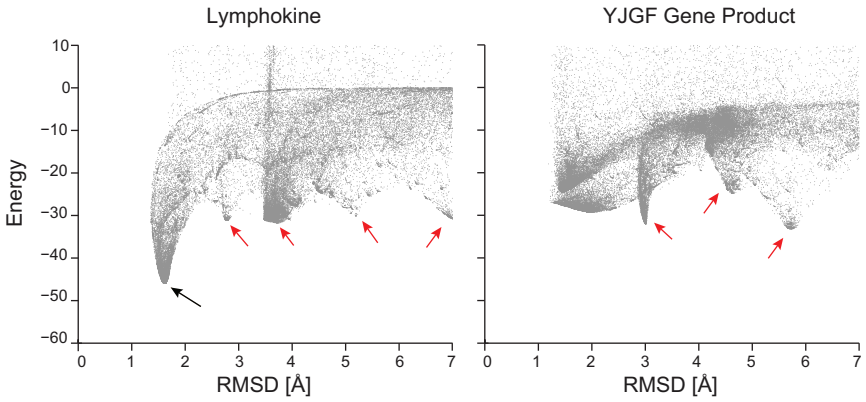


Figure A.1: The majority of current algorithms for predicting macromolecular assemblies evaluate an energy function on each candidate assembly and try to minimize this energy. We show sampled candidate solutions for two assembly prediction problems: Lymphokine and the YJGF gene product. Following the energy gradient in many problems helps discovering solutions with minimal RMSD distance from the true structure (left, Lymphokine, black arrow). However, the presence of multiple wells of energy (left, Lymphokine, red arrows) can hinder the ability of an algorithm in discovering the RMSD optimal solutions. Predicting certain assembly structures is particularly hard, as energy wells associated with bad (large) RMSD values (right, YJGF gene product, red arrow) can have lower energy than wells associated to minimal RMSD values.

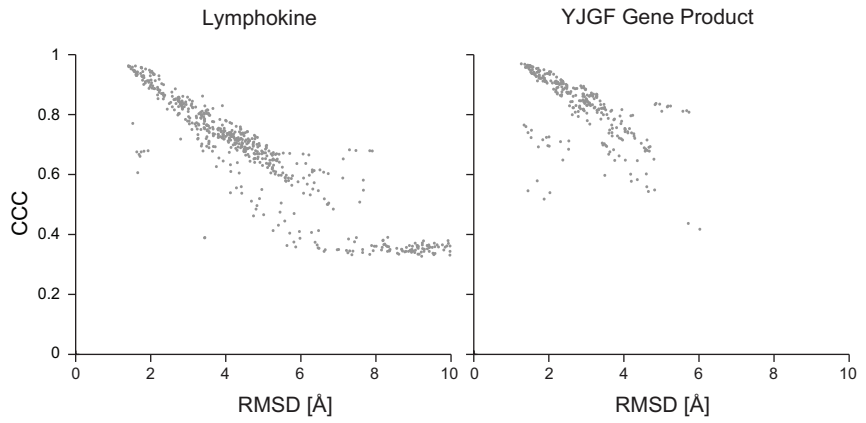


Figure A.2: Cross-correlation coefficient versus RMSD of assembly candidate predictions for the Lymphokine and YJGF gene product. the cross-correlation coefficient is computed against a user-provided density map. Following the CCC gradient can be more effective than following an energy gradient for reaching minimal RMSD assemblies.

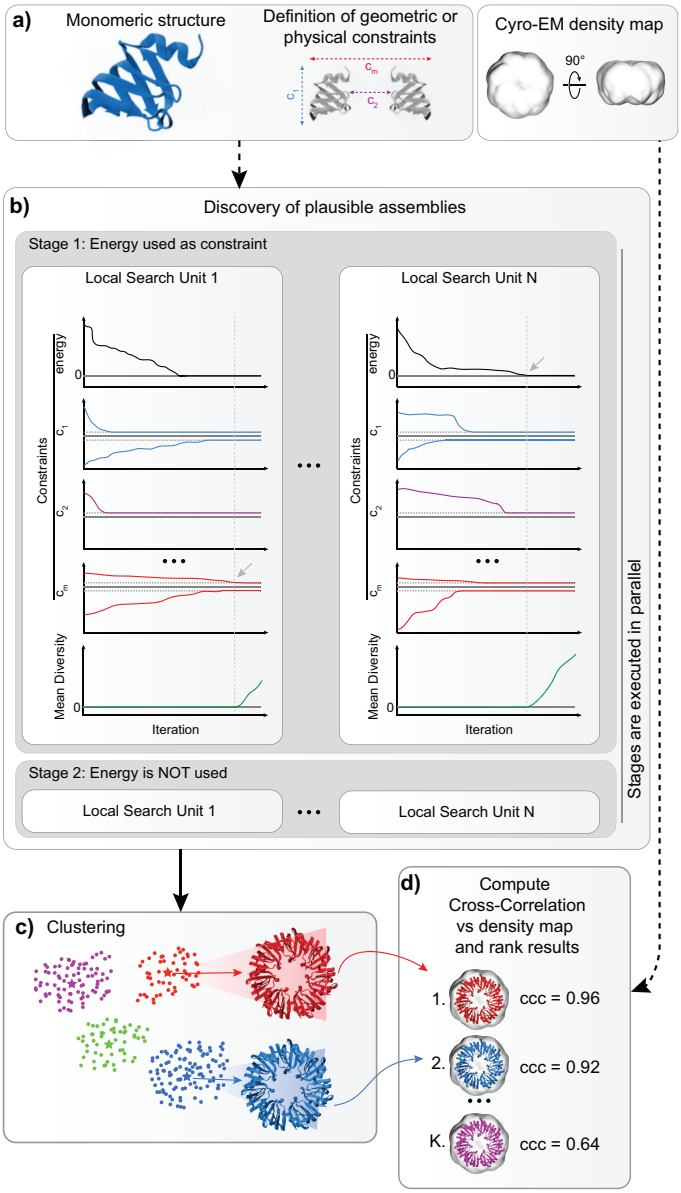
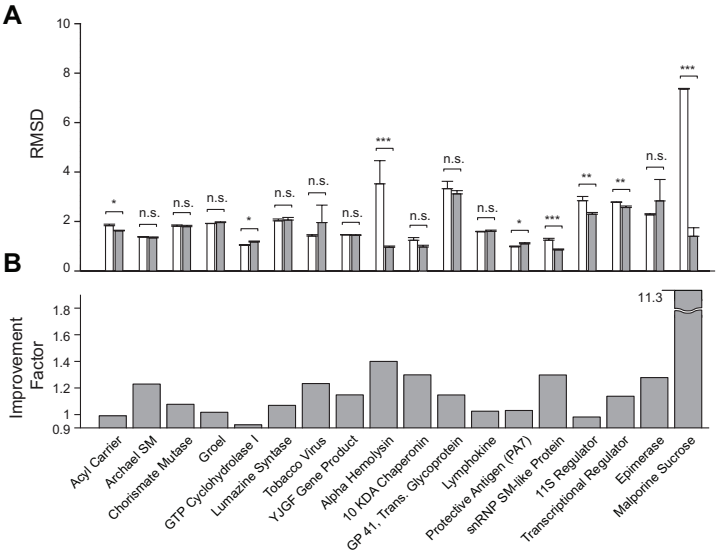


Figure A.3: We propose a two stage protocol for predicting macromolecular assemblies. (a) The ePOW protocol requires the definition of spatial constraints that limit the possible configurations of predicted assembly. Furthermore, the user provides a density map and a crystal structure of a subunit of the assembly. (b) The ePOW protocol employ a two-stages structure prediction pipeline. Both stages uses a stochastic search algorithm, mVIE, presented in Chapter 4 to search for candidate assemblies. mVIE uses a population of multiple search units that try to discover assembly structures satisfying all the constraints. Once a search unit discovers assemblies that do not violate any constraint, it tries to discover assemblies that maximize the diversity with respect to the assemblies predicted by other search units. In the first stage of the protocol, the energy is used as a constraint, looking for assemblies with negative energy levels, i.e. clash free assemblies. In the second stage, however, no energy constraint is used. Candidate assemblies out of the two stages (colored points) that do not violate the constraints are selected and (c) are clustered within a RMSD distance of 1A to remove duplicate sampled assemblies. The clusters' centroids (colored stars) are extracted. (d) The assemblies corresponding to the cluster's centroids are ranked based on their cross-correlation coefficient computed against the density map provided by the user. Finally, the ranked predicted assemblies are returned to the user.



C

	Best RMSD	
	POW [XX]	ePOW
Acyl Carrier	**1.50**	1.51
Archael SM	1.34	**1.09**
Chorismate Mutase	1.61	**1.49**
Groel	1.89	**1.86**
GTP Cyclohydrolase I	**0.94**	1.02
Lumazine Syntase	1.72	**1.61**
Tobacco Virus	1.17	**0.95**
YJGF Gene Product	1.45	**1.27**
Alpha Hemolysin	1.05	**0.75**
10 KDA Chaperonin	0.74	**0.57**
GP 41, Trans. Glycoprotein	2.75	**2.39**
Lymphokine	1.45	**1.41**
Protective Antigen (PA7)	0.86	**0.84**
snRNP SM-like Protein	0.91	**0.70**
11S Regulator	**1.89**	1.92
Transcriptional Regulator	2.67	**2.34**
Epimerase	2	**1.56**
Malporine Sucrose	7.23	**0.64**

Figure A.4: (A) RMSD mean and standard error of top five candidate assemblies returned by the POW (white bars) and ePOW (grey bars) protocols, computed over 10 repetitions of the protocols for each tested assembly. The superscripts above the error bars indicate * $P < 0.05$, ** $P < 0.01$, *** $P < 0.001$ using a Wilcoxon rank sum test, n.s. no statistical significance. (B) Factor of improvement of the ePOW protocol over POW for each tested assembly. (C) Backbone RMSDs of the best predictions found POW and ePOW, for each tested assembly. We highlight in bold the best RMSD value across the two methods.

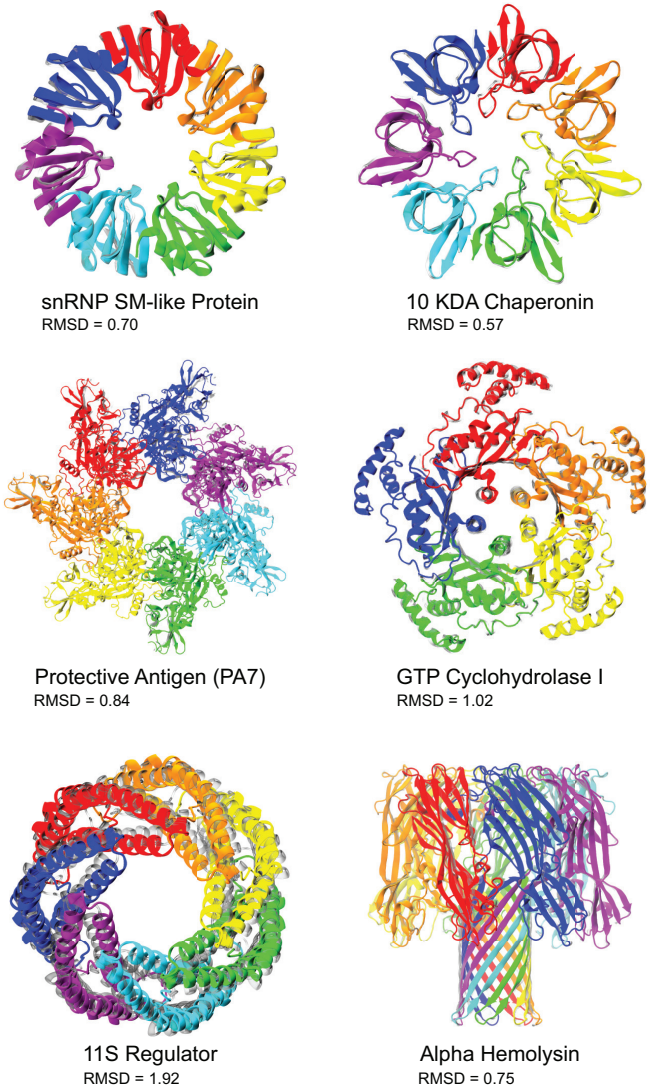


Figure A.5: Best assemblies obtained from the ePOW assembly protocol (in color) superimposed to known bound crystal structures (in transparent white), and RMSD difference between the assemblies.

Bibliography

Adami, C. (2006). Digital genetics: unravelling the genetic basis of evolution. *Nature Review Genetics*, 7(2):109–118, doi:10.1038/nrg1771.

Adra, S. and Fleming, P. J. (2011). Diversity management in evolutionary many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 15(2):183–195, doi:10.1109/TEVC.2010.2058117.

Aguirre, A., Muñoz Zavala, A., Villa Diharce, E., and Botello Rionda, S. (2007). COPSO: Constrained optimization via PSO algorithm. Technical Report I-07-04, Center of Research in Mathematics (CIMAT), Guanajuato, México.

Ahnert, K. and Mulansky, M. (2011). Odeint - solving ordinary differential equations in c++. In *Proceedings of the International Conference on Numerical Analysis and Applied Mathematics*, ICNAAM 2011, pages 1586–1589. AIP.

Ai, M., Min, S., Grosjean, Y., Leblanc, C., Bell, R., Benton, R., and Suh, S. B. (2010). Acid sensing by the drosophila olfactory system. *Nature*, 468(7324):691–695, doi:10.1038/nature09537.

Ali, M. and Zhu, W. (2013). A penalty function-based differential evolution algorithm for constrained global optimization. *Computational Optimization and Applications*, 54(3):707–739, doi:10.1007/s10589-012-9498-3.

Angantyr, A., Andersson, J., and Aidanpaa, J.-O. (2003). Constrained optimization based on a multiobjective evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 3 of *CEC 2003*, pages 1560–1567, Piscataway, NJ, USA. IEEE.

Arnold, D. V. and Hansen, N. (2010). Active covariance matrix adaptation for the (1+1)-cma-es. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 385–392, New York, NY, USA. ACM.

Arnold, D. V. and Hansen, N. (2012). A (1+1)-cma-es for constrained optimisation. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 297–304, New York, NY, USA. ACM.

Atmar, W. (1994). Notes on the simulation of evolution. *IEEE Transactions on Neural Networks*, 5(1):130–147, doi:10.1109/72.265967.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, New York, NY, USA.

Barkat Ullah, A., Sarker, R., and Lokan, C. (2011). Handling equality constraints with agent-based memetic algorithms. *Memetic Computing*, 3(1):51–72, doi:10.1007/s12293-010-0051-6.

Barricelli, N. A. (1954). Esempi numerici di processi di evoluzione. *Methodos*, pages 45–68.

Barricelli, N. A. (1962). Numerical testing of evolution theories. part i: Theoretical introduction and basic tests. *Acta Biotheoretica*, 16(1–2):69–98.

Barricelli, N. A. (1963). Numerical testing of evolution theories. part ii: Preliminary tests of performance. symbiogenesis and terrestrial life. *Acta Biotheoretica*, 16(3–4):99–126.

Baum, E. B., Boneh, D., and Garrett, C. (2001). Where genetic algorithms excel. *Evolutionary Computation*, 9(1):93–124, doi:10.1162/1063656015107513.

Becerra, R. L. and Coello Coello, C. A. (2006). Cultured differential evolution for constrained optimization. *Computer Methods in Applied Mechanics and Engineering*, 195(33-36):4303–4322, doi:10.1016/j.cma.2005.09.006.

Beer, R. D. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4):469–509, doi:0.1177/105971239500300405.

Beer, R. D. and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122, doi:10.1177/105971239200100105.

Benettin, G., Galgani, L., Giorgilli, A., and Strelcyn, J.-M. (1980). Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. part 1: Theory. *Meccanica*, 15(1):9–20, doi:10.1007/BF02128236.

Bernardino, H., Barbosa, H., and Lemong, A. (2007). A hybrid genetic algorithm for constrained optimization problems in mechanical engineering. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2007, pages 646–653. IEEE.

Beyer, H.-G. and Finck, S. (2012). On the design of constraint covariance matrix self-adaptation evolution strategies including a cardinality constraint. *IEEE Transactions on Evolutionary Computation*, 16(4):578–596, doi:10.1109/TEVC.2011.2169967.

Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies – a comprehensive introduction. *Natural Computing*, 1(1):3–52, doi:10.1023/A:1015059928466.

Bidaye, S. S., Machacek, C., Wu, Y., and Dickson, B. J. (2014). Neuronal control of drosophila walking direction. *Science*, 344(6179):97–101, doi:10.1126/science.1249964.

Bongard, J. (2011). Morphological change in machines accelerates the evolution of robust behavior. *PNAS*, 108(4):1234–1239, doi:10.1073/pnas.1015390108.

Bonyadi, M., Li, X., and Michalewicz, Z. (2013). A hybrid particle swarm with velocity mutation for constraint optimization problems. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1–8. ACM.

Brajevic, I., Tuba, M., and Subotic, M. (2011). Performance of the improved artificial bee colony algorithm on standard engineering constrained problems. *International Journal of Mathematics And Computers In Simulation*, 5(1):135–143.

Branson, K., Robie, A. A., Bender, J., Perona, P., and Dickinson, M. H. (2009). High-throughput ethomics in large groups of drosophila. *Nature Methods*, 6:451–457, doi:10.1038/nmeth.1328.

Brest, J., Zumer, V., and Maucec, M. (2006). Self-adaptive differential evolution algorithm in constrained real-parameter optimization. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B., Coello Coello, C. A., and Runarsson, T. D., editors, *2006 IEEE Congress on Evolutionary Computation*, CEC 2006, pages 215–222. IEEE.

Cagnina, L. C., Esquivel, S. C., and Coello, C. A. C. (2008). Solving engineering optimization problems with the simple constrained particle swarm optimizer. *Informatica*, 32(3):319–326.

Cai, Z. and Wang, Y. (2006). A multiobjective optimization-based evolutionary algorithm for constrained optimization. *IEEE Transactions on Evolutionary Computation*, 10(6):658–675, doi:10.1109/TEVC.2006.872344.

Censi, A., Straw, A. D., Sayaman, R. W., Murray, R. M., and Dickinson, M. H. (2013). Discriminating external and internal causes for heading changes in freely flying drosophila. *PLoS Computational Biology*, 9(2):e1002891, doi:10.1371/journal.pcbi.1002891.

Clerc, M. (2010). *Particle Swarm Optimization*. John Wiley & Sons, New Your, NY, USA.

Clevenger, L., Ferguson, L., and Hart, W. E. (2005). A filter-based evolutionary algorithm for constrained optimization. *Evolutionary Computation*, 13(3):329–352, doi:10.1162/1063656054794789.

Clune, J., Goldsby, H. J., Ofria, C., and Pennock, R. T. (2011). Selective pressures for accurate altruism targeting: evidence from digital evolution for difficult-to-test aspects of inclusive fitness theory. *Proceedings of the Royal Society B*, 278(1706):666–674, doi:10.1098/rspb.2010.1557.

Clune, J., Misevic, D., Ofria, C., Lenski, R. E., Elena, S. F., and Sanjuán, R. (2008). Natural selection fails to optimize mutation rates for long-term adaptation on rugged fitness landscapes. *PLoS Computational Biology*, 4(9):e1000187, doi:10.1371/journal.pcbi.1000187.

Coelho, L. d. S. (2010). Gaussian quantum-behaved particle swarm optimization approaches for constrained engineering design problems. *Expert Systems with Applications*, 37(2):1676–1683, doi:10.1016/j.eswa.2009.06.044.

Coello Coello, C. A. (2000). Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, 41(2):113–127, doi:10.1016/S0166-3615(99)00046-9.

Coello Coello, C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering*, 191(11-12):1245–1287, doi:10.1016/S0045-7825(01)00323-1.

Coello Coello, C. A. (2006). Evolutionary multi-objective optimization: a historical view of the field. *IEEE Computational Intelligence Magazine*, 1(1):28–36, doi:10.1109/MCI.2006.1597059.

Coello Coello, C. A. and Mezura-Montes, E. (2002). Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, 16(3):193–203, doi:10.1016/S1474-0346(02)00011-3.

Coen, P., Clemens, J., Weinstein, A. J., Pacheco, D. A., Deng, Y., and Murthy, M. (2014). Dynamic sensory cues shape song structure in drosophila. *Nature*, 507(7491):223–227, doi:10.1038/nature13131.

Collange, G., Delattre, N., Hansen, N., Quinquis, I., and Schoenauer, M. (2010). Multidisciplinary optimization in the design of future space launchers. In *Multidisciplinary Design Optimization in Computational Mechanics*, pages 459–468. John Wiley & Sons, Inc.

Collard, P. and Escazut, C. (1995). Genetic operators in a dual genetic algorithm. In *Proceedings of the Seventh International Conference on Tools with Artificial Intelligence*, pages 12–19.

Cossart, R., Aronov, D., and Yuste, R. (2003). Attractor dynamics of network up states in the neocortex. *Nature*, 423(6937):283–288, doi:10.1038/nature01614;.

Cuevas, E. and Cienfuegos, M. (2014). A new algorithm inspired in the behavior of the social-spider for constrained optimization. *Expert Systems with Applications*, 41(2):412–425, doi:10.1016/j.eswa.2013.07.067.

Darwin, C. (1859). *On the origin of species by means of natural selection*. Murray, London, UK.

De Jong, K. A. (1975). *Analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan.

de Melo, V. V. and Carosio, G. L. C. (2012). Evaluating differential evolution with penalty function to solve constrained engineering problems. *Expert Systems with Applications*, 39(9):7860–7863, doi:10.1016/j.eswa.2012.01.123.

de Melo, V. V. and Iacca, G. (2014). A modified covariance matrix adaptation evolution strategy with adaptive penalty function and restart for constrained optimization. *Expert Systems with Applications*, 41(16):7077–7094, doi:10.1016/j.eswa.2014.06.032.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2-4):311–338, doi:10.1016/S0045-7825(99)00389-8.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, Inc., New York, NY, USA.

Deb, K. and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002a). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, doi:10.1109/4235.996017.

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2002b). Scalable multi-objective optimization test problems. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 1 of *CEC 2002*, pages 825–830. IEEE.

Degiacomi, M. T. and Dal Peraro, M. (2013). Macromolecular symmetric assembly prediction using swarm intelligence dynamic modeling. *Structure*, 21(7):1097–1106, doi:10.1016/j.str.2013.05.014.

den Boer, P. J. (1999). Natural selection or the non-survival of the non-fit. *Acta Biotheoretica*, 47(2):83–97, doi:10.1023/A:1002053820381.

Destexhe, A. and Rudolph-Lilith, M. (2012). *Neuronal noise*. Springer US, New York, NY, USA.

Dorigo, M., Maniezzo, V., and Colorni, A. (1991). Positive feedback as a search strategy. Technical report, Politecnico di Milano.

Dunn, N. A., Lockery, S. R., Pierce-Shimomura, J. T., and Conery, J. S. (2004). A neural network model of chemotaxis predicts functions of synaptic connections in the nematode caenorhabditis elegans. *Journal of Computational Neuroscience*, 17(2):137–147, doi:10.1023/B:JCNS.0000037679.42570.d5.

Eberhart, R. C. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. IEEE.

Eiben, A. and Bäck, T. (1997). Empirical investigation of multiparent recombination operators in evolution strategies. *Evolutionary Computation*, 5(3):347–365, doi:10.1162/evco.1997.5.3.347.

Eiben, A. and Schoenauer, M. (2002). Evolutionary computing. *Information Processing Letters*, 82(1):1–6, doi:10.1016/S0020-0190(02)00204-1.

Eiben, A. E. and Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Springer, Berlin, Germany.

Faisal, A. A., Selen, L. P. J., and Wolpert, D. M. (2008). Noise in the nervous system. *Nature Reviews Neuroscience*, 9:292–303, doi:10.1038/nrn2258.

Faisal, A. A., White, J. A., and Laughlin, S. B. (2005). Ion-channel noise places limits on the miniaturization of the brain's wiring. *Current Biology*, 15(12):1143–1149, doi:10.1016/j.cub.2005.05.056.

Farmer, J. D., Packard, N. H., and Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Phys. D*, 2(1-3):187–204, doi:10.1016/0167-2789(81)90072-5.

Fatt, P. and Katz, B. (1952). Spontaneous subthreshold activity at motor nerve endings. *Journal of Physiology*, 117(1):109–128.

Flavell, S. W., Pokala, N., Macosko, E. Z., Albrecht, D. R., Larsch, J., and Bargmann, C. I. (2013). Serotonin and the neuropeptide pdf initiate and extend opposing behavioral states in c. elegans. *Cell*, 154(5):1023–1035, doi:10.1016/j.cell.2013.08.001.

Floreano, D. and Keller, L. (2010). Evolution of adaptive behaviour in robots by means of darwinian selection. *PLoS Biology*, 8(1):e1000292, doi:10.1371/journal.pbio.1000292.

Fogel, D. B. (1994). An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, doi:10.1109/72.265956.

Fogel, D. B. (1995). *Evolutionary computation: toward a new philosophy of machine intelligence*. IEEE Press, Piscataway, NJ, USA.

Fogel, D. B. (1998a). *Evolutionary computation: The Fossil Record*. Wiley-IEEE Press, Piscataway, NJ, USA.

Fogel, D. B. (1998b). Unearthing a fossil from the history of evolutionary computation. *Fundamenta Informaticae*, 35(1–4):1–16.

Fogel, L. J. (1962). Autonomous automata. *Industrial Research*, 4(2):14–19.

Foster, J. A. (2001). Evolutionary computation. *Nature Review Genetics*, 2(6):428–36, doi:10.1038/35076523.

Funahashi, K.-I. and Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806, doi:10.1016/S0893-6080(05)80125-X.

Gandomi, A. H. and Yang, X.-S. (2011). Benchmark problems in structural optimization. In Koziel, S. and Yang, X.-S., editors, *Computational Optimization, Methods and Algorithms*, chapter 12, pages 259–281. Springer, Berlin, Germany.

Gandomi, A. H., Yang, X.-S., and Alavi, A. H. (2011). Mixed variable structural optimization using firefly algorithm. *Computers & Structures*, 89(23):2325–2336, doi:10.1016/j.compstruc.2011.08.002.

Geiger, R. and Sanchez-Sinencio, E. (1985). Active filter design using operational transconductance amplifiers: A tutorial. *IEEE Circuits and Devices Magazine*, 1(2):20–32, doi:10.1109/MCD.1985.6311946.

Gieseke, F. and Kramer, O. (2013). Towards non-linear constraint estimation for expensive optimization. In Esparcia-Alcázar, A., editor, *Applications of Evolutionary Computation*, pages 459–468. Springer Berlin Heidelberg.

Ginley, B. M., Maher, J., Riordan, C. O., and Morgan, F. (2011). Maintaining healthy population diversity using adaptive crossover, mutation and selection. *IEEE Transactions on Evolutionary Computation*, 15(5):692–714, doi:10.1109/TEVC.2010.2046173.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

Gomes, J., Urbano, P., and Christensen, A. (2012). Progressive minimal criteria novelty search. In Pavón, J., Duque-Méndez, N., and Fuentes-Fernández, R., editors, *Advances in Artificial Intelligence – IBERAMIA 2012*, volume 7637 of *Lecture Notes in Computer Science*, pages 281–290. Springer Berlin Heidelberg.

Gordon, M. D. and Scott, K. (2009). Motor control in a drosophila taste circuit. *Neuron*, 61(3):373–384, doi:10.1016/j.neuron.2008.12.033.

Grillner, S. and Jessell, T. (2009). Measured motion: searching for simplicity in spinal locomotor networks. *Current Opinion in Neurobiology*, 19(6):572–586, doi:10.1016/j.conb.2009.10.011.

Gu, H., Jiang, S., Campusano, J., Iniguez, J., Su, H., Hoang, A., Lavian, M., Sun, X., and O'Dowd, D. (2008). $Ca_2$2-type calcium channels encoded by cac regulate ap-independent neurotransmitter release at cholinergic synapses in adult drosophila brain. *Journal of Neurophysiology*, 101(1):42–53, doi:10.1152/jn.91103.2008.

Hamida, S. B. and Schoenauer, M. (2002). ASCHEA: New Results Using Adaptive Segregational Constraint Handling. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1 of *CEC 2002*, pages 884–889. IEEE.

Hamza, N. M., Sarker, R. A., Essam, D. L., Deb, K., and Elsayed, S. M. (2014). A constraint consensus memetic algorithm for solving constrained optimization problems. *Engineering Optimization*, 46(11):1447–1464, doi:10.1080/0305215X.2013.846336.

Hansen, N., Auger, A., Ros, R., Finck, S., and Pošík, P. (2010). Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '10, pages 1689–1696, New York, NY, USA. ACM.

Hansen, N., Muller, S., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, doi:10.1162/106365603321828970.

He, Q. and Wang, L. (2007a). An effective co-evolutionary particle swarm optimization for constrained engineering design problems. *Engineering Applications of Artificial Intelligence*, 20(1):89–99, doi:10.1016/j.engappai.2006.03.003.

He, Q. and Wang, L. (2007b). A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. *Applied Mathematics and Computation*, 186(2):1407–1422, doi:10.1016/j.amc.2006.07.134.

Hoffmeister, F. and Sprave, J. (1996). Problem-independent handling of constraints by use of metric penalty functions. In *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 289–294. MIT Press.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press.

Hornby, G., Lohn, J., and Linden, D. (2011). Computer-automated evolution of an x-band antenna for nasa's space technology 5 mission. *Evolutionary computation*, 19(1):1–23, doi:10.1162/EVCO_a_00005.

Hu, X., Eberhart, R., and Shi, Y. (2003). Engineering optimization with particle swarm. In *Proceedings of the IEEE Swarm Intelligence Symposium*, SIS 2003, pages 53–57.

Huang, V., Qin, A., and Suganthan, P. (2006). Self-adaptive differential evolution algorithm for constrained real-parameter optimization. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B., Coello Coello, C. A., and Runarsson, T. D., editors, *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 17–24. IEEE.

Igel, C., Suttorp, T., and Hansen, N. (2006). A computational efficient covariance matrix update and a (1+1)-cma for evolution strategies. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 453–460, New York, NY, USA. ACM.

Izquierdo, E. and Lockery, S. (2010). Evolution and analysis of minimal neural circuits for klinotaxis in caenorhabditis elegans. *The Journal of Neuroscience*, 30(39):12908–12917, doi:10.1523/JNEUROSCI.2606-10.2010.

Jia, G., Wang, Y., Cai, Z., and Jin, Y. (2013). An improved $(\mu + \lambda)$-constrained differential evolution for constrained optimization. *Information Sciences*, 222:302–322, doi:10.1016/j.ins.2012.01.017.

Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, doi:10.1016/j.swevo.2011.05.001.

Juric, M. (1994). An anti-adaptationist approach to genetic algorithms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 2, pages 619–623.

Kämpf, J. H. and Robinson, D. (2010). Optimisation of building form for solar energy utilisation using constrained evolutionary algorithms. *Energy and Buildings*, 42(6):807–814, doi:10.1016/j.enbuild.2009.11.019.

Kaplan, J. and Yorke, J. (1979). Chaotic behavior of multidimensional difference equations. In Peitgen, H.-O. and Walther, H.-O., editors, *Functional Differential Equations and Approximation of Fixed Points*, volume 730 of *Lecture Notes in Mathematics*, pages 204–227. Springer Berlin Heidelberg.

Kawasaki, F., Felling, R., and Ordway, R. W. (2000). A temperature-sensitive paralytic mutant defines a primary synaptic calcium channel in drosophila. *The Journal of Neuroscience*, 20(13):4885–4889.

Koch, C., Cziesla, F., and Tsatsaronis, G. (2007). Optimization of combined cycle power plants using evolutionary algorithms. *Chemical Engineering and Processing: Process Intensification*, 46(11):1151–1159, doi:10.1016/j.cep.2006.06.025.

Kominami, M. and Hamagami, T. (2007). A new genetic algorithm with diploid chromosomes by using probability decoding for non-stationary function optimization. In *IEEE International Conference on Systems, Man and Cybernetics*, ISIC 2007, pages 1268–1273.

Kong, X., Ouyang, H., and Piao, X. (2013). A prediction-based adaptive grouping differential evolution algorithm for constrained numerical optimization. *Soft Computing*, 17(12):2293–2309, doi:10.1007/s00500-013-1090-y.

Kramer, O., Barthelmes, A., and Rudolph, G. (2009). Surrogate constraint functions for cma evolution strategies. In *KI 2009: Advances in Artificial Intelligence*, volume 5803, pages 169–176. Springer Berlin Heidelberg.

Kramer, O., Schlachter, U., and Spreckels, V. (2013). An adaptive penalty function with meta-modeling for constrained problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2013, pages 1350–1354. IEEE.

Kramer, O. and Schwefel, H. (2006). On three new approaches to handle constraints within evolution strategies. *Natural Computing*, 5(4):1–22, doi:10.1007/s11047-006-0001-x.

Kramer, O., Ting, C.-K., and Büning, H. K. (2005). A new mutation operator for evolution strategies for constrained problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 3 of *CEC 2005*, pages 2600–2606. IEEE.

Kukkonen, S. and Lampinen, J. (2006). Constrained real-parameter optimization with generalized differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 911–918, Vancouver, BC, Canada. IEEE.

Kusakci, A. O. and Can, M. (2013a). An adaptive penalty based covariance matrix adaptation–evolution strategy. *Computers & Operations Research*, 40(10):2398–2417, doi:10.1016/j.cor.2013.03.013.

Kusakci, A. O. and Can, M. (2013b). A novel evolution strategy for constrained optimization in engineering design. In *XXIV International Symposium on Information, Communication and Automation Technologies (ICAT)*, pages 1–6. IEEE.

Lässig, J. and Hoffmann, K. H. (2009). Threshold-selecting strategy for best possible ground state detection with genetic algorithms. *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, 79(4):046702, doi:10.1103/PhysRevE.79.046702.

Leguizamón, G. and Coello Coello, C. A. (2009). Boundary search for constrained numerical optimization problems with an algorithm inspired on the ant colony metaphor. *IEEE Transactions on Evolutionary Computation*, 13(2):350–368, doi:10.1109/TEVC.2008.926731.

Lehman, J. and Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336.

Lehman, J. and Stanley, K. O. (2010). Revising the evolutionary computation abstraction: Minimal criteria novelty search. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 103–110, New York, NY, USA. ACM.

Lehman, J. and Stanley, K. O. (2011). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 211–218, New York, NY, USA. ACM.

Lenski, R. E., Ofria, C., Pennock, R., and Adami, C. (2003). Evolutionary origin of complex features. *Nature*, 423:139–144, doi:10.1038/nature01568.

Lewontin, R. C. (1970). The units of selection. *Annual Review of Ecology and Systematics*, 1(1):1–18, doi:10.1146/annurev.es.01.110170.000245.

Li, J.-P., Balazs, M. E., Parks, G. T., and Clarkson, P. J. (2002). A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation*, 10(3):207–34, doi:10.1162/106365602760234081.

Liang, J., Runarsson, T., Mezura-Montes, E., Clerc, M., Suganthan, P., Coello Coello, C. A., and Deb, K. (2006). Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization. Technical report, Nanyang Technological University, Singapore.

Liang, J. and Suganthan, P. (2006). Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B., Coello Coello, C. A., and Runarsson, T. D., editors, *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 9–16. IEEE.

Liu, H., Cai, Z., and Wang, Y. (2010). Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. *Applied Soft Computing*, 10(2):629–640, doi:10.1016/j.asoc.2009.08.031.

Long, W., Liang, X., Huang, Y., and Chen, Y. (2013). A hybrid differential evolution augmented lagrangian method for constrained numerical and engineering optimization. *Computer-Aided Design*, 45(12):1562–1574, doi:10.1016/j.cad.2013.07.007.

Lung, R. I. (2004). A subpopulation stability based evolutionary technique for multimodal optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '04, pages 26–30.

Mackay, T. F. C., Richards, S., Stone, E. A., Barbadilla, A., Ayroles, J., Zhu, D., Casillas, S., Han, Y., Magwire, M. M., Cridland, J. M., Richardson, M. F., Anholt, R. R. H., Barrón, M., Bess, C., Blankeburg, K. P., Carbone, M. A., Castellano, D., Chaboub, L., Duncan, L., Harris, Z., Javaid, M., Jayaseelan, J. C., Jhangiani, S. N., Jordan, K. W., Lara, F., Lawrence, F., Lee, S. L., Librado, P., Linheiro, R. S., Lyman, R. F., Mackey, A. J., Munidasa, M., Muzny, D. M., Nazareth, L., Newsham, I., Perales, L., Pu, L., Qu, C., Ramia, M., Reid, J. G., Rollmann, S. M., Rozas, J., Saada, N., Turlapati, L., Worley, K. C., Wu, Y., Yamamoto, A., Zhu, Y., Bergman, C. M., Thornton, K. R., Mittelman, D., and Gibbs, R. A. (2012). The drosophila melanogaster genetic reference panel. *Nature*, 482(7384):173–178, doi:10.1038/nature10811.

Mackay, T. F. C., Stone, E. A., and Ayroles, J. F. (2009). The genetics of quantitative traits: challenges and prospects. *Nature Reviews Genetics*, 10:565–577, doi:10.1038/nrg2612.

Maesani, A., Fernando, P. R., and Floreano, D. (2014). Artificial evolution by viability rather than competition. *PLoS ONE*, 9(1):e86831, doi:10.1371/journal.pone.0086831.

Maesani, A. and Floreano, D. (2014). Viability principles for constrained optimization using a (1+1)-cma-es. In *Parallel Problem Solving from Nature*, PPSN 14. To appear.

Maimon, G., Straw, A. D., and Dickinson, M. H. (2010). Active flight increases the gain of visual motion processing in drosophila. *Nature Neuroscience*, 13(3):393–399, doi:10.1038/nn.2492.

Mallipeddi, R. and Suganthan, P. N. (2010). Ensemble of constraint handling techniques. *IEEE Transactions on Evolutionary Computation*, 14(4):561–579, doi:10.1109/TEVC.2009.2033582.

Mante, V., Sussillo, D., Shenoy, K. V., and Newsome, W. T. (2013). Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503(7474):78–84, doi:10.1038/nature12742.

Marder, E. (2011). Variability, compensation, and modulation in neurons and circuits. *Proceedings of the National Academy of Science USA*, 108(SUPPL. 3):15542–15548, doi:10.1073/pnas.1010674108.

Marder, E. and Taylor, A. L. (2011). Multiple models to capture the variability in biological neurons and networks. *Nature Neuroscience*, 14:133–138, doi:10.1038/nn.2735.

Marín, J. and Solé, R. (1999). Macroevolutionary algorithms: A new optimization method on fitness landscapes. *IEEE Transactions on Evolutionary Computation*, 3(4):272–286, doi:10.1109/4235.797970.

Martin, J. R. (2004). A portrait of locomotor behaviour in drosophila determined by a video-tracking paradigm. *Behavioural processes*, 67(2):207–219, doi:10.1016/j.beproc.2004.04.003.

Martin, J. R., Ernst, R., and Heisenberg, M. (1999). Temporal pattern of locomotor activity in drosophila melanogaster. *Journal of Computational Physiology A*, 184(1):73–84, doi:10.1007/s003590050307.

Martin, J. R., Faure, P., and Ernst, R. (2001). The power law distribution for walking-time intervals correlates with the ellipsoid-body in drosophila. *Journal of Neurogenetics*, 15(3-4):205–219, doi:10.3109/01677060109167377.

Mattiussi, C. and Floreano, D. (2003). Viability evolution: Elimination and extinction in evolutionary computation. Technical report, EPFL. EPFL-REPORT-177577, Available from: http://infoscience.epfl.ch/record/177577.

Mattiussi, C., Waibel, M., and Floreano, D. (2004). Measures of diversity for populations and distances between individuals with highly reorganizable genomes. *Evolutionary Computation*, 12(4):495–515, doi:10.1162/1063656043138923.

Maye, A., Hsieh, C., Sugihara, G., and Brembs, B. (2007). Order in spontaneous behavior. *PLoS ONE*, 2(5):e443, doi:10.1371/journal.pone.0000443.

Mayr, E. (2002). *What Evolution Is.* Orion Books Ltd., London, UK.

McDonnell, M. D. and Abbott, D. (2009). What is stochastic resonance? definitions, misconceptions, debates, and its relevance to biology. *PLoS Computational Biology*, 5(5):e1000348, doi:10.1371/journal.pcbi.1000348.

McDonnell, M. D. and Ward, L. M. (2011). The benefits of noise in neural systems: bridging theory and experiment. *Nature Reviews Neuroscience*, 12:415–426, doi:10.1038/nrn3061.

Mezura-Montes, E. and Coello, C. A. C. (2008). Constrained optimization via multiobjective evolutionary algorithms. In Knowles, J., Corne, D., Deb, K., and Chair, D., editors, *Multiobjective Problem Solving from Nature*, Natural Computing Series, pages 53–75. Springer Berlin Heidelberg.

Mezura-Montes, E., Coello Coello, C., and Tun-Morales, E. (2004). Simple feasibility rules and differential evolution for constrained optimization. In Monroy, R., Arroyo-Figueroa, G., Sucar, L., and Sossa, H., editors, *MICAI 2004: Advances in Artificial Intelligence*, volume 2972 of *Lecture Notes in Computer Science*, pages 707–716. Springer Berlin Heidelberg.

Mezura-Montes, E. and Coello Coello, C. A. (2005). A simple multimembered evolution strategy to solve constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 9(1):1–17, doi:10.1109/TEVC.2004.836819.

Mezura-Montes, E. and Coello Coello, C. A. (2011). Constraint-handling in nature-inspired numerical optimization: past, present and future. *Swarm and Evolutionary Computation*, 1(4):173–194, doi:10.1016/j.swevo.2011.10.001.

Mezura-Montes, E., Coello Coello, C. A., and Velázquez-Reyes, J. (2006). Increasing successful offspring and diversity in differential evolution for engineering design. In *Proceedings of the 7th International Conference on Adaptive Computing in Design and Manufacture*, ACDM 2006, pages 131–139.

Mezura-Montes, E. and Lopez-Ramirez, B. C. (2007). Comparing bio-inspired algorithms in constrained optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2007, pages 662–669. IEEE.

Mezura-Montes, E., Miranda-Varela, M. E., and del Carmen Gomez-Ramon, R. (2010). Differential evolution in constrained numerical optimization: An empirical study. *Information Sciences*, 180(22):4223–4262, doi:10.1016/j.ins.2010.07.023.

Mezura-Montes, E. and Palomeque-Ortiz, A. (2009). Self-adaptive and deterministic parameter control in differential evolution for constrained optimization. In Mezura-Montes, E., editor, *Constraint-Handling in Evolutionary Optimization*, volume 198 of *Studies in Computational Intelligence*, pages 95–120. Springer Berlin Heidelberg.

Mezura-Montes, E. and Palomeque-Ortiz, A. G. (2009). Parameter control in differential evolution for constrained optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2009, pages 1375–1382. IEEE.

Mezura-Montes, E., Velazquez-Reyes, H., and Coello Coello, C. A. (2006). Modified differential evolution for constrained optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 25–32. IEEE.

Michalewicz, Z. (1995). A survey of constraint handling techniques in evolutionary computation methods. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155, Cambridge, MA, USA. MIT Press.

Michalewicz, Z., Dasgupta, D., Riche, R. L., and Schoenauer, M. (1996). Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering Journal*, 30(4):851–870, doi:10.1016/0360-8352(96)00037-X.

Mitri, S., Floreano, D., and Keller, L. (2009). The evolution of information suppression in communicating robots with conflicting interests. *PNAS*, 106(37):15786–14790, doi:10.1073/pnas.0903152106.

Mohamed, A. W. and Sabry, H. Z. (2012). Constrained optimization based on modified differential evolution algorithm. *Information Sciences*, 194:171–208, doi:10.1016/j.ins.2012.01.008.

Montanier, J.-M. and Bredeche, N. (2013). Evolution of altruism and spatial dispersion: an artificial evolutionary ecology approach. In *Advances in Artificial Life*, volume 12 of *ECAL 2013*, pages 260–267.

Mouret, J.-B. (2011). Novelty-based multiobjectivization. In *New Horizons in Evolutionary Robotics*, volume 341, pages 139–154. Springer Berlin Heidelberg.

Mühlenbein, H. and Schlierkamp-Voosen, D. (1993). The science of breeding and its application to the breeder genetic algorithm. *Evolutionary Computation*, 1(4):335–360, doi:10.1162/evco.1993.1.4.335.

Munoz-Zavala, A., Hernandez-Aguirre, A., Villa-Diharce, E., and Botello-Rionda, S. (2006). Peso+ for constrained optimization. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B.-T., Coello Coello, C. A., and Runarsson, T. P., editors, *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 231–238. IEEE.

Nelson, A. L., Barlow, G. J., and Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370, doi:10.1016/j.robot.2008.09.009.

Neri, F., Cotta, C., and Moscato, P., editors (2011). *Handbook of Memetic Algorithms*, volume 379 of *Studies in Computational Intelligence*. Springer.

Neri, F., Toivanen, J., Cascella, G. L., and Ong, Y.-S. (2007). An adaptive multimeme algorithm for designing hiv multidrug therapies. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(2):264–278, doi:10.1109/TCBB.2007.070202.

Ofria, C. and Wilke, C. O. (2004). Avida: a software platform for research in computational evolutionary biology. *Artificial Life*, 10(2):191–229, doi:10.1162/106454604773563612.

Oyman, A., Deb, K., and Beyer, H.-G. (1999). An alternative constraint handling method for evolution strategies. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, volume 1 of *CEC 1999*, pages 612–619. IEEE.

Pant, M., Thangaraj, R., and Abraha, A. (2009). Low discrepancy initialized particle swarm optimization for solving constrained optimization problems. *Fundamenta Informaticae*, 95(4):511–531, doi:10.3233/FI-2009-162.

Park, T. and Ryu, K. R. (2010). A dual-population genetic algorithm for adaptive diversity control. *IEEE Transactions on Evolutionary Computation*, 14(6):865–884, doi:10.1109/TEVC.2010.2043362.

Pescador Rojas, M. and Coello Coello, C. A. (2012). A memetic algorithm with simplex crossover for solving constrained optimization problems. In *World Automation Congress*, WAC 2012, pages 1–6. IEEE.

Petrowski, A. (1996). A clearing procedure as a niching method for genetic algorithms. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 798–803.

Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, doi:10.1007/s11721-007-0002-0.

Portugues, R., Feierstein, C. E., Engert, F., and Orger, M. B. (2014). Whole-brain activity maps reveal stereotyped, distributed networks for visuomotor behavior. *Neuron*, 81(6):1328–1343, doi:10.1016/j.neuron.2014.01.019.

Price, K. V., Storn, R. M., and Lampinen, J. A. (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Springer Berlin Heidelberg, Berlin, Germany.

Proekt, A., Banavar, J. R., Maritan, A., and Pfaff, D. W. (2012). Scale invariance in the dynamics of spontaneous behavior. *Proceedings of the National Academy of Science USA*, 109(26):10564–10569, doi:10.1073/pnas.1206894109.

R. Mallipeddi, P. S. (2010). Problem definitions and evaluation criteria for the cec 2010 competition on constrained real-parameter optimization. Technical report, Nanyang Technological University, Singapore.

Ramdya, P. and Benton, R. (2010). Evolving olfactory systems on the fly. *Trends in Genetics*, 26(7):307–316, doi:10.1016/j.tig.2010.04.004.

Ramdya, P. and Engert, F. (2008). Emergence of binocular functional properties in a monocular neural circuit. *Nature Neuroscience*, 11(9):1083–1090, doi:10.1038/nn.2166.

Ramdya, P., Schaffter, T., Floreano, D., and Benton, R. (2012). Fluorescence behavioral imaging (fbi) tracks identity in heterogeneous groups of drosophila. *PLoS ONE*, 7(11):e48381, doi:10.1371/journal.pone.0048381.

Rasheed, K. (1998). An adaptive penalty approach for constrained genetic-algorithm optimization. In *Proceedings of the Third Annual Genetic Programming Conference*, pages 584–590. Morgan Kaufmann Publishers.

Ray, T. (1991). Evolution and optimization of digital organisms. In R., B. K., Derohanes, E., and H. Brown, I., editors, *Scientific Excellence in Supercomputing: The IBM 1990 Contest Prize Papers*, pages 489–531. The Baldwin Press.

Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. In *Royal Aircraft Establishment Translation No. 1122, B. F. Toms, Trans.* Ministry of Aviation, Royal Aircraft Establishment, Farnborough Hants.

Reichardt, W. (1961). Autocorrelation, a principle for the evaluation of sensory information by the central nervous system. *Sensory Communication*, pages 303–317.

Rieckhof, G. E., Yoshinara, M., Guan, Z., and Littleton, J. (2003). Presynaptic n-type calcium channels regulate synaptic growth. *Journal of Biological Chemistry*, 278(42):41099–41108, doi:10.1074/jbc.M306417200.

Rönkkönen, J., Li, X., Kyrki, V., and Lampinen, J. (2008). A generator for multimodal test functions with multiple global optima. In Li, X., Kirley, M., Zhang, M., Green, D., Ciesielski, V., Abbass, H., Michalewicz, Z., Hendtlass, T., Deb, K., Tan, K., Branke, J., and Shi, Y., editors, *Simulated Evolution and Learning*, volume 5361 of *Lecture Notes in Computer Science*, pages 239–248. Springer Berlin Heidelberg.

Runarsson, T. (2006). Approximate evolution strategy using stochastic ranking. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B., Coello Coello, C. A., and Runarsson, T. D., editors, *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 745–752. IEEE.

Runarsson, T. P. and Y., X. (2005). Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics. C, Applications Review*, 35(2):233–243, doi:10.1109/TSMCC.2004.841906.

Runarsson, T. P. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, doi:10.1109/4235.873238.

Ruppin, E. (2002). Evolutionary autonomous agents: A neuroscience perspective. *Nature Reviews Neuroscience*, 3(2):132–141, doi:10.1038/nrn729.

Ruta, V., Datta, S. R., Vasconcelos, M. L., Freeland, J., Looger, L. L., and Axel, R. (2010). A dimorphic pheromone circuit in drosophila from sensory input to descending output. *Nature*, 468(7324):686–690, doi:10.1038/nature09554.

Sadollah, A., Bahreininejad, A., Eskandar, H., and Hamdi, M. (2013). Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems. *Applied Soft Computing*, 13(5):2592–2612, doi:10.1016/j.asoc.2012.11.026.

Salcedo-Sanz, S. (2009). A survey of repair methods used as constraint handling techniques in evolutionary algorithms. *Computer Science Review*, 3(3):175–192, doi:10.1016/j.cosrev.2009.07.001.

Sareni, B. and Krahenbuhl, L. (1998). Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97–106, doi:10.1109/4235.735432.

Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, doi:10.1126/science.1165893.

Schraudolph, N. N. (1999). A fast, compact approximation of the exponential function. *Neural Computation*, 11(4):853–862.

Schwefel, H.-P. (1965). Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. Master's thesis, Technische Universität, Berlin, Germany.

Schwefel, H.-P. P. (1993). *Evolution and optimum seeking: the sixth generation*. John Wiley & Sons, Inc., New York, NY, USA.

Seelig, J., Chiappe, M., Lott, G., Dutta, A., Osborne, J., Reiser, M., and Jayaraman, V. (2010). Two-photon calcium imaging from head-fixed drosophila during optomotor walking behavior. *Nature Methods*, 7(7):535–540, doi:10.1038/nmeth.1468.

Shir, O. and Bäck, T. (2006). Niche radius adaptation in the cma-es niching algorithm. In Runarsson, T., Beyer, H.-G., Burke, E., Merelo-Guervós, J., Whitley, L., and Yao, X., editors, *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 142–151. Springer Berlin Heidelberg.

Siarry, P., Pétrowski, A., and Bessaou, M. (2002). A multipopulation genetic algorithm aimed at multimodal optimization. *Advances in Engineering Software*, 33(4):207–213, doi:http://dx.doi.org/10.1016/S0965-9978(02)00010-8.

Singh, G. and Deb, K. (2006). Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 1305–1312, New York, NY, USA. ACM.

Sinha, A., Srinivasan, A., and Deb, K. (2006). A population-based, parent centric procedure for constrained real-parameter optimization. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B., Coello Coello, C. A., and Runarsson, T. D., editors, *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 239–245. IEEE.

Smith, A. E. and Tate, D. (1993). Genetic optimization using a penalty function. In *Proc. 5th Int. Conf. Genetic Algorithms*, pages 499–505.

Smith, L. A., Wang, X., Peixoto, A. A., Neumann, E., Hall, L. M., and Hall, J. C. (1996). A drosophila calcium channel $\alpha 1$ subunit gene maps to a genetic locus associated with behavioral and visual defects. *The Journal of Neuroscience*, 16(24):7868–7879.

Smith, S. L., Gaughan, P., Halliday, D. M., Ju, Q., Aly, N. M., and Playfer, J. R. (2007). Diagnosis of parkinson's disease using evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 8(4):433–447, doi:10.1007/s10710-007-9043-9.

Smith, S. L. and Timmis, J. (2008). An immune network inspired evolutionary algorithm for the diagnosis of parkinson's disease. *Biosystems*, 94(1-2):34–46, doi:10.1016/j.biosystems.2008.05.024.

Sorribes, A., Armendariz, B. G., Lopez-Pigozzi, D., Murga, C., and de Polavieja, G. G. (2011). The origin of behavioral bursts in decision-making circuitry. *PLoS Comput Biol*, 7(6):e1002075, doi:10.1371/journal.pcbi.1002075.

Storn, R. (1999). System design by constraint adaptation and differential evolution. *IEEE Transactions on Evolutionary Computation*, 3(1):22–34, doi:10.1109/4235.752918.

Storn, R. and Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, doi:10.1023/A:1008202821328.

Strogatz, S. (2001). *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry and engineering*. Westview Press, Boulder, CO, USA.

Sun, J. and Garibaldi, J. M. (2010). A novel memetic algorithm for constrained optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2010, pages 549–556.

Takahama, T. and Sakai, S. (2006). Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B., Coello Coello, C. A., and Runarsson, T. D., editors, *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 1–8. IEEE.

Takahama, T. and Sakai, S. (2010). Constrained optimization by the ε constrained differential evolution with an archive and gradient-based mutation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2010, pages 1–9. IEEE.

Takahama, T. and Sakai, S. (2012). Efficient constrained optimization by the ε constrained rank-based differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2012, pages 1–8. IEEE.

Takahama, T. and Sakai, S. (2013). Efficient constrained optimization by the ε constrained differential evolution with rough approximation using kernel regression. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2013, pages 1334–1341. IEEE.

Takahama, T., Sakai, S., and Iwane, N. (2005). Constrained optimization by the ε constrained hybrid algorithm of particle swarm optimization and genetic algorithm. In Zhang, S. and Jarvis, R., editors, *AI 2005: Advances in Artificial Intelligence*, volume 3809 of *Lecture Notes in Computer Science*, pages 389–400. Springer Berlin Heidelberg.

Tasgetiren, M. and Suganthan, P. (2006). A multi-populated differential evolution algorithm for solving constrained optimization problem. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B., Coello Coello, C. A., and Runarsson, T. D., editors, *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 33–40. IEEE.

Tessema, B. and Yen, G. G. (2009). An adaptive penalty formulation for constrained evolutionary optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 39(3):565–578, doi:10.1109/TSMCA.2009.2013333.

Toffolo, A. and Benini, E. (2003). Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation*, 11(2):151–167, doi:10.1162/106365603766646816.

Tsuda, I. (2001). Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems. *Behavioral Brain Sciences*, 24(5):793–809, doi:10.1017/S0140525X01000097.

Tsutsui, S., Fujimoto, Y., and Ghosh, A. (1997). Forking genetic algorithms: Gas with search space division schemes. *Evolutionary Computation*, 5(1):61–80, doi:10.1162/evco.1997.5.1.61.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 49:e433–460.

Ullah, A. S. S. M. B., Sarker, R., and Cornforth, D. (2008). Search space reduction technique for constrained optimization with tiny feasible space. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '08, pages 881–888. ACM.

Ullah, A. S. S. M. B., Sarker, R., Cornforth, D., and Lokan, C. (2007). An agent-based memetic algorithm (ama) for solving constrainted optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2007, pages 999–1006. IEEE.

Ullah, A. S. S. M. B., Sarker, R., Cornforth, D., and Lokan, C. (2009a). Ama: a new approach for solving constrained real-valued optimization problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 13(8–9):741–762, doi:10.1007/s00500-008-0349-1.

Ullah, A. S. S. M. B., Sarker, R., and Lokan, C. (2009b). An agent-based memetic algorithm (ama) for nonlinear optimization with equality constraints. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2009, pages 70–77. IEEE.

Ursem, R. (1999). Multinational evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 3 of *CEC 1999*, pages 1633–1640.

Valente, D., Golani, I., and Mitra, P. (2007). Analysis of the trajectory of drosophila melanogaster in a circular open field arena. *PLoS ONE*, 2(10):e1083, doi:10.1371/journal.pone.0001083.

Von Neumann, J. (1966). *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA.

Wang, Y. and Cai, Z. (2011). Constrained evolutionary optimization by means of $(\mu + \lambda)$-differential evolution and improved adaptive trade-off model. *Evolutionary Computation*, 19(2):249–285, doi:10.1162/EVCO_a_00024.

Wang, Y., Cai, Z., Zhou, Y., and Zeng, W. (2008). An adaptive tradeoff model for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 12(1):80–92, doi:10.1109/TEVC.2007.902851.

Ward, A. B., Sali, A., and Wilson, I. A. (2013). Integrative structural biology. *Science*, 339(6122):913–915, doi:10.1126/science.1228565.

Wessing, S. (2013). Repair methods for box constraints revisited. In *Proceedings of the 16th European Conference on Applications of Evolutionary Computation*, EvoApplications '13, pages 469–478, Berlin, Heidelberg. Springer-Verlag.

While, R. L. and Hingston, P. (2013). Usefulness of infeasible solutions in evolutionary search: An empirical and mathematical study. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2013, pages 1363–1370. IEEE.

Whitley, D. (1989). The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In Schaffer, J. D., editor, *Proceedings of the third international conference on genetic algorithms*, pages 116–121. George Mason University.

Wilson, C. J. and Groves, P. M. (1981). Spontaneous firing patterns of identified spiny neurons in the rat neostriatum. *Brain Research*, 220(1):67–80.

Wineberg, M. and Oppacher, F. (2003). The underlying similarity of diversity measures used in evolutionary computation. In *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation: PartII*, GECCO '03, pages 1493–1504, Berlin, Heidelberg. Springer-Verlag.

Wischmann, S., Floreano, D., and Keller, L. (2012). Historical contingency affects signaling strategies and competitive abilities in evolving populations of simulated robots. *PNAS*, 109(3):864–868, doi:10.1073/pnas.1104267109.

Wolf, F. W., Rodan, A. R., Tsai, L. T.-Y., and Heberlein, U. (2002). High-resolution analysis of ethanol-induced locomotor stimulation in drosophila. *The Journal of Neuroscience*, 22(24):11035–11044.

Yang, X.-S. and Deb, S. (2010). Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation*, 1(4):330–343, doi:10.1504/IJMMNO.2010.03543.

Zhang, G., Cheng, J., Gheorghe, M., and Meng, Q. (2013). A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems. *Applied Soft Computing*, 13(3):1528–1542, doi:10.1016/j.asoc.2012.05.032.

Zhang, W., Yen, G. G., and He, Z. (2014). Constrained optimization via artificial immune system. *IEEE Transactions on Cybernetics*, 44(2):185–198, doi:10.1109/TCYB.2013.2250956.

Zielinski, K. and Laur, R. (2006). Constrained single-objective optimization using particle swarm optimization. In Yen, G. G., Lucas, S. M., Fogel, G., Kendall, G., Salomon, R., Zhang, B.-T., Coello Coello, C. A., and Runarsson, T. P., editors, *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2006, pages 443–450. IEEE.

Zou, D., Liu, H., Gao, L., and Li, S. (2011). A novel modified differential evolution algorithm for constrained optimization problems. *Computers & Mathematics With Applications*, 61(6):1608–1623, doi:10.1016/j.camwa.2011.01.029.

Andrea Maesani - Curriculum Vitae

Education

September 2010 – October 2014
PhD in "Computer, Communication and Information sciences", EPFL, Lausanne

- Enrolled within Laboratory of Intelligent Systems, Prof. Dario Floreano
- Main thesis research on artificial intelligence, computational neuroscience and evolutionary robotics
- Collaborated on research in neurotechnologies for stroke recovery and on soft robotics
- Teaching assistant for the "Mobile Robots" and "Bio-Inspired Artificial Intelligence" class
  - Supervised 10 semester student projects
- Education on entrepreneurship and business creation
  - "CTI Business Creation" (2014), course on start-up creation
  - "Management of Innovation and Technology Transfer" (2013), course on IP management
  - "Venture Challenge" (2012/2013), 6 months business course on start-up creation
  - "Pre-Seed Workshop" (2012), workshop on start-up creation

---

September 2007 – October 2009
Master of Science, Computer Engineering, Politecnico di Milano, Milan, Italy

- Thesis on data management for novel search engine platform within the EU project "Search Computing"
- Semester Projects: "EKG peak detection techniques", "Memory garbage collectors for Java Virtual Machine " and "Emotion classification from biometric data"
- Final Mark: 110/110 cum laude

September 2004 – July 2007
Bachelor of Science, Computer Engineering, Politecnico di Milano, Milan, Italy

- Thesis on object recognition with stereovision-based distance measurement for humanoid robots

Work Experience

January 2010 – August 2010
Software Engineer, Peripheral Systems Laboratory, École Polytechnique Fédérale de Lausanne (EPFL)

- Hardware development of a planar gonio-spectrophotometer
- Printer drivers development for precision printing to support document-security applications

---

November 2009 – December 2009
Software Engineer , Politecnico di Milano

- Java SE/EE design and implementation of software components for the Search Computing project

---

2005-2009
Self-Employed Web Developer

- Freelance consultant for PhP-based web solutions, Web development and Flash Design

Languages

Italian	Native language
English	Professional working proficiency
French	Good working proficiency
German	Basic school proficiency

Skills

Programming Languages/Development: C/C++, Java SE/EE, Python, Matlab, PhP, HTML, JavaScript, SQL
Embedded Systems: Atmel ASF, Impulse C, Arduino
Fabrication: Solidworks, 3D Printing, Soft Polymers, PneuNets, Skills in fast-prototyping
Machine learning and Statistics

Publications – Peer-Reviewed Journals and Conference Proceedings

Evolutionary computation

- A. **Maesani**, P. R. Fernando and D. Floreano. *Artificial Evolution by Viability Rather Than Competition*, in PLOS One, vol. 9, num. 1, p. e86831, 2014
- A. **Maesani**, G. Iacca, and D. Floreano*, Memetic Viability Evolution for constrained optimization*, in review at IEEE Transactions on Evolutionary Computation
- A. **Maesani** and D. Floreano, *Viability Principles for Constrained Optimization Using a (1+1)-CMA-ES*. 13th International Conference on Parallel Problem Solving From Nature, Ljubljana, Slovenia, September 13-17, 2014

Computational biology

- A. **Maesani***, P. Ramdya*, S. Cruchet, K. Gustafson, A. Massouras, B. Deplancke, R. Benton, and D. Floreano, *Neural noise shapes Drosophila behavior*, in revision, * = Equal Contributions
- G. Tamo, A. **Maesani** et al., *Prediction of symmetric protein assemblies without aggregating energy functions with geometric restraints*, in preparation

Soft robotics

- J. M. Germann, A. **Maesani**, R. Pericet Camara and D. Floreano. *Soft Cells for Programmable Self-Assembly of Robotic Modules*, accepted in journal "Soft Robotics", 2014
- J. M. Germann, A. **Maesani**, M. Stöckli and D. Floreano. *Soft Cell Simulator: A tool to study Soft Multi-Cellular Robots*. IEEE International Conference on Robotics and Biomimetics, Shenzhen, China, December 12-14, 2013

Various

- J. Auerbach, D. Aydin, A. **Maesani** et al., *RoboGen: Robot Generation through Artificial Evolution*, Artificial Life 14, New York, NY, USA, July 30-August 2, 2014
- T. Bugnon, A. **Maesani** and R. D. Hersch. *Enhancing the Specular Effect of Metallic Color Prints by Reducing the Use of Yellow Ink*, in Journal of Imaging Science and Technology, vol. 55, 2011
- A. Campi, S. Ceri, A. **Maesani**, and S. Ronchi, *Designing Service Marts for Engineering Search Computing Applications*, ICWE, 2010

Publications - Patents

- Co-inventor of : Neuroprosthetic system restoring upper limb function through coordinated electrical stimulation, 2013, Provisional Patent Application (PCT/IB2014/065417)
- Co-inventor of : Non-invasive drawable electrodes for transcutaneous electrical stimulation or biological signal sensing, 2014, Provisional Patent Application (PCT filing expected Dec 2014)

Other Activities

- **Sports:** Swimming, Running, Snowboarding
- **Other hobbies:** Reading, Travelling, Electronics DIY