



Deposited via The University of Sheffield.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/93028/>

Version: Accepted Version

Article:

Moraglio, A. and Sudholt, D. (2015) Principled Design and Runtime Analysis of Abstract Convex Evolutionary Search. *Evolutionary Computation*. ISSN: 1063-6560

https://doi.org/10.1162/EVCO_a_00169

Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Principled Design and Runtime Analysis of Abstract Convex Evolutionary Search*

Alberto Moraglio

Department of Computer Science, University of Exeter, Exeter EX4 4QF, UK

a.moraglio@exeter.ac.uk

Dirk Sudholt

Department of Computer Science, University of Sheffield, Sheffield S1 4DP, UK

d.sudholt@sheffield.ac.uk

Abstract

Geometric crossover is a formal class of crossovers which includes many well-known recombination operators across representations. In previous work, it was shown that all evolutionary algorithms with geometric crossover (but with no mutation) do the same form of convex search regardless of the underlying representation, the specific selection mechanism, offspring distribution, search space, and problem at hand. Furthermore, it was suggested that the generalised convex search could perform well on generalised forms of concave and approximately concave fitness landscapes, regardless of the underlying space and representation. In this article, we deepen this line of enquiry and study the runtime of generalised convex search on concave fitness landscapes. This is a first step towards linking a geometric theory of representations and runtime analysis in the attempt to (i) set the basis for a more general, unified approach for the runtime analysis of evolutionary algorithms across representations, and (ii) identify the essential matching features of evolutionary search behaviour and landscape topography that cause polynomial performance. We present a general runtime result that can be systematically instantiated to specific search spaces and representations, and present its specifications to three search spaces. As a corollary, we obtain that the convex search algorithm optimises LEADINGONES in $O(n \log n)$ fitness evaluations, which is faster than all unbiased unary black-box algorithms.

1 Introduction

The Evolutionary Computation (EC) field needs unification and systematization in a rational framework to survive its own success (De Jong [5]).

The various flavours of evolutionary algorithms (EAs) look very similar when cleared of algorithmically irrelevant differences such as domain of application, phenotype interpretation and representation-independent algorithmic characteristics that, in effect, can be freely exchanged between algorithms, such as the selection scheme. Ultimately, the origin of the differences of the various flavours of evolutionary algorithms is rooted in the solution representation and relative genetic operators.

Are these differences only superficial? Is there a deeper unity encompassing all evolutionary algorithms beyond the specific representation? Formally, is a general mathematical framework that unifies search operators for all solution representations at all possible? Would such a general framework be able to capture essential properties

*A preliminary version of this paper with parts of the results was published at GECCO 2012 [22]. This extended version contains new results about the optimal choice of the population size in Convex Evolutionary Search on binary spaces in Section 6.

encompassing all EAs or would it be too abstract to say anything useful? These are important, difficult open research questions which the present work starts attacking.

A number of researchers have been pursuing EC unification across representations. However, so far, no one has been able to build a fully-fledged theory of representations. Radcliffe pioneered a unified theory of representations [26]; Poli unified schema theories for traditional genetic algorithms and genetic programming [14]; Stephens suggested that all evolutionary algorithms can be unified using dynamical systems and coarse graining [30]; Rothlauf initiated a popular but less formal theory of representations [28]; Rowe et al., building upon Radcliffe's work, have devised a theory of representations based on group theory [29]; Stadler et al. built a theory of fitness landscapes that connects with representations and search operators [27]; Mitavskiy devised category theoretic methods to compare rigorously evolutionary algorithms with different representations [17].

For the No Free Lunch (NFL) theorem [38], no search algorithm is better than any other on all fitness landscapes. This implies that any non-futile theory of search algorithms that aims at proving performance better than random search has to focus on a restricted class of search algorithms and on a corresponding "well-matched" restricted class of fitness landscapes. Matching classes of search algorithms and fitness landscapes is an important open problem in the field.

The class of evolutionary algorithms (EAs) with geometric crossover [20] encompasses and formalises many well-known representation-specific EAs [18]. In recent work [19], Moraglio showed that this class of algorithms (without mutation) does a generalised (or abstract) form of convex search. The unity in behaviour of EAs across representations calls for a single class of fitness landscapes well-matched to them all – a matching that goes beyond the specific representation. Well-matched "topographic features" of fitness landscapes are those essential features that can be exploited by the common "behavioural features" characterising this class of search algorithms, and produce good search performance. Intuition on continuous spaces suggests that convex search may be well-matched with functions which have a globally concave trend (when maximising). There is some formal evidence that this intuition is correct and that it extends naturally to general metric spaces [19].

Runtime analysis is the standard approach to analytically analyse algorithmic performance. In the last decade it has been applied, with an ever increasing success, to randomised search heuristics and it is establishing itself as a leading theory [2, 10, 24]. Despite its success, it has been difficult to analyse EAs with populations and crossover, although there are some results. Crossover was shown to be able to find optima located within the convex hull of suitable local peaks [11–13]. It was demonstrated how crossover speeds up search by combining features, or building blocks, of good solutions [6, 33]. Furthermore, crossover proved beneficial for combinatorial problems including colouring problems [31] and the all-pairs shortest path problem [7].

Analyses like these are done on a per-algorithm-and-per-problem basis, and approaches that have worked on a algorithm-problem pair do not usually transfer to others. Few general mathematical tools to do runtime analysis have been developed (e.g., drift analysis [8, 9, 15] and fitness levels [4, 34]). However, also in this case, their application is specific for an algorithm-problem pair and requires a great deal of effort and ingenuity to understand how the general tool can be applied to the specific case. A more general and systematic approach to determine the performance of a *class* of algorithms on a *class* of problems would be a major progress.

In this article, we start studying the runtime of a generalised form of convex search on concave fitness landscapes. This is a first step towards linking a geometric theory of representations and runtime analysis in the attempt to set the basis for a more general and unified approach for the runtime analysis of evolutionary algorithms across representations.

We strip to the bare essentials both the specific class of evolutionary algorithms and the matching class of problems considered. Whereas both algorithms and problems are not by themselves of practical interest or relevance, they allow us to identify the essential matching features of evolutionary search behaviour and landscape topography that cause polynomial performance, on problems on which Pure Random Search (PRS) runs in exponential time. Interestingly, their simplicity allows us to draw tight links with the work on Pure Adaptive Search (PAS) [25], an *ideal* algorithm which has been studied since the eighties and which presents an exponential speed-up on PRS on almost all problems. The main open challenge about PAS is finding how to implement it efficiently.

The remainder of the paper is structured as follows. Section 2 describes the general framework of abstract convex evolutionary search. Section 3 reports the details of the specific representation-independent search algorithm, class of fitness landscapes and performance measure used subsequently in the runtime analysis. Section 4 describes the general Pure Adaptive Search algorithm and shows that it is efficient on the class of fitness landscapes considered. Section 5 proposes a general runtime result for convex search on quasi-concave landscapes that holds across spaces and representations. It makes explicit the link with Pure Adaptive Search, and shows how the general result can be specialised to determine the runtime of convex search on quasi-concave landscapes for three specific spaces. Section 6 shows that the population size is a crucial parameter of the search algorithm and determines population sizes that optimise the runtime. It reports also computational experiments on `LeadingOnes` to complement the theory. Section 7 discusses how the results presented here could be extended, in the future, to EAs in actual use and to more realistic fitness landscapes, making this line of theory potentially relevant to practice.

2 Abstract Convex Evolutionary Search

We give some necessary definitions and previous results¹ from [19]. For each definition, we give the general version for a generic metric space, and the corresponding definition specialised to the Hamming distance on binary strings obtained by substituting the Hamming distance in the general definition, and rephrasing the outcome in terms on representation-specific language, i.e., binary strings, bits and schemata.

2.1 Abstract Convexity

A metric is a generalization of the notion of distance. A metric space is a set X with a distance function $d : X \times X \rightarrow \mathbf{R}$ that, for every two points x and y in X , gives the distance $d(x, y)$ between them. A metric space must satisfy the following axioms for any $x, y, z \in X$:

1. $d(x, y) \geq 0$ and $d(x, y) = 0$ iff $x = y$
2. $d(x, y) = d(y, x)$

¹The framework in this section may look related to the field of Convex Optimisation, but it differs fundamentally from it. See [19] for a discussion of the differences.

$$3. d(x, z) + d(z, y) \geq d(x, y)$$

Metric spaces arise naturally from graphs. The distance between two points (i.e., nodes) in the graph is defined as the length of the shortest path between them. Any so defined distance meets the metric axioms. The Hamming distance (HD) on binary strings is associated with a hypercube graph (i.e., its neighbourhood structure), hence it is a metric.

Given a metric space $M = (X, d)$ the *line segment* between x and y , termed *extremes*, is the set $[x, y]_d = \{z \in X \mid d(x, z) + d(z, y) = d(x, y)\}$. For example, the string 0001 is in the Hamming segment $[0101, 1001]_{HD}$ as $HD(0101, 0001) + HD(0001, 1001) = HD(0101, 1001)$.

In the Euclidean space, a set is convex iff the line segment connecting any two points in the set lies entirely in the set. We can generalise the notion of convex set to metric spaces as follows. The *abstract geodesic convexity* [36] \mathcal{C} on X induced by the metric space $M = (X, d)$ is the collection of subsets of X that are geodesically convex, where a subset C of X is geodesically convex provided $[x, y]_d \subseteq C$ for all x, y in C . An important property of *any* abstract geodesic convexity is that it is closed under intersection, i.e., the intersection of geodesically convex sets is a geodesically convex set. We can now generalise the notion of convex hull to metric spaces. The *metric convex hull* of a subset A of X is the smallest (geodesically) convex set that includes A , i.e., $co(A) = \bigcap \{C \mid A \subseteq C \in \mathcal{C}\}$.

In the Hamming space, the notions of segment, convex set and schema coincide², as we illustrate in the following. Let $H(a, b)$ be the schema obtained from the binary strings a and b by position-wise inserting a '*' symbol where they mismatch and inserting the common bit otherwise (e.g., $H(0101, 1001) = **01$). By abuse of notation, we consider a schema as being both a template and the set of strings matching the template. The binary string c is in the Hamming segment between the binary strings a and b iff c matches the schema $H(a, b)$ (e.g., 0001 is in the segment $[0101, 1001]$ as verified earlier using the definition of segment, and it matches the schema $**01$). Every segment in the Hamming space is convex, because for $c, d \in H(a, b)$ the schema $H(c, d)$ can be obtained by changing some of the '*' symbols in $H(a, b)$ to 0 or 1, hence it is more specific than $H(a, b)$ (i.e., $H(c, d) \subseteq H(a, b)$, hence $[c, d] \subseteq [a, b]$). Every schema is a convex set as it corresponds to a segment between some pair of binary strings belonging to it. Every convex set is a schema because the set of all Hamming segments form the convexity structure on the Hamming space, as it is the convexity on the product space arising from the product of the convexities on each dimension (i.e., product convexity) induced by the trivial metric [36]. Consequently, the intersection of two schemata is a schema or the empty set (e.g., $**101 \cap 1**01 = 1*101$) and the convex hull of a set of binary strings is the smallest schema (the schema matching the minimum number of strings) matching all of them (e.g., $co(0101, 1001, 0000) = **0*$).

2.2 Formal Evolutionary Algorithm and Abstract Convex Evolutionary Search

Definition 1. (*Geometric crossover* [20]) *A recombination operator is a geometric crossover under the metric d if all offspring are in the d -metric segment between its parents.*

This definition is *representation-independent*, hence well-defined for any representation, as it depends on the underlying specific representation only indirectly via the metric d which is defined on the representation.

²The fact that convex sets in the Hamming space correspond to schemata is an indication that this theory shares common ground with older ideas about how crossover works. The geometric language however allows us to talk about crossover in much greater generality, without specific reference to the underlying representation.

Let us consider a simple example on binary strings with one-point crossover. With parent strings 0101 and 1001 and crossover point between first and second position, we obtain the offspring string 0001, which is in the Hamming segment between its parents. As this is true for any choice of parent strings and for all their offspring, one-point crossover is a geometric crossover w.r.t. the Hamming distance. The class of geometric crossover is really broad and extends far beyond this simple example [18]. For vectors of reals, various types of blend or line crossovers are geometric crossovers under Euclidean distance, and box recombinations and discrete recombinations are geometric crossovers under Manhattan distance. For binary and multary strings, all mask-based crossovers are geometric under Hamming distance. For permutations, PMX and Cycle crossover are geometric under swap distance and merge crossover is geometric under adjacent swap distance; other crossovers for permutations are also geometric. For genetic program trees, the family of homologous crossovers is geometric under structural Hamming distance. For biological sequences, various homologous recombinations that resemble more closely biological recombination at molecular level (as they align variable-length sequences on their contents, rather than position-wise, before swapping genetic material) are geometric under Levenshtein distance. Recombinations for several more complex representations are also geometric. Notice however that the class of geometric crossover does not fully exhaust the range of crossover operators in common use. For example, sub-tree swap crossover for genetic program trees is provably not a geometric crossover under any metric.

Geometric crossover and geometric mutation can be understood as functional forms taking the distance function d as a parameter. Therefore, we can see an evolutionary algorithm using these geometric operators as a function of the metric d too. That is, d can be considered as a parameter of the algorithm like any others, such as the mutation rate. However, notice the difference in the complexity of the objects passed as parameter: the mutation rate parameter takes values in the interval $[0, 1]$, that is, it is a simple real number, whereas the metric parameter takes values in the set of metrics, that is, it is a whole space.

We can now look at an evolutionary algorithm as a function of the distance d from an abstract point of view. To do this, we do not consider any metric in particular and we treat an evolutionary algorithm using geometric operators as a formal specification of a representation and space independent algorithm with a well-defined formal semantic arising from the metric axioms only. The transition to this more general point of view is analogous to the transition from geodesic convexity with respect to a specific metric space to the notion of abstract geodesic convexity. We refer to an evolutionary algorithm seen according to the latter interpretation as a *formal evolutionary algorithm*. A different notion of formal search algorithm based on equivalence classes was introduced by Radcliffe and Surry [35].

Normally, an algorithm can actually be run only when all its parameters have been assigned a value. We call an algorithm with all its parameters specified, a fully-specified algorithm. However, a formal model of the algorithm can be used to infer the behavior of a partially specified algorithm in which some parameters are left unspecified. In other words, using a formal model one can “run” a partially specified algorithm and infer its abstract behavior, i.e., those behavioral properties common to all specific behaviors obtained by assigning all possible specific values to the parameter left unspecified. We term *abstract evolutionary search* the behavior of a formal evolutionary algorithm in which the underlying metric d is unspecified. As this behavior is inferred

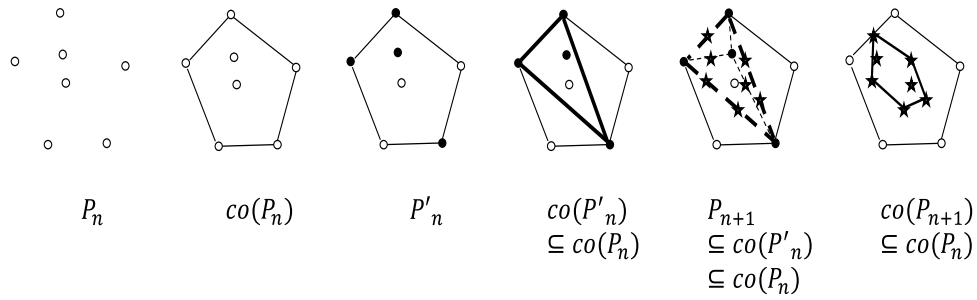


Figure 1: Convex evolutionary search.

from the formal evolutionary algorithm and the metric axioms only, it is the behavior of the formal evolutionary algorithm on all possible search spaces and associated representations.

The abstract behavior of a formal evolutionary algorithm is an formal object itself based on the metric axioms. In the following sections, we will show that the behavior of a formal evolutionary algorithm can be profitably described axiomatically using the language of abstract convexity.

Regardless of the specific metric space and associated representation, and of the specific fitness landscape considered, all evolutionary algorithms with geometric crossover (and selection and replacement) but with no mutation do a form of *convex search* (see Figure 1). The term $co()$ denotes the metric convex hull operator, and P_n , P'_n and P_{n+1} are the current population, the set of selected parents, and the offspring population, respectively. Figure 1 shows that the convex hull of the population of offspring is always included in the convex hull of the population of parents. So, the overall search forms a nested chain of convex hulls of populations reducing in size with time. Specifically for the Hamming space, this result says that once a bit is fixed in a population to a certain value (i.e., 0 or 1) that bit will stay fixed in all subsequent populations, or equivalently, that the smallest schemata spanning subsequent populations are increasingly more specific (they can only lose ‘*’ symbols).

2.3 Concave Fitness Landscapes

On the Euclidean space, intuition suggests that convex search may do well on (approximately) concave functions when maximising³ (see Figure 2). This is because the population of offspring in the convex hull of the parent population is likely to have a better fitness on functions with this shape.

Various traditional notions of concave function can be generalised to general metric spaces, hence to combinatorial spaces associated with any representation, by replacing the notion of (Euclidean) segment with that of metric segment in the original definitions. However, caution is needed, as the resulting generalisation may not be suitable for combinatorial spaces (only degenerate landscapes on combinatorial spaces may be encompassed by the definition, e.g., only flat landscapes). The following well-behaved generalisations were proposed in [19]:

Average-concave landscapes for all $x, y : z \sim U([x, y])$, $E[f(z)] \geq (f(x) + f(y))/2$.

Quasi-concave landscapes for all $x, y : z \in [x, y]$, $f(z) \geq \min(f(x), f(y))$.

³In this work, we assume maximisation on concave landscapes. Analogous considerations hold unchanged for minimisation on convex landscapes.

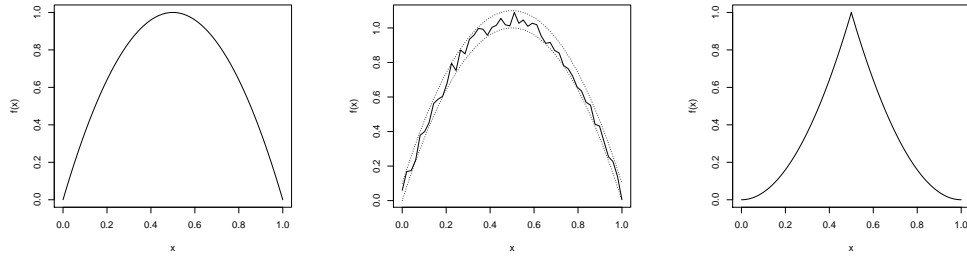


Figure 2: Examples of concave function (left) ϵ -approximately concave function (middle) and quasi-concave function which is not concave (right) on the real line.

They specialise to the space of binary strings with the Hamming distance by considering the Hamming segment in their definition. It can be shown that the ONE-MAX problem is average-concave and average-convex (i.e., it is average-affine), and the LEADINGONES problem is quasi-concave.

Adding an ϵ -bounded perturbation function to the above definitions, we obtain *approximately average-concave* and *approximately quasi-concave* landscapes: $E[f(z)] \geq (f(x) + f(y))/2 - \epsilon$ and $f(z) \geq \min(f(x), f(y)) - \epsilon$. Any function is approximately average/quasi concave for ϵ large enough.

In [19] it was shown that, *regardless of the specific metric, on average/quasi concave landscapes, convex search with geometric crossover and selection produces steady improvements*: for any population, the average/worst fitness of the next population is in expectation/deterministically no worse than the average/worst fitness of the given population (even without selection). This result degrades gracefully as the landscape becomes less concave (for increasing ϵ). Whereas this result in general implies neither convergence to the optimum nor efficient performance, it is a strong one-step performance result that holds across all representations.

In the remainder, we investigate the possibility of a general runtime analysis of evolutionary algorithms on concave landscapes *across representations* starting from the result above. Is a general result possible *regardless of the underlying space*? Does convex search on concave landscapes have exponentially better runtime than random search *regardless of the underlying space*? If not, what are the features of the underlying space that critically affect the runtime, making it change from polynomial to exponential? In order to start answering these questions we need further assumptions on the specific algorithm, landscape and performance measure presented in Section 3.

2.4 Abstract-to-Specific Interface

Figure 3 illustrates the envisioned functional relationship between search algorithm (SA), fitness landscape (FL), search behavior (SB) and search performance (SP) and their abstract counter-parts, formal search algorithm (FSA), abstract fitness landscape (AFL), abstract search behavior (ABS) and abstract search performance (ASP). The horizontal arrows in the bottom means that the algorithm SA is fed with the parameter fitness landscape FL which when run together give rise to the search behavior SB that produces the search performance SP. The horizontal arrows in the top mirror those in the bottom and depict analogous relations at an abstract level in which the underlying

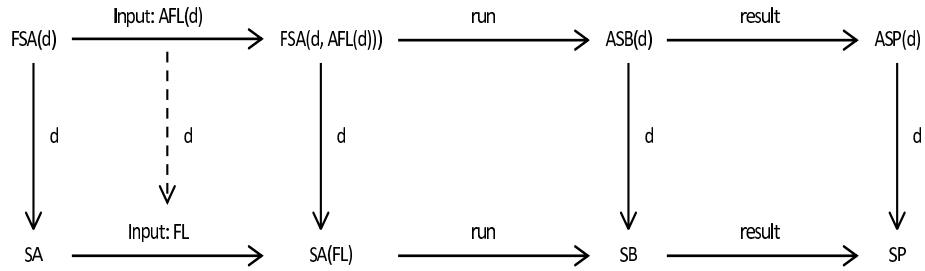


Figure 3: Functional relations between search algorithm, fitness landscape, search behavior, search performance and their abstract counter-parts.

distance d is left unspecified. The vertical arrows relate abstract and concrete levels by functional application of the functional forms in the top with a specific distance d .

3 Algorithm, Landscape and Performance Measure

We strip to the bare essentials both the specific class of EAs and the matching class of problems considered. This will allow us to pinpoint the essential mechanism that links matching features of evolutionary search behaviour and landscape topography to polynomial performance, on problems on which random search runs in exponential time.

3.1 Convex Search Algorithm

Convex Evolutionary Search is the general class of search algorithms in which the offspring population is in the convex hull of the parent population. For the analysis, the search algorithm considered is the Convex Search Algorithm (CSA) (Algorithm 1), which does a particular type of Convex Evolutionary Search. This is an *abstract and representation-independent* algorithm as it is well-defined over any metric space, as all its algorithmic elements are well-defined on any metric space. The *metric convex hull uniform recombination* returns an offspring sampled uniformly at random from the (metric) convex hull formed by its parents. This recombination operator is a multi-parental recombination operator which like geometric crossover performs a convex search when used in a EA without mutation, regardless of the specific underlying space and representation.

The specific convex hull recombination for binary strings with Hamming distance can be formally obtained by plugging in the Hamming distance in the general definition of metric convex hull recombination. It can then be turned into an operational (i.e., algorithmic) definition by re-writing the operator in terms of manipulation of the underlying representation (binary strings). This turns the geometric description into a description of how to act on the parent binary strings to obtain the offspring. These two are different descriptions of the same operator. The operational description of the uniform convex hull recombination for the Hamming space is shown in Algorithm 2.

The (abstract) Convex Search Algorithm can be formally specified to the Hamming space by replacing the Metric Uniform Convex Hull Recombination with the Binary Uniform Convex Hull Recombination, obtaining the Binary Convex Search Algorithm. This is a variant of Genetic Algorithm with Gene Pool Recombination, with a recomb-

Algorithm 1 Convex Search Algorithm

```

1: Input:  $\mu$ : population size
2: Output: individual in the last population
3:
4: Initialise Population Uniformly at Random
5: while Population has not converged to the same individual do
6:   Rank individuals on fitness
7:   if there are at least two fitness values in the current population then
8:     remove all individuals with the worst fitness
9:   end if
10:  Apply  $\mu$  times Metric Convex Hull Uniform Recombination to the remaining
    individuals to create the next population
11: end while
12: return any individual in the last population

```

Algorithm 2 Binary Uniform Convex Hull Recombination

```

1: for each position in the binary strings do
2:   if all parents have 1 or 0 at that position then
3:     the offspring has 1 or 0 at that position, respectively
4:   else
5:     (if there is at least one 1 and at least one 0 at that position)
6:     the offspring has 1 or 0 at that position with probability 0.5
7:   end if
8: end for

```

nation which does not depend on the frequency of 0s and 1s at each position, and with a particular type of truncation selection.

3.2 Quasi-Concave Landscapes

In section 2, we have considered two types of generalisations of concave landscapes, average-concave and quasi-concave, and their approximately concave extensions. These landscapes are “well-matched” with Convex Evolutionary Search *at an abstract level* because there the average/worst fitness of the offspring population is not worse than the average/worst fitness of the parent population *irrespective of the underlying specific metric and representation*. In this work, we focus on quasi-concave landscapes leaving the analysis of the other notions of concave landscapes to future work.

We define (*canonical*) *fitness-level sets* as follows. Assume that the codomain of the fitness function is finite⁴, with values $a_0 < a_1 < \dots < a_q$. Then for $0 \leq i \leq q$ we define the level set L_i as $\{x \in S: f(x) \geq a_i\}$ (slightly abusing the term *level set* as L_i includes higher levels). All these level sets form a nested chain of sets by construction: $L_0 \supseteq L_1 \supseteq \dots \supseteq L_q$. In particular, $L_0 = S$. The sets $(L_0 \setminus L_1), (L_1 \setminus L_2), \dots, L_q$ form a *fitness-level partition* in the sense defined by Wegener [37].

The following properties hold for quasi-concave landscapes irrespective of the underlying metric and representation:

1. If f is quasi-concave: for all sets $C \subseteq S$, if $z \in \text{co}(C)$ then $f(z) \geq \min\{f(x) \mid x \in C\}$.

⁴This assumption, which is important for the runtime analysis presented later, makes this theory inapplicable to continuous spaces as they have a continuous codomain which comprises infinitely many values.

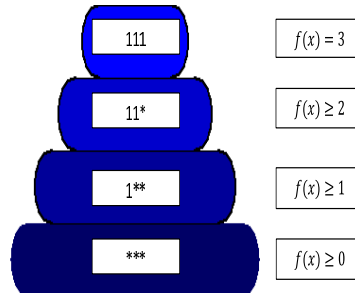


Figure 4: Level sets of `LeadingOnes`.

2. A landscape f is quasi-concave iff all level sets are (metric) convex sets.

A direct consequence of the first property is that the convex hull recombination on quasi-concave landscapes returns offspring whose fitness is not worse than those of any of its parents, for any underlying metric and representation.

The second property allows us to characterise quasi-concave landscapes constructively: a landscape is quasi-concave iff it is a “Tower of Hanoi” of convex sets (see Figure 4 for a specific example). This property is interesting when considered together with combinatorial spaces because they give rise to discrete sets of fitness values. This allows us to check quasi-concavity of a landscape by checking the convexity of all level sets one by one. Also, we can use this property to construct any quasi-concave function for any representation and metric space.

We illustrate this for the Hamming space on binary strings. The idea is to iteratively build a nested chain of convex level sets and subsequently increase fitness values for inner sets. We start from the largest level set and assign the same minimum fitness to all points. We then choose a convex subset of the current fitness level as next fitness level and increase the fitness of all points therein. The process is iterated by further dividing the next fitness level, and so on.

For example, for any x_0 matching the schema `*****`, we initially assign, say, $f(x_0) := 0$. Consider the convex subset `0*****` of `*****` and let us assign to any x_1 matching this schema fitness values of 1, i.e., $f(x_1) := 1$. So, the quasi-concave function we will obtain has fitness 0 for any string starting with 1 and fitness strictly larger than 0 for any string starting with 0. Now consider the fitness level `0*****`. We choose the convex subset `01**1*` of `0*****` to have fitness value 2, and then choose a further convex subset in `01**1*`, and so on. The procedure ends when a convex set with a single element is reached, or earlier, so obtaining a landscape with a plateau. As the `LEADINGONES` landscape can be built in this way as illustrated in Figure 4, it is a quasi-concave landscape. It is possible to show that `ONEMAX` is not a quasi-concave landscape because its level sets are Hamming balls, which unlike Euclidean balls, are not geodesically convex.

The class of quasi-concave landscapes for the Hamming space on binary strings can be also completely characterised⁵ as follows. Let w_1, \dots, w_n be a set of non-negative weights, p a permutation of size n and t a target binary string. We can now construct a weighted generalised version of `LeadingOnes` in which any string with exactly k bits

⁵This class may seem quite narrow. Notice, however, that the generality of the theory is due to the fact that it applies across search spaces and representations, not because it covers a large landscape class when instantiated to a specific space.

set correctly w.r.t. the target t whose positions are specified by the first k entries of p has fitness $w_1 + \dots + w_k$.

Quasi-concave landscapes are a rather nice class of landscapes, potentially solvable efficiently. However, it is easy to see that the NEEDLE landscape belongs to this class. This shows that this class, in the worst case, is intractable by any search algorithm that does not know a priori information about the position of the needle. What makes NEEDLE intractable is that convex fitness levels are too few and shrink too fast. Therefore we restrict the class of quasi-concave landscapes as follows. A *polynomial quasi-concave landscape* is a quasi-concave landscape with:

1. the number $q + 1$ of fitness levels is polynomial in n (problem size, $n = \log(|S|)$).
2. the rate between sizes of successive fitness levels is an inverse polynomial: $r := \min\{|L_{i+1}|/|L_i| \mid 0 \leq i < q\} \geq 1/\text{poly}(n)$.

A polynomial quasi-concave landscape has two characteristic parameters q and r . It is easy to verify that LEADINGONES is a polynomial quasi-concave landscape (it has n fitness levels, and for each level the area of the next level is half of the area of the previous level), whereas NEEDLE is a quasi-concave landscape but not polynomial quasi-concave (it has only two fitness levels, but the area of the second level is exponentially smaller than that of the first level).

3.3 Performance Measures

We say that the Convex Search Algorithm has converged when the whole population contains copies of the same search point. As the algorithm does not always converge to the optimum, we are interested in estimating the probability that the algorithm converges to a global optimum (PC). We also look at the runtime conditional on convergence (RT). A lower bound on PC and a deterministic upper bound on RT give an upper bound on the expected time a global optimum is found when restarts are used. Restarting Convex Search Algorithm after RT generations yields an upper bound of RT/PC on the expected total number of generations.

The number of function evaluations is by a factor of μ larger than the number of generations. The former might be a more fair performance measure as it excludes the possibility that a polynomial number of generation is achieved by means of an exponentially large population. So we are most interested in settings where $\mu \cdot \text{RT}/\text{PC}$ is polynomial.

The class of search algorithms and fitness landscapes considered are well-defined for any underlying metric space and associated representation. In their definitions, the underlying space appears as a parameter, so the class of search algorithms and fitness landscapes can be considered as functional forms that can be instantiated to a specific space by functional application to it. At an abstract level, we could imagine that an abstract search algorithm is “run” on an abstract fitness landscape *before* specifying the underlying specific space in both algorithm and landscape, obtaining a performance that is itself a functional form on the underlying space, i.e., in which the underlying space appears as a parameter. This functional way of interfacing general and specific results allows us to separate neatly the part of the analysis which is valid for any underlying space and associated representation (i.e., a general theory essentially resulting from matching the abstract notions of convexity of the search algorithm and the concavity of the landscape), and pinpoint which particular features of the underlying space and representation have an effect on the performance. The space-specific values of these features can then be measured and plugged in the general expression of the

runtime result, as any other parameter, to obtain the specific runtime for the specific space. This way of shaping a theory has the benefit of systematising and automatising the runtime analysis for any new space and associated representation.

4 Pure Adaptive Search

Pure Adaptive Search (PAS) (see Algorithm 3) is an ideal algorithm, which is in general not implementable efficiently, that has been studied analytically since the 80s in the field of Global Optimisation. This search algorithm requires a search operator able to sample offspring uniformly at random in the level set corresponding to the fitness of the best solution found so far. As Pure Random Search (PRS), which at each iteration samples offspring uniformly at random in the entire search space, the performance of PAS does not depend on the structure of the space S but only on the frequency distribution of the values in the range of the objective function f . This makes PAS a representation-independent algorithm, well-defined on any metric space and across representations. PAS has a well-developed general theory [39], which essentially says that on almost all functions, PAS is exponentially better than PRS. This result is conceptually interesting because it shows that the two simple ingredients of (i) getting at each step better offspring, and (ii) getting them uniformly distributed in the improving set are *sufficient conditions* to produce an exponential speed up on random search performance, very generally, on any search space and virtually on any problem. Interestingly, the result above holds also for a number of relaxations of PAS which are closer to implementable algorithms, e.g., Hesitant Adaptive Search [39]. In this algorithm offspring can land in the improving fitness level only with a certain probability, and the probability distribution of the offspring may deviate from uniformity (in some restricted ways).

Algorithm 3 Pure Adaptive Search

- 1: Pick an initial point X_0 uniformly at random.
 - 2: $i = 0$
 - 3: **while** optimum not found **do**
 - 4: Generate X_{i+1} uniformly at random on the level set $S_i = \{x : x \in S, f(x) \geq f(X_i)\}$ (improving set).
 - 5: $i = i + 1$
 - 6: **end while**
-

The big challenge about PAS is being able to find efficient implementations of it. Looking for general efficient implementations of PAS working on any problem is a chimera, as it would imply $RP = NP$ (where RP is the randomized polynomial time complexity class). So, it makes sense to look for specific efficient implementations of PAS when applied to restricted classes of problems. This objective can be approached the other way around: starting from a randomised search heuristic and a class of problems and showing that the search heuristic can be seen as an implementation of PAS for that class of problems. This would at the same time (i) give a general and fundamental explanation of why that specific search heuristic is efficient on the class of problems considered (i.e., because it can be reduced to PAS) (ii) show that PAS is implementable efficiently for a specific class of problems (i.e., because the specific search heuristic can be implemented efficiently). In this work, we will show that convex search on concave landscapes is essentially an implementation of PAS, linking for the first time the PAS framework to Evolutionary Algorithms.

The following theorem gives the runtime of PAS and PRS on the particular class of function considered in this study.

Theorem 2. Consider fitness levels L_0, \dots, L_q and let $r := \min\{|L_{i+1}|/|L_i| \mid 0 \leq i < q\}$ as well as $s := \max\{|L_{i+1}|/|L_i| \mid 0 \leq i < q\}$.

The expected running time of PAS is at most q/r . In particular, the expected running time of PAS on any polynomial quasi-concave landscape is polynomial.

The expected running time of PRS is at least s^{-q} . If $s = 1 - \Omega(1)$ and $q = n^{\Omega(1)}$ this time is exponential.

Proof. The expected runtime of PAS can be upper bounded by $1/\Pr(L_1 \mid L_0) + 1/\Pr(L_2 \mid L_1) + \dots + 1/\Pr(L_q \mid L_{q-1})$. As each term is at most $1/r$, we get an upper bound of q/r . For any polynomial quasi-concave landscape we have $q \leq \text{poly}(n)$ and $1/r \leq \text{poly}(n)$, hence we get a polynomial upper bound.

The hitting probability of PRS can be written as a product of conditional probabilities: $\Pr(L_0) \cdot \Pr(L_1 \mid L_0) \cdot \dots \cdot \Pr(L_q \mid L_{q-1})$, which is upper bounded by s^q . The expected hitting time of PRS is hence at least s^{-q} . \square

5 Runtime Analysis of Convex Search Algorithm

In this section we first propose a general runtime result for the convex search algorithm on quasi-concave landscapes that holds across spaces and representations. This result applies in principle to *any* metric space and representation. In order to illustrate its applicability, we show how the general result can be specialised to determine the runtime of convex search on quasi-concave landscapes for three specific spaces: Boolean spaces endowed with the Hamming distance, the space of integer vectors endowed with the Hamming distance, and the space of integer vectors endowed with Manhattan distance.

Before presenting the formal statement and proof of the general result, we give an informal description of the idea. The reasoning to determine the runtime of a successful convex search is as follows (see also Figure 5). The initial population comprises a number of points sampled uniformly at random from the search space (i.e., L_0). Then selection is applied which removes all points in the population with the worst fitness value (i.e., those in $L_0 \setminus L_1$). The remaining points are uniformly distributed at random on L_1 as this is a subset of L_0 (by rejection sampling).

If the convex hull of these selected points covers L_1 completely, the convex hull recombination of these points generates the offspring population uniformly at random on L_1 . The mentioned condition is met with high probability on combinatorial spaces provided the population size is sufficiently large⁶. Then the cycle is repeated, until the optimum has been found. Assuming that the convex hull of the selected points at a previous level always covers the next level (or, in rare cases, a further one), the algorithm always conquers a new fitness level at each iteration. Then, the runtime of the algorithm is bounded by the number of non-optimal fitness levels, q . As the algorithm performs uniform sampling on each traversed fitness-level set, in such a successful run, this makes the link to PAS explicit.

Note that the only space-specific element in this reasoning is the (worst-case) probability that a convex set in the specific space is covered by the convex hull of a number of points sampled uniformly at random in the convex set. This probability is important

⁶On continuous spaces, this condition is never met. However, this condition is sufficient and not necessary to guarantee that the optimum is reached. So, this does not necessarily imply that the convex search algorithm cannot be efficient on continuous spaces.

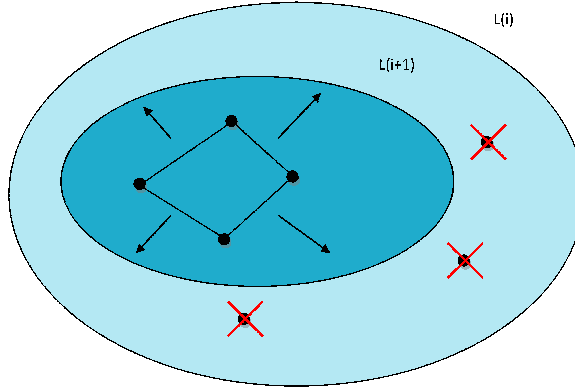


Figure 5: Convex Search Algorithm dynamics on level sets.

as it affects the required size of the population for the optimum to be reached with high probability, and it can be passed as the only required space-specific parameter to the general runtime expression.

5.1 A General Runtime Bound

We formalise these ideas. With regard to a given metric space, we say that a set P covers a set C if $\text{co}(P) \supseteq C$.

Definition 3. Given a metric space $M = (S, d)$, let \mathbf{C}_M be the set of all geodesically convex sets on M . We define $P_M^{\text{Cov}}(m)$ as the worst-case probability that the convex hull of the set P of m points drawn uniformly at random from a geodesically convex set $C \in \mathbf{C}_M$ covers entirely the set C :

$$P_M^{\text{Cov}}(m) = \min_{C \in \mathbf{C}_M} \Pr(\text{co}(P) = C \mid P = U_m(C)).$$

Note that the worst-case covering probability is monotone in the number of samples m as additional samples can only increase the convex hull:

$$\forall m' \geq m: P_M^{\text{Cov}}(m') \geq P_M^{\text{Cov}}(m).$$

The following lemma gives a lower bound on the probability that the next fitness level is covered completely.

Lemma 4. Assume fitness levels L_0, \dots, L_q where $r = \min\{|L_{i+1}|/|L_i| \mid 0 \leq i < q\}$. Consider a (parent) population P'_t and suppose for some fitness level $0 \leq i < q$ we have $\text{co}(P'_t) = L_i$. Let the new population P_{t+1} result from μ -times Metric Convex Hull Uniform Recombination of P'_t and let $P'_{t+1} = \text{sel}(P_{t+1})$ be the next parent population. The probability that $\text{co}(P'_{t+1}) = L_j$ for some $i < j \leq q$ is at least

$$P_M^{\text{Cov}}\left(\frac{\mu r}{4}\right) - \exp\left(-\frac{9\mu r}{32}\right).$$

Proof. Let j be the largest index such that $P_{t+1} \subseteq L_{j-1}$. (Typically $j = i + 1$.) Note that all offspring are still distributed uniformly in L_{j-1} . And as $P_{t+1} \cap (L_{j-1} \setminus L_j) \neq \emptyset$, selection will remove all points in $L_{j-1} \setminus L_j$, leading to $\text{sel}(P_{t+1}) \subseteq L_j$. The probability of covering L_j is at least $P_M^{\text{Cov}}\left(\frac{\mu r}{4}\right)$, provided that at least $\frac{\mu r}{4}$ offspring are in L_j .

The probability for this event can be estimated by standard Chernoff bounds [23]. The expected number of offspring in L_j is $\mu|L_j|/|L_{j-1}| \geq \mu r$. The probability that less than $\mu r/4$ offspring fall in L_j is at most

$$\exp(-\mu r \cdot (3/4)^2 \cdot 1/2) = \exp\left(-\frac{9\mu r}{32}\right).$$

The claim then follows by the union bound. \square

Lemma 4 easily generalises, using that all probabilities for covering different fitness levels are independent and taking the union bound for error probabilities $\exp(-\frac{9\mu r}{32})$. We also take into account the probability of covering L_0 in the initialisation. It is at least $P_M^{\text{Cov}}(\mu) \geq P_M^{\text{Cov}}(\frac{\mu r}{4})$.

Theorem 5. *Assume a quasi-concave fitness function on any metric space M with fitness levels L_0, \dots, L_q where $r = \min\{|L_{i+1}|/|L_i| \mid 0 \leq i < q\}$. The Convex Search Algorithm with population size μ finds a global optimum within q generations and μq fitness evaluations with probability at least*

$$\left(P_M^{\text{Cov}}\left(\frac{\mu r}{4}\right)\right)^{q+1} - q \cdot \exp\left(-\frac{9\mu r}{32}\right).$$

5.2 Boolean Spaces & Hamming Distance

We specialise the general runtime result to the space of binary strings endowed with the Hamming distance HD. The specialised result is the runtime of the convex search algorithm specialised to this space (the convex search algorithm with the binary uniform convex hull operator, Algorithm 2) on the quasi-concave landscape specialised to this space (the class of binary landscapes presented in section 3.2).

We first bound the probability of covering any set.

Lemma 6. *For $M = (\{0, 1\}^n, \text{HD})$ and $m \in \mathbb{N}$ we have*

$$P_M^{\text{Cov}}(m) \geq (1 - 2^{-m+1})^n.$$

Proof. For every geodesically convex set $C \in \mathbf{C}_M$ for the specific case of $M = (S, \text{HD})$ a necessary requirement for the convex hull of m points uniformly drawn in C not covering C is that there is a bit that is fixed to the same value in all m samples. The probability for fixing a specific bit is 2^{-m+1} as the bit needs to be set to either 0 or 1 in all m uniform samples. The probability that no bit is fixed is $(1 - 2^{-m+1})^n$. \square

Then the bound from Theorem 5 simplifies as follows.

Theorem 7. *Let $M = (\{0, 1\}^n, \text{HD})$ and the assumptions for Theorem 5 hold. Then the Convex Search Algorithm with population size μ finds a global optimum within at most q generations and μq fitness evaluations with probability at least*

$$\begin{aligned} & \left(1 - 2^{-\frac{\mu r}{4} + 1}\right)^{n(q+1)} - q \cdot \exp\left(-\frac{9\mu r}{32}\right) \\ & \geq 1 - (2q + 3)n \cdot 2^{-\frac{\mu r}{4}}. \end{aligned}$$

where the second bound requires $\mu r \geq 4$.

Proof. The first bound follows directly from Theorem 5 by plugging in Lemma 6. For the second bound we use Bernoulli's inequality $((1-x)^y \geq 1-xy$ for $x \leq 1$ and $y \in \mathbb{N}_0$) along with $2^{-\frac{\mu r}{4}+1} \leq 1$ to estimate $(1-2^{-\frac{\mu r}{4}+1})^{n(q+1)} \geq 1-2(q+1)n \cdot 2^{-\frac{\mu r}{4}}$. We also have $2^{-\mu r/4} \geq e^{-9\mu r/32}$ since $2^{-1/4} \geq e^{-9/32}$. Further, $q \leq n$ as at least one bit is fixed with every fitness level. So $q+2(q+1)n \leq (2q+3)n$. Putting everything together gives the second bound. \square

In particular, we can easily derive how large the population must be in order to guarantee that Convex Search Algorithm with restarts finds a global optimum. If $\mu \geq 4 \log((4q+6)n)/r$, the probability bound from Theorem 5 is at least $1/2$. This gives the following.

Corollary 8. *Under the conditions of Theorem 7, if $\mu \geq 4 \log((4q+6)n)/r$ then Convex Search Algorithm with population size μ , restarting after q generations, finds a global optimum within $2q$ expected generations and $2\mu q$ expected fitness evaluations.*

On polynomial quasi-concave landscapes, as both q and $\frac{1}{r}$ are polynomial in n , the number of generations and the population size are polynomial. For LEADINGONES $q = n$ and $r = 1/2$, so with $\mu \geq 8 \log((4n+6)n) \approx 16 \log n$ the expected number of generations with restarts is at most $2n$.

Corollary 9. *Convex Search Algorithm with population size $\mu = 8 \log((4n+6)n)$ and restarts optimises LEADINGONES in $O(n \log n)$ function evaluations.*

This is remarkable as EAs using only mutation need at least $\Omega(n^2)$ function evaluations [34]. The same holds for all unary unbiased black-box algorithms [16], whereas binary black-box algorithms run in time $O(n \log \log n)$ [1].

5.3 Integer Vectors & Hamming and Manhattan Distances

We give two more examples of specialisation of the general runtime result. We consider two metric spaces on the same set: the Hamming distance HD and the Manhattan distance MD on integer vectors $S_d = \{0, 1, \dots, d-1\}^n$. The latter space is a natural space for integer optimisation problems, and it can be understood as a discretisation of a continuous space endowed with Manhattan distance. The former space is a natural space e.g. for colouring problems, in which each number is interpreted as a symbol identifying a colour rather than a discrete quantity or an ordinal item in an ordered set. The geometric crossovers associated to both spaces correspond to established crossovers.

Before presenting the specialisation of the runtime, we explain the specialisation to these spaces in order to understand the space-specific forms of the convex search algorithm and the class of quasi-concave landscapes. Both metrics are product metric spaces; distances between two vectors can be written as a dimension-wise sum of (uni-dimensional) distances between their elements in each dimension. The unidimensional distance corresponding to the Manhattan distance is the absolute value of the difference between two integers, and for the Hamming distance is the "discrete" distance over a set of integers, which is 1 for any two different integers, and 0 otherwise. The product structural property of these spaces allows us to build geometric elements (e.g., segments) in these spaces by doing the cartesian product of uni-dimensional segments across all dimensions.

Space structure: For (S_d, HD) , the neighbourhood graph associated with each dimension is a clique with d nodes as any two non-coinciding integers (i.e, nodes) are at distance one (i.e., they are direct neighbours and connected by an edge). The neighbourhood structure of (S_d, HD) is the cartesian product of clique graphs across dimensions. For (S_d, MD) , the neighbourhood graph associated with each dimension is a line

graph of d nodes (i.e., an integer line ranging from 0 to $d - 1$). The neighbourhood structure of (S_d, MD) is the cartesian product of line graphs across dimensions (i.e., “discretised” hyper-boxes).

Segment: For (S_d, HD) , unidimensional segments are edges, i.e., degenerate segments that are made of only the end-points of the segments. For (S_d, MD) , unidimensional segments are integer intervals delimited by the end-points of the segments. The segments in (S_d, HD) and (S_d, MD) are the cartesian product of the unidimensional segments across dimensions. E.g., for (S_d, HD) , the segment between $(0, 2, 1)$ and $(2, 2, 0)$ are the vectors obtained by $\{0, 2\} \times \{2\} \times \{0, 1\} = \{(0, 2, 0), (0, 2, 1), (2, 2, 0), (2, 2, 1)\}$. For (S_d, MD) , the segment between the same vectors is obtained by $\{0, 1, 2\} \times \{2\} \times \{0, 1\} = \{(0, 2, 0), (0, 2, 1), (1, 2, 0), (1, 2, 1), (2, 2, 0), (2, 2, 1)\}$.

Geometric crossover: A geometric crossover on product spaces can be seen as the vector-wise aggregation of geometric crossovers acting on each dimension, each one defined on the distance associated with each dimension [21]. So, for (S_d, HD) , any mask-based crossover on integer vectors is a geometric crossover, as a segment in each dimension is an edge (i.e., it comprises only the end-points of the segment). For (S_d, MD) , geometric crossovers are component-wise blend-crossovers in which each position in the offspring can take intermediate integer values between the values of parents at that position, as a segment in each dimension is the integer interval bounded by the values of the end-points.

Convex set: A convex set on a product space is the cartesian product of unidimensional convex sets across dimensions. For (S_d, HD) , for a dimension, the set of all (unidimensional) convex sets are all sub-cliques of the neighbourhood graph of that dimension. Convex sets on the entire space can be represented as generalised schemata in which there are special symbols indicating all possible superpositions of values in $\{0, 1, \dots, d - 1\}^n$. E.g., on an alphabet with values in $\{0, 1, 2\}$ generalised schemata use the values 0, 1, 2 and the special symbols $*_{01}, *_{02}, *_{12}, *_{012}$, each one denoting the superposition of the values appearing in its index. For example, the generalised schema $0 *_{01} *_{02} 2$ matches the vectors 0002, 0102, 0022, 0122. For (S_d, MD) , like for the Hamming space on binary strings, convex sets coincide with segments, i.e., they are integer hyper-boxes. Notice instead that in (S_d, HD) not all convex sets are segments.

Quasi-concave landscapes: We can derive the specific class of quasi-concave landscapes for any space by piling up a nested chain of convex sets. For (S_d, HD) , this translates to creating functions using a recursive construction analogous to the one in section 3.2 for the Hamming space on binary strings but using generalised schemata instead. Notice however, that unlike the case of binary strings, not every successive level fixes a “*” symbol to specific value, but it may simply reduce the degree of freedom of a “*” symbol, e.g., “fixing” $*_{012}$ to $*_{01}$. For example, the function `LeadingTwos` defined on strings with alphabet $\{0, 1, 2\}$ that counts the number of leading twos in the string is quasi-concave on (S_3, HD) as it can be built by piling up the generalised schemata $*_{012} *_{012} \dots *_{012}, 2 *_{012} \dots *_{012}, 22 \dots *_{012}, \dots, 22 \dots 2$. For (S_d, MD) , piling up a nested chain of convex sets translates into piling up hyper-boxes from the largest to the smallest. E.g., on two-dimensional vectors, these landscapes are traditional quasi-concave functions with rectangular or squared levels, whose domain is restricted to integer values.

Convex hull: We can derive the convex hull from its definition and the notion of convex set. For (S_d, HD) , the convex hull of a set of integer vectors is the most specific generalised schema matching all vectors. For (S_d, MD) , the convex hull of a set of integer vectors (i.e., points) is the smallest hyper-box covering all points.

Uniform convex hull recombination: We can derive this operator from the notion of convex hull. For (S_d, HD) , it is the operator that at each position returns, with the same probability, any value that occurs in that position in the parents at least once. This corresponds to sampling uniformly the most specific generalised schemata matching all parent vectors. For (S_d, MD) , it is the operator that at each position (i.e., dimension) returns with the same probability any value in the interval between the minimum and maximum values of the parents in that position. This corresponds to sampling the hyper-box uniformly, as this can be achieved by sampling uniformly at random each coordinate (within the specified range that allows to cover all parents).

We now specify the runtime to the space (S_d, HD) .

Lemma 10. *For $M = (S_d, \text{HD})$ and $m \in \mathbb{N}$ we have*

$$P_M^{\text{Cov}}(m) \geq 1 - dn \left(1 - \frac{1}{d}\right)^m \geq 1 - dn \cdot \exp\left(-\frac{m}{d}\right).$$

Proof. The worst case is attained for covering the whole space S_d . We cover S_d if in every component of the integer vector every value is present in at least one sample. For each dimension we need all values because to cover a clique graph (i.e., a unidimensional subspace of S_d) we need all nodes (i.e., values) in the clique.

For a fixed component this corresponds to the well-known coupon collector's problem: we are drawing m coupons (integer values) uniformly at random and are interested in the probability that we have at least one coupon of each kind.

This gives rise to the following tail bound. The probability of one fixed value not appearing in m draws is $(1 - 1/d)^m$. Taking the union bound over d values, the probability that there is a value not appearing in m draws is at most $d \cdot (1 - 1/d)^m$. Taking the union bound over n processes on all bits, we get a probability of at most $dn \cdot (1 - 1/d)^m$ that S_d is not covered. \square

Then the bound from Theorem 5 simplifies as follows, the second bound being derived as in Theorem 7, using Bernoulli's inequality and $\exp(-\frac{1}{4d}) \geq \exp(-\frac{1}{4}) \geq \exp(-\frac{9}{32})$ to subsume the second subtrahend.

Theorem 11. *Let $M = (\{0, 1, \dots, d-1\}^n, \text{HD})$ and the assumptions for Theorem 5 hold. Then Convex Search Algorithm with population size μ finds a global optimum within at most q generations and μq fitness evaluations with probability at least*

$$\begin{aligned} & \left(1 - dn \cdot \exp\left(-\frac{\mu r}{4d}\right)\right)^{q+1} - q \cdot \exp\left(-\frac{9\mu r}{32}\right) \\ & \geq 1 - (q+2)dn \cdot \exp\left(-\frac{\mu r}{4d}\right) \end{aligned}$$

where the second bound requires $\exp\left(\frac{\mu r}{4d}\right) \geq dn$.

We get a similar corollary as for Boolean spaces.

Corollary 12. *Under the conditions of Theorem 11, if $\mu \geq 4d \ln(2(q+2)dn)/r$ then Convex Search Algorithm with population size μ , restarting after q generations, finds a global optimum within $2q$ expected generations and $2\mu q$ expected fitness evaluations.*

On polynomial quasi-concave landscapes, if $d \leq \text{poly}(n)$, we get polynomial expected numbers of generations and fitness evaluations for appropriate population sizes.

For the space (S_d, MD) we get similar results.

Lemma 13. For $M = (S_d, \text{MD})$ and $m \in \mathbb{N}$ we have

$$P_M^{\text{Cov}}(m) \geq 1 - 2n \left(1 - \frac{1}{d}\right)^m \geq 1 - 2n \cdot \exp\left(-\frac{m}{d}\right).$$

Proof. The worst case is attained for covering S_d . We cover S_d if in every component of the integer vector both values 0 and $d - 1$ are present in at least one sample. The probability of one fixed value not appearing in m samples is $(1 - 1/d)^m$. The probability that there is a component and a value not appearing in m samples is at most $2n \cdot (1 - 1/d)^m$. \square

Theorem 14. Let $M = (\{0, 1, \dots, d - 1\}^n, \text{MD})$ and the assumptions for Theorem 5 hold. Then Convex Search Algorithm with population size μ finds a global optimum within at most q generations and μq fitness evaluations with probability at least

$$\begin{aligned} & \left(1 - 2n \cdot \exp\left(-\frac{\mu r}{4d}\right)\right)^{q+1} - q \cdot \exp\left(-\frac{9\mu r}{32}\right) \\ & \geq 1 - 2(q + 2)n \cdot \exp\left(-\frac{\mu r}{4d}\right) \end{aligned}$$

where the second bound requires $\exp\left(\frac{\mu r}{4d}\right) \geq 2n$.

Corollary 15. Under the conditions of Theorem 14, if $\mu \geq 4d \ln(4(q + 2)n)/r$ then Convex Search Algorithm with population size μ , restarting after q generations, finds a global optimum within $2q$ expected generations and $2\mu q$ expected fitness evaluations.

For polynomial quasi-concave landscapes we get the same conclusions as for the Hamming space on integer vectors.

6 Optimal Population Sizes for Convex Search Algorithm in Boolean Spaces

Our analysis has revealed certain population sizes that guarantee the efficiency of Convex Search Algorithm on quasi-concave problems. However, it is not clear whether these population sizes are really necessary, or whether in fact smaller population sizes would suffice. Since the number of function evaluations in one generation is equal to the population size, identifying minimal population sizes is key for efficient optimisation.

In this section we show that the population size can exhibit a threshold behaviour: if the population size decreases below the lower-bound of the population sizes given in Section 5 by more than a constant factor, the success probability can become exponentially small. This result shows that the population size is a crucial parameter in Convex Search Algorithm, and it also leads to a better understanding of this algorithm.

In order to make this point, it suffices to look at a particular metric space. In the following we consider $M_B = (\{0, 1\}^n, \text{HD})$, i. e., Boolean spaces with the Hamming distance metric. The analysis can be easily transferred to other product spaces.

For didactic reasons, in Section 6.1 we first give a simple and universal result on the population size that only depends on the problem size n , but does not take into account the difficulty of the problem (Theorem 16). In Section 6.2 we then present a refined result (Theorem 17) that establishes a threshold behaviour for the population size. In both cases we assume that there is only a single global optimum⁷.

⁷This is not an essential restriction: for functions with multiple global optima, $|L_q| > 1$, Theorem 16 remains valid when multiplying the bound on the probability of reaching L_q by $|L_q|$ (and making straight-

6.1 A Universal Threshold for the Population Size of CSA

The following lower bound on the running time shows that, for every n -bit problem, if the population size μ is less than $\log n$ by a constant factor, the running time of Convex Search Algorithm with restarts is exponential.

Theorem 16. *Consider Convex Search Algorithm with population size μ on $M_B = (\{0, 1\}^n, \text{HD})$ and a problem with a unique global optimum. Then the probability that the algorithm converges to the global optimum is at most*

$$e^{-n \cdot 2^{-\mu}}.$$

If $\mu \leq (1 - \varepsilon) \log n$ for some constant $\varepsilon > 0$ the above is at most e^{-n^ε} and the expected number of restarts needed to find the optimum is at least e^{n^ε} .

Proof of Theorem 16. The probability that during initialisation a specific bit is set to the “wrong” bit value (i. e., the bit value opposite of the one in the global optimum) in all offspring is $2^{-\mu}$. Then it is clear that the global optimum cannot be found. The probability that this event does not happen is at most

$$(1 - 2^{-\mu})^n \leq e^{-n \cdot 2^{-\mu}}.$$

The following statements follow from $n \cdot 2^{-\mu} \geq n \cdot 2^{-(1-\varepsilon) \log n} = n^\varepsilon$. □

Theorem 16 gives a universal result in that it applies to every problem. The lesson learnt is that population sizes less than $\log n$ (by a constant factor) always lead to inefficient running times.

Note that Theorem 16 does *not* claim that populations sizes above $\log n$ are efficient. In fact, the universality of Theorem 16 is also a weakness: the theorem does not consider the problem, yet difficult problems may demand a population size larger than $\log n$ for efficient optimisation. Recall from Section 5.2 that Corollary 8 gives an upper bound on the expected running time for CSA with restarts of $2\mu q$, given that $\mu \geq 4 \log((4q + 6)n)/r$, where r describes the worst-case size ratio between adjacent levels: $|L_{i+1}|/|L_i| \geq r$ for all $0 \leq i < q$. For polynomial numbers of fitness levels q , the population size threshold simplifies to $4 \log((4q + 6)n)/r = \Theta((\log n)/r)$.

The parameter r may be regarded, loosely speaking, as describing the difficulty of the problem. The smaller r , the more difficult it is to locate a better fitness level (at least for the most difficult fitness levels as a problem might contain fitness levels of varying difficulty). The function LEADINGONES is an easy function as here $r = 1/2$, the largest value possible. But other polynomial quasi-concave landscapes may have smaller values, such $r = n^{-c}$ for a constant $c > 0$, in which case the above population size threshold would be $\Theta((\log n)/r) = \Theta(n^c \log n)$, hence asymptotically way larger than $\log n$.

So far we only know that CSA with restarts is inefficient for $\mu \leq (1 - \varepsilon) \log n$ and efficient if $\mu = \Omega((\log n)/r)$. For small r , $r = o(1)$, there is an asymptotic gap between these two realms, leaving open the question of whether population sizes within this gap are sufficient or not. The following subsection answers this question by showing that population sizes of order $(\log n)/r$ are indeed necessary under certain conditions. This result takes the “problem difficulty” r into account and improves upon Theorem 16 in cases where $r = o(1)$.

forward adjustments to the following statements). This is justified by the union bound: the probability of finding one of $|L_q|$ optima is at most $|L_q|$ times as large as the probability of finding one specific optimum. We conjecture that the same modification would also work for Theorem 17, but its complicated proof makes this less obvious.

6.2 Tight Bounds on the Population Size

In order to derive tight bounds on the population size, we need to use more detailed considerations. The proof of Theorem 16 relies on the initialisation only, and that the initial population of Convex Search Algorithm is confined to a subspace that does not contain the optimum. But this confinement to wrong subspaces can happen in any generation of Convex Search Algorithm, as long as the optimum has not been found. The only difference to the situation at initialisation is that during the course of optimisation bits may have converged to the correct bit value. This decreases the number of random experiments that may go wrong.

In order to include the difficulty of the problem at hand, given by the size ratio between fitness levels, for the purpose of showing a negative result we define the best-case size ratio s such that $|L_{i+1}|/|L_i| \leq s$ for all $0 \leq i < q$. This is analogous to the worst-case size ratio r used in our positive results.

The following theorem—the main result of this section—gives a better bound on the probability that Convex Search Algorithm finds a global optimum. This bound will turn out to be strong enough to show that population sizes of $\mu = \Omega((\log n)/s)$ are necessary for efficient optimisation, under certain conditions.

Theorem 17. *Consider the Convex Search Algorithm with population size μ on the metric space M_B with fitness levels L_1, \dots, L_q such that $|L_{i+1}|/|L_i| \leq s$ for all $1 \leq i < q$, and L_q contains a unique global optimum. Then the probability that a single run converges to L_q is at most*

$$e^{-\Omega(q \cdot 4^{-\mu s})}.$$

If $\mu \leq (1 - \varepsilon) \log(q)/(2s)$ for some constant $\varepsilon > 0$ the above is at most $e^{-\Omega(q^\varepsilon / \log(\mu))}$ and the expected number of restarts needed to find the optimum is at least $e^{\Omega(q^\varepsilon / \log(\mu))}$.

For polynomial quasi-concave landscapes we have $s \geq r \geq 1/\text{poly}(n)$ (by definition of r and s) and $q = n^{\Omega(1)}$ (as otherwise, by the pigeon-hole principle, there must be two subsequent fitness levels with a size ratio of $2^{-n/q}$, contradicting $r \geq 1/\text{poly}(n)$). In this case, and if additionally $s = \Theta(r)$, the above threshold of $(1 - \varepsilon) \log(q)/(2s)$ is only by a constant factor smaller than the threshold from Corollary 8, above which good performance is guaranteed. This gives the following threshold result, distinguishing between polynomial and exponential running times.

Theorem 18. *Consider Convex Search Algorithm with population size μ and restarts on the metric space M_B with fitness levels L_1, \dots, L_q such that $r \leq |L_{i+1}|/|L_i| \leq s$ for all $1 \leq i < q$, and L_q contains a unique global optimum. If $s = \Theta(r)$, $r \geq 1/\text{poly}(n)$ and $q = n^{\Omega(1)}$ then there are constants c_l, c_u such that the expected number of generations is bounded as follows.*

- if $\mu \leq c_l \log(qn)/r$ the expected running time is at least $e^{n^{\Omega(1)}}$, i. e. exponential in n
- if $\mu \geq c_u \log(qn)/r$ the expected running time is at most $2q$.

Proof. For the first statement, Theorem 17 yields that if $\mu \leq (1 - \varepsilon) \log(q)/(2s)$ then the expected number of restarts, and hence the expected number of generations, is at least $e^{\Omega(q^\varepsilon / \log(\mu))}$. Since $q = n^{\Omega(1)}$ and $\mu \leq \log(q)/(2s) \leq \text{poly}(n)$, the exponent is $\Omega(n^{\Omega(1)}/O(\log n)) = n^{\Omega(1)}$.

The condition $\mu \leq (1 - \varepsilon) \log(q)/(2s)$ is implied by $\mu \leq c_l \log(qn)/r$ using $s = \Theta(r)$, $\log(q) = \Theta(\log(q) + \log(n)) = \Theta(\log(qn))$, and choosing c_l small enough such that $c_l \log(qn)/r \leq (1 - \varepsilon) \log(q)/(2s)$ for some $\varepsilon > 0$.

The second statement follows from Corollary 8, choosing c_u large enough such that $c_u \log(qn) \geq 4 \log(2(q + 2)n)$. \square

The remainder of this section is devoted to the proof of Theorem 17. The main proof idea is that with a small population size Convex Search Algorithm may find itself not covering the current fitness level with the current population's convex hull. Then we speak of a *misalignment*, formally defined as follows.

Definition 19. Consider a run of Convex Search Algorithm on a metric space with fitness levels L_1, \dots, L_q . We say that the run leads to a misalignment on fitness level L_i if Convex Search Algorithm during its run arrives at a population P such that either $\text{co}(P) \subsetneq L_i$ and $\text{co}(P) \setminus L_{i+1} \neq \emptyset$ or if for the initial population P_0 we have $\text{co}(P_0) \subsetneq L_i$.

The following lemma states that each misalignment runs the risk of losing a unique global optimum from the convex hull of the current population. This happens with probability at least $1/2$.

Lemma 20. Consider the metric space M_B and fitness levels L_1, \dots, L_q where L_q contains a single optimum. Whenever Convex Search Algorithm reaches a new, misaligned population P on some fitness level L_i , then with probability at least $1/2$ it holds that $\text{co}(P) \cap L_q = \emptyset$.

Proof. First assume that we only knew that $\text{co}(P) \subsetneq L_i$ (i. e., we ignore the additional condition $\text{co}(P) \setminus L_{i+1} \neq \emptyset$ for the time being). Compared to the previous population, creating P has led to at least one bit being fixed to a particular value in all search points. This also applies if P is the initial population. With probability $1/2$ this value is different from the global optimum, and then $\text{co}(P) \cap L_q = \emptyset$.

The additional condition $\text{co}(P) \setminus L_{i+1} \neq \emptyset$ can only increase this probability as it excludes cases where $\text{co}(P) \subseteq L_{i+1}$. This completes the proof. \square

What is left to show is that Convex Search Algorithm experiences many misaligned populations if the population size is too small. As a first step in this direction, we consider the case where a fitness level L_i is covered completely. We estimate the probability that the algorithm encounters a misaligned population on the next fitness level L_{i+1} .

Lemma 21. Assume for the current population P of size μ and some fitness level $i < q - 1$ we have $\text{co}(P) = L_i$. Then for the Convex Search Algorithm the following holds. Assume $|L_i| \geq 2^2$ (L_i contains at least two bits). The probability that the algorithm will eventually reach a population P' with $\text{co}(P') \subsetneq L_{i+1}$ and $\text{co}(P') \setminus L_{i+2} \neq \emptyset$ is at least

$$\min\{\Omega(1), 4^{-\mu|L_{i+1}|/|L_i|}\}.$$

Proof. Abbreviate $\ell_i := |L_i|$. We first assume that $(\frac{e}{4})^{\mu\ell_{i+1}/\ell_i} \leq 1/6$. The following conditions, referring to the generation evolving the population P , are sufficient for creating a population P' where $\text{co}(P') \subsetneq L_{i+1}$ and $\text{co}(P') \setminus L_{i+2} \neq \emptyset$:

1. in the next generation at least one offspring is in $L_i \setminus L_{i+1}$ and at least one offspring is in $L_{i+1} \setminus L_{i+2}$ and
2. in the next generation at most $m_i := 2\mu\ell_{i+1}/\ell_i$ offspring are in L_{i+1} and
3. assuming at most m_i offspring uniformly chosen from L_{i+1} , these offspring do not cover L_{i+1} completely.

The first event guarantees that selection will remove all points in $L_i \setminus L_{i+1}$. The probability of having no offspring in $L_{i+1} \setminus L_{i+2}$ is at most

$$\left(1 - \frac{\ell_{i+1} - \ell_{i+2}}{\ell_i}\right)^\mu \leq \left(1 - \frac{\ell_{i+1}}{2\ell_i}\right)^\mu \leq e^{-\mu\ell_{i+1}/(2\ell_i)}.$$

The probability of having no offspring in $L_i \setminus L_{i+1}$ is at most

$$\left(1 - \frac{\ell_i - \ell_{i+1}}{\ell_i}\right)^\mu \leq \left(1 - \frac{1}{2}\right)^\mu = 2^{-\mu}.$$

So, by the union bound the probability of the first event is at least

$$1 - 2^{-\mu} - e^{-\mu\ell_{i+1}/(2\ell_i)} \geq 1 - 2 \cdot \left(\frac{e}{4}\right)^{\mu\ell_{i+1}/\ell_i}.$$

The probability of the second event is again estimated by standard Chernoff bounds. The probability that less than twice the expected number of offspring fall in L_{i+1} is at least

$$1 - \left(\frac{e}{4}\right)^{\mu\ell_{i+1}/\ell_i}.$$

The probability of the third event is at least $2^{-2\mu\ell_{i+1}/\ell_{i+1}}$ as a sufficient condition is that there is one bit position on which all at most m_i offspring have the same value.

Taking the union bound for the first two events and noting that the third event is independent from these, we get a lower probability bound for the described event of

$$2^{-2\mu\ell_{i+1}/\ell_{i+1}} \cdot \left(1 - 3 \cdot \left(\frac{e}{4}\right)^{\mu\ell_{i+1}/\ell_i}\right) \geq 4^{-\mu\ell_{i+1}/\ell_i}$$

where in the last inequality we used $\left(\frac{e}{4}\right)^{\mu\ell_{i+1}/\ell_i} \leq 1/6$

If $\left(\frac{e}{4}\right)^{\mu\ell_{i+1}/\ell_i} > 1/6$ we use a different argument. The above condition implies $\mu\ell_{i+1}/\ell_i < 6$. That is, at most a constant number of offspring fall in L_{i+1} in each generation, in expectation. If in one generation no offspring falls in L_{i+1} , we continue our considerations with the next population. We consider the first generation where there is at least one offspring in L_{i+1} . Let X denote this number and $p := \ell_{i+1}/\ell_i$, then

$$\begin{aligned} \Pr(X = 1 \mid X \geq 1) &= \frac{\Pr(X = 1)}{\Pr(X \geq 1)} \\ &= \frac{\mu p (1-p)^{\mu-1}}{1 - (1-p)^\mu} \\ &\geq \frac{\mu p (1-p)^\mu}{1 - (1-\mu p)} = (1-p)^\mu. \end{aligned}$$

Note that $0 < p \leq 1/2$ as L_{i+1} is a proper subset of L_i . We also have $(1-p)^{1/p-1} \geq e^{-1}$ for $0 < p \leq 1$. Thus,

$$\begin{aligned} (1-p)^\mu &= (1-p)^{(1/p-1) \cdot \mu p + \mu p} \\ &\geq e^{-\mu p} \cdot (1-p)^{\mu p} \geq (2e)^{-\mu p} = \Omega(1). \end{aligned}$$

Hence, the first time the algorithm reaches L_{i+1} , with a conditional probability of $\Omega(1)$ there is exactly one offspring in L_{i+1} . Furthermore, independently of this, this offspring will be in $L_{i+1} \setminus L_{i+2}$ with probability at least $1/2$. In this case the population will have converged to a single point in $L_{i+1} \setminus L_{i+2}$. This establishes $\Omega(1)$ as lower probability bound. \square

We also need to bound the probability of Convex Search Algorithm advancing by many fitness levels in a single step. We say that a population P is on level i if $\text{co}(P) \subseteq L_i$ and $\text{co}(P) \setminus L_{i+1} \neq \emptyset$. Now assume the current population is on level i and consider the population reached after Convex Search Algorithm finds a better fitness level for the first time. Then the conditional probability of jumping to a level $j > i$ (or larger) is bounded as follows.

Lemma 22. *Consider Convex Search Algorithm with population size μ on any metric space with fitness levels L_1, \dots, L_q . Assume that the current population P is on level i . Consider the first population P' where $\text{co}(\text{sel}(P)) \subseteq L_{i+1}$ holds. Then for every $j > i$ we have*

$$\Pr(\text{co}(\text{sel}(P')) \subseteq L_j) \leq 1 - \left(1 - \frac{|L_j|}{|L_{i+1}|}\right)^\mu \leq \mu \cdot \frac{|L_j|}{|L_{i+1}|}.$$

Proof. A necessary event for $\Pr(\text{co}(\text{sel}(P')) \subseteq L_j)$ is that L_j is reached in the first place. The assumption on P' implies that at least one offspring must be in L_{i+1} . We only overestimate the probability of reaching L_j if we assume that all μ offspring lie in L_{i+1} . Since then all offspring are chosen uniformly and independently from L_{i+1} , the probability that L_j is not reached by any offspring is at most $(1 - |L_j|/|L_{i+1}|)^\mu$. This proves the first inequality.

The second inequality follows from the first by Bernoulli's inequality. \square

Taking the two previous lemmas together, we then get a pessimistic estimation of the number of misaligned populations Convex Search Algorithm goes through in a run, unless it has prematurely converged to a non-optimal point.

Lemma 23. *Consider Convex Search Algorithm with population size μ on the metric space M_B with fitness levels L_1, \dots, L_q , where $|L_{i+1}|/|L_i| \leq s$ for all $1 \leq i < q$. With probability $1 - e^{-\Omega(q/\log(\mu) \cdot 4^{-\mu s})}$ during a run the number of fitness levels with misalignments is stochastically dominated by a binomially distributed random variable with parameters $\Omega(q/\log(\mu))$ and $\Omega(4^{-\mu s})$, or Convex Search Algorithm converges to a non-optimal point.*

Proof. We may assume that for the initial population P_0 we have $\text{co}(P_0) = L_1$ as otherwise, if $\text{co}(P_0) \not\subseteq L_i$ for some i , the initial population already generates misalignments on all fitness levels $1, \dots, i$.

Now divide all fitness levels into blocks of length $b := \log(\mu) + 2$, i. e., L_1, \dots, L_b form a block, so do L_{b+1}, \dots, L_{2b} , etc. We claim that each block leads to at least one misaligned fitness level with probability $\Omega(4^{-\mu s})$. This implies the claim as the number of blocks is at least $\lfloor q/b \rfloor = \Omega(q/\log(\mu))$.

We first argue that for each block $L_{ib+1}, \dots, L_{ib+b}$ with probability at least $1/2$ the block will be reached, and it will first be reached on one of the levels $L_{ib+1}, \dots, L_{ib+b-1}$. In order for this not to happen, CSA would have to jump from a population P with $\text{co}(P) \supseteq L_{ib+1}$ to a population P' with $\text{co}(P') \subseteq L_{ib+b}$. Note that $|L_{ib+b}|/|L_{ib+1}| \leq 2^{-\log(\mu)-1} = 1/(2\mu)$. By Lemma 22 we have that this jump has probability at most $\mu \cdot 1/(2\mu) = 1/2$.

Assume that CSA reaches a population P on one of the levels $L_{ib+1}, \dots, L_{ib+b-1}$. If $\text{co}(P) \not\subseteq L_j$ and $\text{co}(P) \setminus L_{j+1} \neq \emptyset$ for some j , we have a misalignment. Otherwise, we have $\text{co}(P) = L_j$ and by Lemma 21 there will be a misalignment on level $j+1$ with probability at least $\min\{\Omega(1), 4^{-\mu|L_{j+1}|/|L_j|}\} = \Omega(4^{-\mu|L_{j+1}|/|L_j|}) = \Omega(4^{-\mu s})$. Since $j+1 \leq ib+b$, this will still count towards the considered block. Along with the probability $1/2$ from the previous paragraph, each block leads to a misalignment with probability $\Omega(4^{-\mu s})$, independent from other blocks. This implies the claim. \square

We also need the following closed formula for an event that occurs in a binomially distributed number of trials.

Lemma 24. *Consider an event that happens with probability α^X , where X is an independent, binomially distributed random variable with parameters m and p . Then the event occurs with probability*

$$(1 - (1 - \alpha)p)^m.$$

Proof. The probability of the event is

$$\begin{aligned} \sum_{x=0}^m \Pr(X = x) \cdot \alpha^x &= \sum_{x=0}^m \binom{m}{x} p^x (1-p)^{m-x} \cdot \alpha^x \\ &= \sum_{x=0}^m \binom{m}{x} (\alpha p)^x (1-p)^{m-x} \\ &= (1 - (1 - \alpha)p)^m \end{aligned}$$

where the last step follows from the binomial theorem. \square

Now we can finally put all these lemmas together to prove the promised result. Every misaligned population with probability at least $1/2$ excludes the optimum in the current population's convex hull. Along with our estimate of the number of misaligned populations, the probability of not excluding the global optimum in all misaligned populations is small if μ is small.

Proof of Theorem 17. Combining Lemma 23 with Lemma 20, we have that on each fitness level with a misalignment there is a probability of at least $1/2$ that the algorithm converges prematurely. Hence the probability of converging to a population P with $\text{co}(P) \cap L_q = \emptyset$ is at least $(1/2)^X$, where X is a binomially distributed random variable with parameters $\Omega(q/\log(\mu))$ and $\Omega(4^{-\mu s})$. Invoking Lemma 24, the probability of this premature convergence is at least

$$\left(1 - \frac{1}{2} \cdot \Omega(4^{-\mu s})\right)^{\Omega(q/\log(\mu))} = (1 - \Omega(4^{-\mu s}))^{\Omega(q/\log(\mu))} \geq 1 - \Omega(q/\log(\mu)) \cdot 4^{-\mu s}$$

by Bernoulli's inequality. This proves the claim. \square

6.3 Experiments for LeadingOnes

We conclude this section with a brief empirical study⁸ highlighting the average performance of CSA and comparing it against our theoretical results. Our theoretical results are very broad as they apply to arbitrary quasi-concave landscapes, and even different metric spaces. For the purpose of an empirical investigation we restrict our attention to the function LEADINGONES in Boolean spaces.

Figure 6 and Figure 7 show the average success rate of CSA on LEADINGONES with $n = 10$ and $n = 100$ bits, respectively. The plots also show upper and lower confidence intervals (95% confidence intervals on the binomial probabilities), the two theoretical lower bounds on the success probability from Theorem 7 and the universal upper bound on the success probability from Theorem 16. The refined probability bound from Theorem 17 is not considered here for two reasons: the latter is stronger

⁸Python implementation of the algorithm used here is available at <https://github.com/amoraglio>.

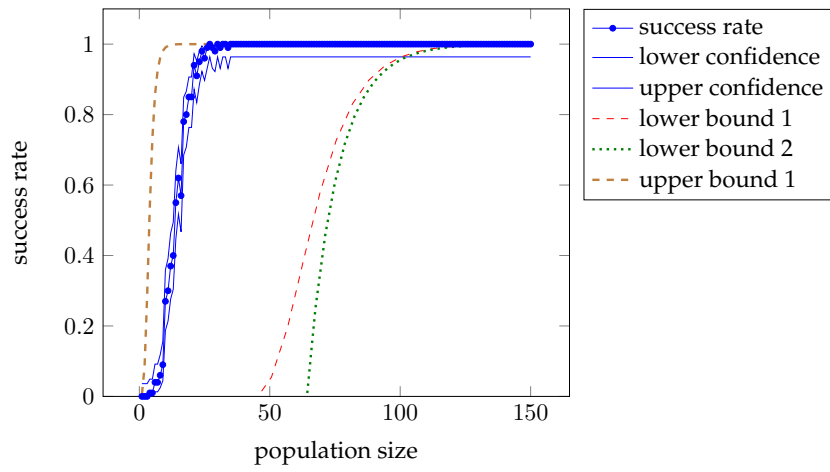


Figure 6: Average success rates for CSA on LEADINGONES with $n = 10$ in 100 runs. The plot also shows the two lower bounds on the success probability from Theorem 7 and the upper bound on the success probability from Theorem 16.

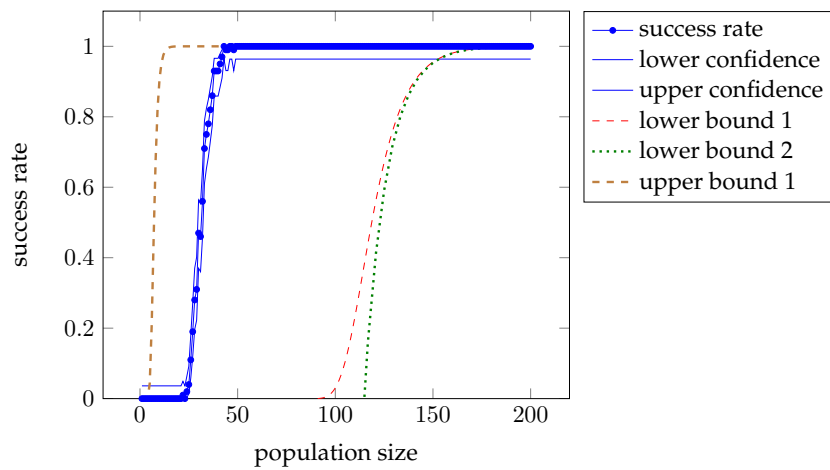


Figure 7: Average success rates for CSA on LEADINGONES with $n = 100$ in 100 runs. The plot also shows the two lower bounds on the success probability from Theorem 7 and the upper bound on the success probability from Theorem 16.

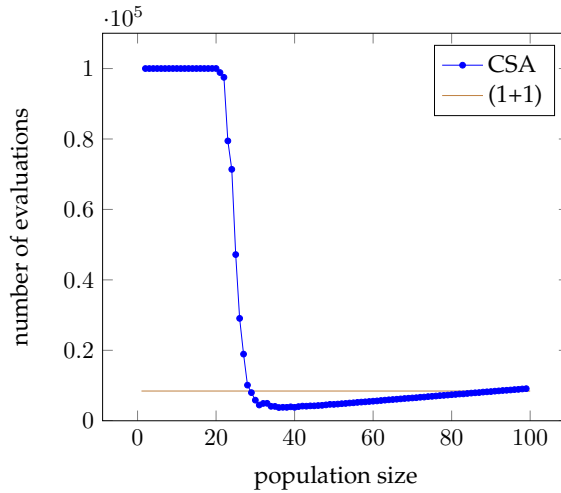


Figure 8: Average number of function evaluations for CSA of 100 runs with varying population sizes, for LEADINGONES with $n = 100$. The number of evaluations was capped at 10^5 . The plot also shows the expected number of evaluations for the (1+1) EA as a baseline.

than the universal bound only for difficult landscapes ($r = o(1)$), however LEADINGONES is the easiest quasi-concave landscape ($r = 1/2$). Hence, we do not expect to see an advantage for the refined bound. Secondly, the refined bound contains an implicit constant that we did not specify further.

As predicted by Theorem 18, we can clearly identify two realms for the choice of the population size: for small population sizes ($\mu \leq c_l \log(qn)/r$) the success rate is exponentially small, whereas for large population sizes ($\mu \geq c_u \log(qn)/r$) the success rate is at least $1/2$, converging to 1 as the population size grows.

Figures 6 and 7 also show that the second, simpler lower bound from Theorem 16 is not much worse than the first, more complex one. The gap in between the theoretical upper and lower bounds results from pessimistic arguments applied during the analysis. Our lower bounds on success probabilities rely on sufficient conditions for finding a global optimum, and the experiments show that CSA in this setting is more effective than indicated by these sufficient conditions. Likewise, the upper bounds on success probabilities are based on necessary events for finding a global optimum.

Figure 8 shows the average number of function evaluations of CSA with restarts to reach the optimum on LEADINGONES with $n = 100$. The number of function evaluations was capped at 10^5 . It also shows the expected number of function evaluations of the (1+1) EA on LEADINGONES, which was independently derived in [3, 32]. The (1+1) EA is provably the fastest evolutionary algorithm that only uses standard bit mutation for variation [34] (modulo an additive small-order term gained by initialising with the best of $1 < \mu = O(n \log n)$ individuals).

One can see that CSA outperforms the (1+1) EA on a range of population sizes, and how the performance increases steady as the population size grows too large. This highlights again the importance of choosing the right population size for CSA.

Figure 9 shows the average number of function evaluations of CSA with restarts to reach the optimum with increasing problem sizes on LEADINGONES. It also shows

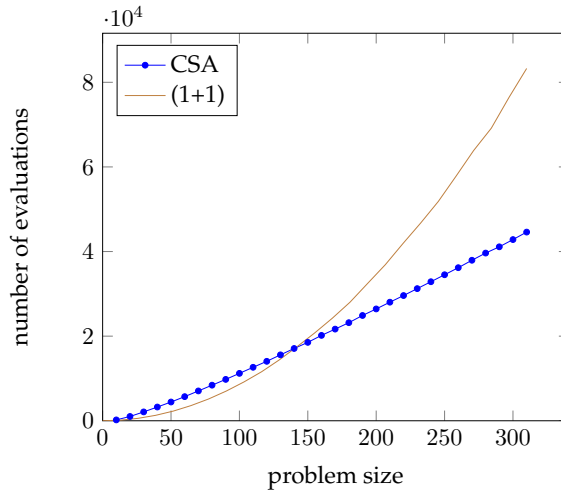


Figure 9: Average number of function evaluations for CSA of 100 runs with increasing problem sizes on LEADINGONES. The plot also shows the expected number of evaluations for the (1+1) EA.

the expected number of function evaluations of the (1+1) EA on LEADINGONES, which is $\Omega(n^2)$ [3, 32]. Following Corollary 9, the population size in the experiments was set to $8 \log_2(4n^2 + 6n)$ for which CAS with restarts optimises LEADINGONES in $O(n \log n)$ function evaluations.

One can see that for sufficiently large problem sizes CSA outperforms the (1+1) EA, and that it scales better with increasing problem size as expected. Again, this is remarkable as EAs using only mutation need at least $\Omega(n^2)$ function evaluations [34].

7 Summary and Future Work

Two important open challenges in the Evolutionary Computation field are (i) finding out on what class of landscapes a certain search algorithm performs well and why, and (ii) devising a more systematical approach to runtime analysis to obtain results that hold for *classes* of search algorithms on *classes* of problems. In this work, we have proposed a novel framework that is a first step towards addressing jointly both challenges. The framework put together a geometric theory of representations and runtime analysis to determine systematically the performance *across, in principle, all spaces and representations* of a population-based evolutionary algorithm with a form of population-wide convex recombination, the convex search algorithm, on a class of concave landscapes. Runtime analysis at this level of generality is possible because at its core relies on abstract properties of convex evolutionary search and of concave landscapes common to all underlying spaces and representations.

The general runtime expression is parametrised on a single space-dependent feature, the worst-case probability of covering convex sets. It can be determined on specific spaces and plugged in the general expression to determine the runtime of the space-specific pair of search algorithm and fitness landscape. We have illustrated this for three spaces and obtained polynomial runtime bounds for polynomial quasi-concave landscapes. We also showed that the convex search algorithm can be regarded

as pure adaptive search on concave landscapes. This is the ultimate cause behind the polynomial runtime of this algorithm on concave landscapes across representations.

Remarkably, for LEADINGONES the convex search algorithm turned out to be much faster than all unbiased unary black-box algorithms, including EAs using only mutation [16,34]. This further demonstrates the potential of convex evolutionary search and population-based EAs.

We also studied the population size as a key parameter for the running time of the convex search algorithm with restarts. For binary spaces we gave a universal lower threshold for the population size: for every n -bit problem with a unique optimum, a population size of up to $(1 - \varepsilon) \log n$, for any constant $\varepsilon > 0$, is inefficient as the algorithm typically converges to a non-optimal search point with high probability, leading to exponentially small success probabilities and exponential expected running times with restarts. This bound was then refined to give a threshold result that takes into account the difficulty of the problem. When ratios between all subsequent fitness levels are of order $\Theta(r)$, population sizes $\mu \geq c_u \log(qn)/r$ for some constant c_u lead to an expected number of generations of at most $2q$. However, decreasing the population size by a constant factor, towards $\mu \leq c_l \log(qn)/r$ for some constant c_l , leads to exponential expected running times. Our results from Section 5 give guidance as to how to choose the population size to guarantee good performance on a range of metric spaces.

There is plenty of future work, in at least three directions of investigation. The first direction is to extend the theory to encompass evolutionary algorithms used in practice with standard forms of crossover, selection and mutation. Pure adaptive search is an idealised algorithm which illustrates very clearly and in great generality that if we are able to sample uniformly at random successive level sets, we get to the optimum in polynomial time. Hesitant adaptive search is a more realistic variant of pure adaptive search that shows that we can relax the assumptions of uniform distribution and certainty of sampling at a successive level set at each generation while retaining the polynomial runtime. We believe that evolutionary algorithms with crossover and (small) mutation approximate the convex search algorithm well enough to meet the more relaxed conditions of hesitant adaptive search. The second direction is to extend the theory to encompass a larger family of fitness landscapes, in particular approximately concave landscapes, and study how the degree of approximation to concavity affects the runtime. From preliminary study, it seems that performance could degrade gracefully as the approximation becomes less good. Furthermore, we would like to determine to what extent interesting non-toy problems fit this framework. There seems to be hope as fitness landscapes associated with well-known combinatorial problems have been shown empirically to have some form of “global concavity”. We would like to develop an analytical methodology to recognise that a certain problem provably fits a certain class of concave landscapes. Clearly, NP-Hard problems can be expected to fit a polynomially solvable class of fitness landscapes only with respect to some relaxed target (e.g., approximation complexity, parametrized complexity). Finally, as third line of investigation, we would like to specialise the results to other representations (e.g., continuous and permutation spaces), and devise a general approach for easily determining the required space-specific convexity feature, using known results from convexity on graphs.

Acknowledgments: The authors thank Jon Rowe and a reviewer for helpful comments. This research was supported by EPSRC grants EP/I010297/1 and EP/D052785/1. The research leading to these results has received funding from the

European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 618091 (SAGE).

References

- [1] P. Afshani, M. Agrawal, B. Doerr, C. Doerr, K. G. Larsen, and K. Mehlhorn. The query complexity of finding a hidden permutation. In A. Brodnik, A. López-Ortiz, V. Raman, and A. Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of LNCS, pages 1–11. Springer, 2013.
- [2] A. Auger and B. Doerr, editors. *Theory of Randomized Search Heuristics – Foundations and Recent Developments*. World Scientific, 2011.
- [3] S. Böttcher, B. Doerr, and F. Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *11th International Conference on Parallel Problem Solving from Nature (PPSN 2010)*, volume 6238 of LNCS, pages 1–10. Springer, 2010.
- [4] D. Corus, D. Dang, A. V. Eremeev, and P. K. Lehre. Level-based analysis of genetic algorithms and other search processes. In *Parallel Problem Solving from Nature (PPSN 2014)*, pages 912–921. Springer, 2014.
- [5] K. De Jong. *Evolutionary computation: a unified approach*. The MIT Press, 2006.
- [6] B. Doerr, C. Doerr, and F. Ebel. From Black-Box Complexity to Designing New Genetic Algorithms. *Theoretical Computer Science*, 567(0):87–104, 2015.
- [7] B. Doerr, E. Happ, and C. Klein. Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425(0):17–33, 2012.
- [8] B. Doerr, D. Johannsen, and C. Winzen. Multiplicative drift analysis. In *Genetic and Evolutionary Computation Conference (GECCO '10)*, pages 1449–1456. ACM Press, 2010.
- [9] J. He and X. Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004.
- [10] T. Jansen. *Analyzing Evolutionary Algorithms – The Computer Science Perspective*. Springer, 2013.
- [11] T. Jansen and I. Wegener. On the analysis of evolutionary algorithms—a proof that crossover really can help. *Algorithmica*, 34(1):47–66, 2002.
- [12] T. Jansen and I. Wegener. Real royal road functions—where crossover provably is essential. *Discrete Applied Mathematics*, 149:111–125, 2005.
- [13] T. Kötzing, D. Sudholt, and M. Theile. How crossover helps in pseudo-Boolean optimization. In *Proc. of GECCO '11*, pages 989–996. ACM Press, 2011.
- [14] W. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.
- [15] P. Lehre and C. Witt. Concentrated hitting times of randomized search heuristics with variable drift. In *Proc. of 25th International Symposium on Algorithms and Computation (ISAAC 2014)*, volume 8889 of LNCS, pages 686–697. Springer International Publishing, 2014.

- [16] P. K. Lehre and C. Witt. Black-box search by unbiased variation. *Algorithmica*, 64(4):623–642, 2012.
- [17] B. S. Mitavskiy. *A Mathematical Model of Evolutionary Computation and Some Consequences*. PhD thesis, University of Michigan, USA, 2004.
- [18] A. Moraglio. *Towards a Geometric Unification of Evolutionary Algorithms*. PhD thesis, University of Essex, 2007.
- [19] A. Moraglio. Abstract convex evolutionary search. In *Proc. of FOGA '11*, pages 151–162. ACM Press, 2011.
- [20] A. Moraglio and R. Poli. Topological interpretation of crossover. In *Proc. of GECCO '04*, pages 1377–1388. Springer, 2004.
- [21] A. Moraglio and R. Poli. Product geometric crossover. In *Parallel Problem Solving from Nature (PPSN 2006)*, pages 1018–1027. Springer, 2006.
- [22] A. Moraglio and D. Sudholt. Runtime analysis of convex evolutionary search. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2012)*, 2012.
- [23] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [24] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer, 2010.
- [25] N. R. Patel, R. L. Smith, and Z. B. Zabinsky. Pure adaptive search in Monte Carlo optimization. *Mathematical Programming*, 4:317–328, 1988.
- [26] N. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–205, 1991.
- [27] C. M. Reidys and P. F. Stadler. Combinatorial landscapes. *SIAM Review*, 44:3–54, 2002.
- [28] F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer, 2002.
- [29] J. E. Rowe, M. D. Vose, and A. H. Wright. Group properties of crossover and mutation. *Evolutionary Computation Journal*, 10(2):151–184, 2002.
- [30] C. R. Stephens and A. Zamora. EC theory: A unified viewpoint. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1394–1405, 2003.
- [31] D. Sudholt. Crossover is provably essential for the Ising model on trees. In *Proc. of GECCO '05*, pages 1161–1167. ACM Press, 2005.
- [32] D. Sudholt. General lower bounds for the running time of evolutionary algorithms. In *11th International Conference on Parallel Problem Solving from Nature (PPSN 2010)*, volume 6238 of LNCS, pages 124–133. Springer, 2010.
- [33] D. Sudholt. Crossover speeds up building-block assembly. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2012)*, pages 689–696. ACM Press, 2012.

- [34] D. Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17(3):418–435, 2013.
- [35] P. D. Surry and N. J. Radcliffe. Formal algorithms + formal representations = search strategies. In *Proceedings of Parallel Problem Solving from Nature Conference*, pages 366–375, 1996.
- [36] M. van de Vel. *Theory of Convex Structures*. Elsevier, Amsterdam, 1993.
- [37] I. Wegener. Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In R. Sarker, X. Yao, and M. Mohammadian, editors, *Evolutionary Optimization*, pages 349–369. Kluwer, 2002.
- [38] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transaction on Evolutionary Computation*, 1(1):67–82, 1996.
- [39] Z. B. Zabinsky. *Stochastic Adaptive Search for Global Optimization*. Kluwer Academic, 2003.